

Introduction to Poseidon for UML

Summary

The Unified Modeling Language is rapidly becoming the lingua franca of the software development world – and with good reason. Its proven track record makes it desirable to any team, large or small.

Gentleware AG produces ‘Poseidon for UML’, a UML tool developed with a distinct difference. With the goals of usability and productivity, Poseidon has become a market leader with a large and loyal following.

This whitepaper is intended to introduce you to Poseidon for UML and illustrate its unique features.



Ludwigstrasse 12 • 20357 Hamburg • Germany

Tel: +49 (0)40 2442 5330 • Fax: +49 (0)40 2442 5331
Email: info@gentleware.com • Web: www.gentleware.com

Table of Contents

1	Do I Need UML?	3
2	Gentleware Philosophy	3
2.1	Human-Centric Development.....	3
3	The 'Poseidon for UML' Modeling Tool	4
3.1	Get to Work.....	4
3.1.1	Navigation.....	5
3.1.2	Settings	6
3.2	Code Generation	6
3.3	Documentation Generation	8
4	UML 2.0 Support	9
4.1	State Machine Diagrams.....	9
4.1.1	One Element, Multiple State Machine Diagrams.....	9
4.1.2	Orthogonal and Submachine States.....	10
4.2	Sequence Diagrams	11
4.2.1	Combined Fragments	11
4.2.2	Inspections	12
5	Features and Benefits	13
5.1	Business Analysts	14
5.2	Programmers	14
5.3	Project Managers	15
5.4	Software Architects	16

Figures

Figure 1	– Poseidon for UML interface.	5
Figure 2	– Settings tab.....	6
Figure 3	– Accessor method button.	7
Figure 4	– Code preview.....	7
Figure 5	– Documentation generation dialog.....	8
Figure 6	- A single use case 'Withdraw Money' and two associated State Machine Diagrams 'sd ATM Machine' and 'sd Customer Account' as seen in the Navigation Pane.	10
Figure 7	– A single state machine with two State Machine Diagrams.	10
Figure 8	– An orthogonal state with three regions and its rapid buttons activated.....	11
Figure 9	– Submachine state displaying entry and exit points.	11
Figure 10	– Sequence Diagram depicting a 'loop' combined fragment.	12
Figure 11	– Properties tabs for combined fragments and constraints.	12
Figure 12	– Sequence Diagram with activated inspection.	12

1 Do I Need UML?

A guy sits at a bar, chuckling to himself while he watches two women at a table try to explain the rules of baseball to a man using toothpicks, salt and pepper shakers, and sugar packets. "What a goofball," he thinks. "It's such a simple game, and he doesn't get it," and he turns back to the bar.

The man at the table asks a question about a subtle rule, and the guy at the bar realizes he doesn't know the answer, either. This game is more complicated than he realizes because he grew up just playing the game; he's never tried to explain it from scratch. He peeks over his shoulder to watch a sugar packet advance to second base as the pepper scores a run. "Oh, I see. Huh, that's interesting. I thought it would have been the salt at second."

Situationally, it is understood that the condiment menagerie represents assorted players. The mental picture formed from these objects clarified the problem and brought everyone to a common understanding.

This same principle holds true in the world of software development. Complex problems require explanations, and many times a graphic representation expresses a solution more elegantly and succinctly than text alone. In response to this idea, graphic notations were developed for several major development methods. Collaboration became difficult as methodologies were combined and translations of the software models became necessary.

Enter the 'Three Amigos': Grady Booch, Ivar Jacobson, and James Rumbaugh. All three had developed their own methods, but collaborated to combine them into the Unified Method, which evolved into the Unified Modeling Language (UML). With these initial UML releases, dozens of competing notations were replaced by the language- and method-independent UML.

The benefits of UML are undeniable, and adoption rates are sky-high. A study conducted in July of 2004 by BZ Research reported that over two-thirds of development managers say that UML is used within their organization. Just using UML is a start, now let Gentleware help you use UML well.

2 Gentleware Philosophy

A possibly inebriated poster on a forum once advocated the sole use of beer napkins for capturing UML. While the fun factor involved in this method is high and tech requirements nil, it just can't compete for readability, scalability, and maintainability. This doesn't mean shifting to a dull and dry program with all of the inherent frustrations of learning a new tool; Gentleware seeks to combine the power of technology with the joy of the creative process (beer not included).

2.1 Human-Centric Development

Software development is an inventive and human-centric process. It is people who decide what and how software is to be made. The interface between human and machine is the most important component of software development. If the human is illogical and ambiguous, the machine will not know how to respond; likewise if the computer is not accessible in human-understandable terms, the human is bogged down in technological details that hinder the process.

Gentleware creates software to bridge this gap with comfortable, flexible, and intuitive applications – the advantages of the pen and paper method coupled with efficient functionality. Each feature is designed with the human in mind – rapid buttons to speed up development, intelligent comments to assist with diagram design, etc. Also important is what is not there – the 'features' that slow down

the learning curve, make the tool unnecessarily complicated, and make only minute contributions to the final project.

Gadgets are great – nothing beats the childlike glee of getting a new tech toy and trying out all of the possibilities. How many different types of scene transition wipes come with that camcorder? What about the special effects like sepia tone or polarization in a digital camera? After the first run through of the user manual, they most often become extra menu options that get in the way after the novelty wears off.

And they can even be dangerous. Integrated GPS in your car sounds like a great help, but incorrect maps are notorious. The lovely soothing voice telling you to turn the wrong way on a one way street will suddenly become extremely irritating when she tells you to make an immediate U-turn and follow her instructions. If the dashboard controls aren't well designed, your attention can be diverted from the road while trying to turn her off and can result in an accident. This same theory applies to software development. Fumble around the menu options for a while, get lost in the details, and before you know it your deadline is approaching and you are staring into the headlights of oncoming traffic.

3 The 'Poseidon for UML' Modeling Tool

Poseidon for UML is, well, for UML. This single purpose helps you focus on your model, not your modeling technique. For instance, a new 'fad' is to include Business Process Modeling Notation (BPMN) within a UML tool. BPMN is extremely similar to a UML Activity Diagram – both use rounded rectangles for activities and solid lines to show flow, but the arrowheads in these notations are semantically different. In Poseidon, there is one clear meaning to the captured business process flow – no hesitating, trying to remember which arrowhead means what in which notation, consulting cheat sheets, etc. Concentrate on your model, not on your notation.

Whether you are a hard core programmer or a business analyst (or fall somewhere in between), Poseidon has something for you. The last section of this paper titled, "Features and Benefits" details how specific functionality of Poseidon correlates to specific job titles. But for now, we'll take a look at how Poseidon works in general.

3.1 Get to Work

Poseidon eliminates the distraction of tool complexities. Every aspect of the look and feel has been carefully considered and implemented with the user in mind. Toolbars are streamlined and compacted to give the user only the options that make sense in a given context. Dialogs are straightforward and informative. Rapid buttons automatically appear around an element for the most used functions. And much more.

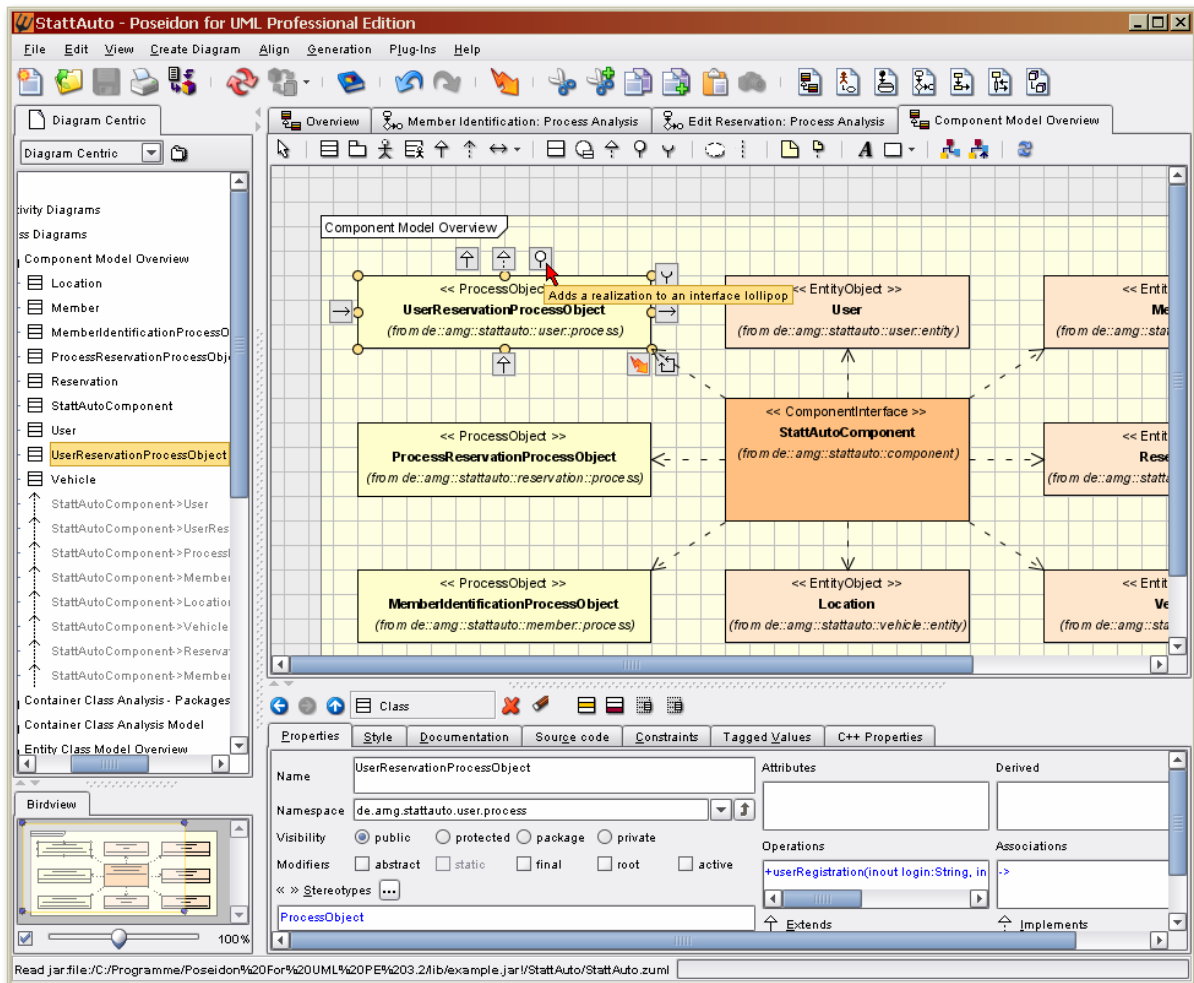


Figure 1 – Poseidon for UML interface.

3.1.1 Navigation

Move from one element to another from numerous areas of the workspace practically effortlessly without having to search for or open the correct palette or menu item. One click will select an element simultaneously in the Navigation Pane with its listing of model elements, in the Diagram Pane that displays the currently selected diagram, and in the Details Pane containing all information on the chosen element.

While working with the diagrams as sketches, while brainstorming, or any other task involving quick editing of the fundamental properties of an element such as the type of an attribute, inline editing directly in the diagram is speedy and efficient.

More nuanced editing is best accomplished through a series of tabs in the Details Pane, located directly below the diagram. Many of the properties here are also navigable as hyperlinks; a single click on an operation name within class properties opens the properties tab for that operation, for example.

3.1.2 Settings

You control what appears in the diagram. The display of stereotypes, multiplicity, etc. can be set at the diagram or model level. Additionally, different classifiers can be generated to and from different source files during roundtrip engineering. You have the freedom to adjust the tool to your work methods, rather than changing your style to fit the tool.

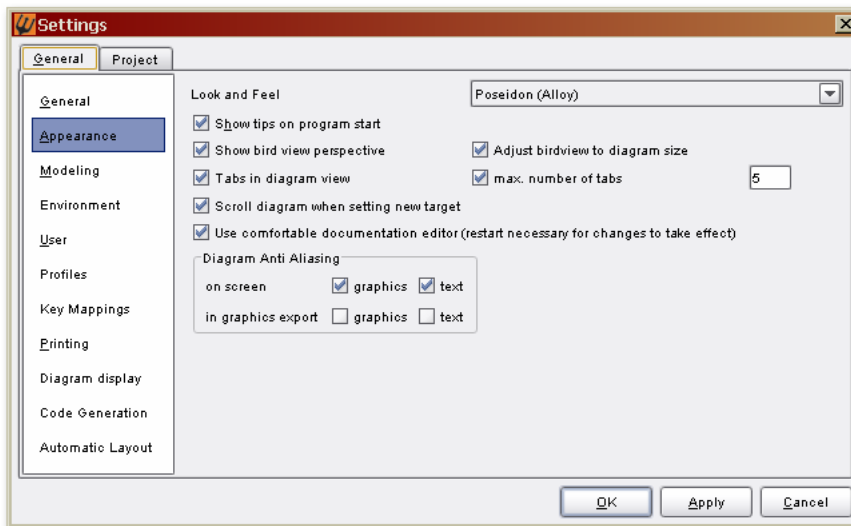


Figure 2 – Settings tab.

3.2 Code Generation

The ultimate goal of modeling is to actualize the system as quickly, efficiently and painlessly as possible. Pen and paper modeling facilitates the communication that is essential to the process, but the true power of a modeling tool commences with code generation. Depending on the level of model detail, generated code can range from a general code skeleton all the way to fully compiled and runnable code.

Even the most basic generated code skeleton can provide enormous leaps in productivity. Many of the minor tedious tasks can be eliminated, such as writing getter and setter methods. A single push of a button enters the code, saving time and reducing minor errors like typos. Furthermore, after inserting these methods Poseidon will keep track of the attribute name and automatically change the getters and setters whenever this name is changed in the model.

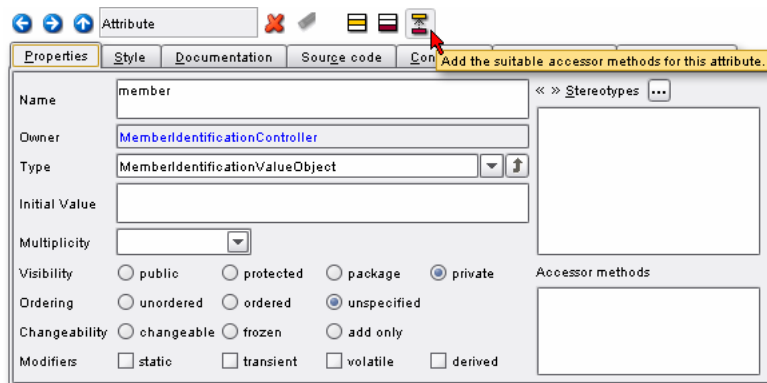


Figure 3 – Accessor method button.

A single model can generate code in a variety of languages; Poseidon for UML supports forward engineering for Java, PHP, Delphi, Perl, C++, C#, CORBA IDL, VB.net and SQL. A code preview tab allows you to directly edit the code from within Poseidon, and upon generation these edits are incorporated into the code. Modifications are saved for each language, so that, for example, a method body written in the C++ preview will not affect the Java version of that method. Switching between the two is simply a matter of a dropdown menu.

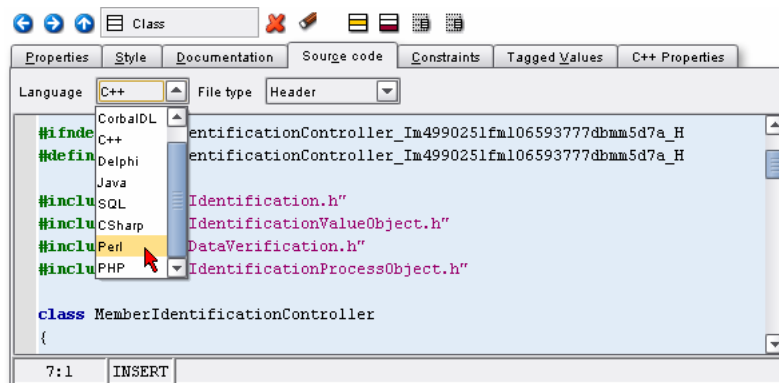


Figure 4 – Code preview.

A complete system can be developed in Poseidon, but quite often other development environments are used, including simple text editors. Existing Java code can be brought into the model by means of reverse engineering, whereby code is translated into a model. This is most often used to provide an overview of the code when working with legacy code. Imagine that you have been brought into a project that has been sitting idle for several months. The collective memory of the team has faded, and many of the details are now relegated to the comments in the code. Even for a simple project, it can take a long time to explore the source and get a good impression of how best to proceed and make the most of the work already invested. Reverse engineering to the rescue – generate a model from the existing code and see it all mapped out.

Great – so now we can get code from a model and a model from code. Any changes made on either side should optimally be reflected in the other – the model keeps the goals and design decisions clear, but the changes in the code need to be reflected in the model to verify that these goals are being upheld, even if the code repository is split between several locations. Who are you going to assign to keep them synchronized? Bob the intern will be disappointed that he's now back to making coffee, because roundtrip engineering has just usurped his task. Team members (even poor Bob)

can work on the code or the model, whichever is more convenient for their immediate task, and then automatically synchronize the two to reduce maintenance overhead and errors.

3.3 Documentation Generation

Rare is the case where systems are commissioned, planned, implemented, and maintained by a single person with a photographic memory. Documentation is essential during all stages of development, from requirements gathering through delivery, for communication, clarity, and sanity. The people involved in throughout the product lifecycle need different information from the single model, and while they could be presented with a single behemoth of a document to wade through, it would be much simpler and elegant to generate just what they need.

Gentleware implemented an ingenious solution where modules, called 'books', can be defined, rearranged, and stored as templates for re-use for each specific target group. Each book defines not only the content, but also the style of the content – such as whether or not to include private attributes or display stereotypes. The level of customization available is unparalleled.

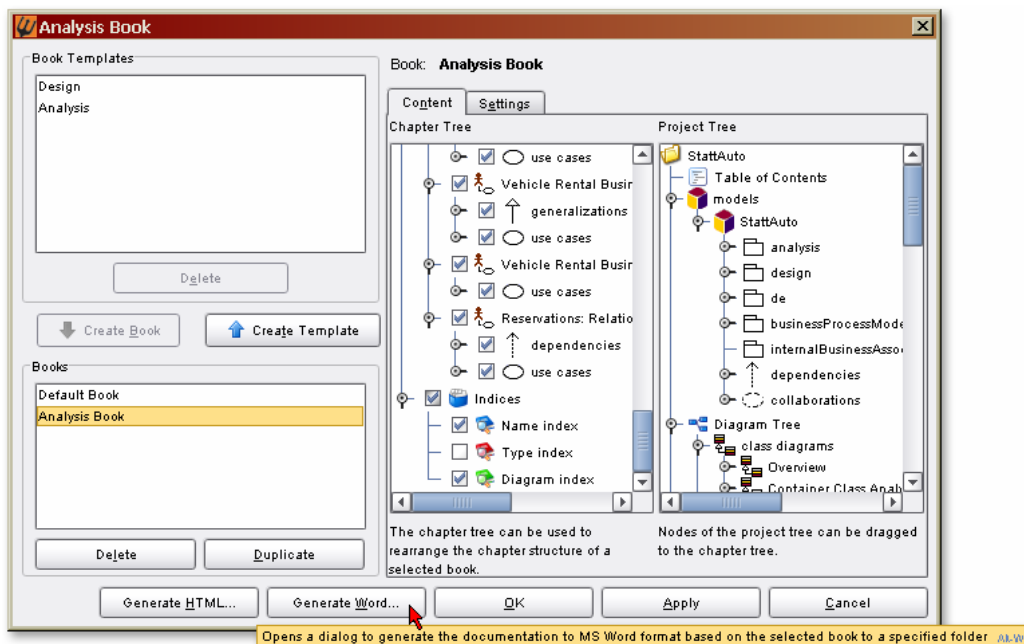


Figure 5 – Documentation generation dialog.

Quick meeting with the client to clarify some processes? Create a word document with just the use case and Activity Diagrams with an index of the diagrams. Architects continuing the meeting after the client leaves to discuss implementation? Add the relevant Class Diagrams and an index of element names to their doc. Programmers ready to get to work? Generate the whole set as UMLdoc, a familiar HTML format based on the Javadoc they know and love. UMLdoc even recognizes and utilizes Javadoc tags inserted into the code via Poseidon or an external IDE.

4 UML 2.0 Support

UML 2.0 offers more detailed and more informative diagrams that more accurately describe your system. While UML 1.x was quite an exciting offering, it lacked more sophisticated notations to denote things like loops and switch statements in Sequence Diagrams. UML 2.0 notation is backwards compatible with UML 1.x – existing 1.x diagrams still make sense within a UML 2.0 context, and can take on all, some, or none of the new features as required. It is completely at the discretion of the user as to which elements to use.

Gentleware focuses on the user experience while modeling; in order to completely assess the best implementation of the UML 2.0 changes, our approach is to focus on optimally updating one diagram type at a time, rather than to create a 'close enough' function that is repeated through all diagrams. In addition to user interface and rendering changes, the updated diagram includes adherence to the UML 2.0 metamodel, rather than the choice of other vendors to create a representation that appears as UML 2.0 but leaves the underlying logic alone.

4.1 State Machine Diagrams

State Machine Diagrams (also referred to as State Diagrams or State Charts) are used to illustrate the possible states of an object and the causes and effects of those state changes within the object. Business process models do not lend themselves to implementation in an object-oriented way; with State Machine Diagrams the business process can be broken down and expressed in terms of states for each object involved in the process. Numerous other systems, real-time systems for example, also have a great need to plan the states of objects.

Once upon a time, developers were afraid (and somewhat rightly so) that generated code was bloated, sub-optimal, and unwieldy. That time has long since passed, and it is time to lay those prejudices to rest. New technology and innovations have made it possible to apply UML and object-oriented development techniques with very low overhead; even small systems with 8 and 16 bit controllers can be implemented efficiently. The ANSI C and C++ code generators in Poseidon have been uniquely created to fit the demanding criteria of embedded systems, such as memory resource and performance issues.

4.1.1 One Element, Multiple State Machine Diagrams

In UML 2.0, a BehavioredClassifier such as a Class, UseCase, Collaboration, Component or Node may have 0..n state machines. The level of clarity this provides is an enormous enhancement; for example, a single use case may now be broken out into several State Machine Diagrams, each with a single purpose. A use case describing an ATM transaction may benefit from one diagram depicting the state of an ATM machine and a second diagram depicting the state of an account, for instance.

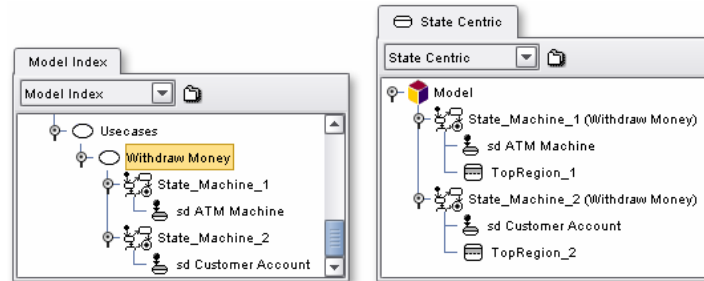


Figure 6 - A single use case 'Withdraw Money' and two associated State Machine Diagrams 'sd ATM Machine' and 'sd Customer Account' as seen in the Navigation Pane.

Likewise, each state machine itself may have 0..n State Machine Diagrams that can depict all or part of the state machine. Select only the relevant details to include in a diagram, reducing clutter in the diagrams to increase comprehension.

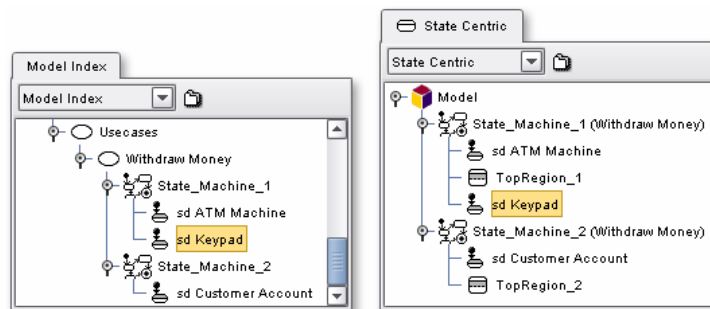


Figure 7 – A single state machine with two State Machine Diagrams.

4.1.2 Orthogonal and Submachine States

The states within a State Machine Diagram have also undergone enhancement. Most notably, orthogonal and submachine states have been added.

According to the UML 2.0 specification, orthogonal states contain two or more concurrent regions within a single state. Poseidon is unique in that it does not restrict orthogonal states to merely two regions, and that regions can be added or deleted from a state 'on the fly' using rapid buttons. Poseidon then automatically updates the element type if required; for instance, an orthogonal state with only one region will automatically become a composite state, in accordance with the specification.

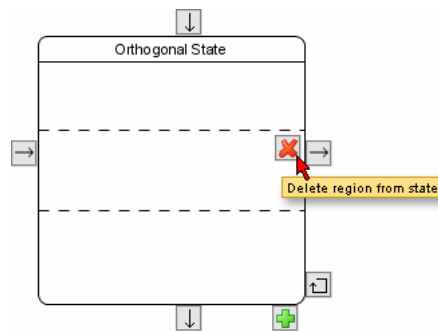


Figure 8 – An orthogonal state with three regions and its rapid buttons activated.

A submachine is used to hide or extract details of a state machine or to reuse the functionality of a state machine. Think of it like a subroutine – a state can in effect ‘call’ a state machine by entering the submachine state. The ‘return’ occurs when the submachine fulfills the ending conditions and returns from a submachine state.

References are used to indicate the entry and exit points of the submachine, which can optionally be displayed in a State Machine Diagram. Deleting the reference does not delete the entry and exit points of the submachine, it only removes their display in the upper-level diagram.

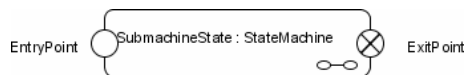


Figure 9 – Submachine state displaying entry and exit points.

4.2 Sequence Diagrams

A Sequence Diagram is an easily comprehensible visualization of single scenarios or examples of business processes with regard to their behavior in time. It focuses on *when* the individual objects interact with each other during execution. It is particularly useful for modeling usage scenarios such as the logic of methods and the logic of services.

The diagram essentially includes a timeline that flows from the top to the bottom of the diagram and is displayed as a dotted line. The interaction between objects is described by specifying the different kinds of messages sent between them.

4.2.1 Combined Fragments

New in UML 2 is the frame surrounding any diagram with a label in the upper left corner. These frames are more importantly used for interaction occurrences and combined fragments within the diagram itself, such as with a loop or switch statement (called ‘alt’ in UML). A fragment is simply a piece of a Sequence Diagram; these fragments can be combined to form logical units surrounded by a frame, each with one or more interaction operands and an interaction operator.

One of the simplest operators is the loop operator. The loop will execute one fragment a specified number of times, and the conditional for the loop can either be a number or a Boolean expression.

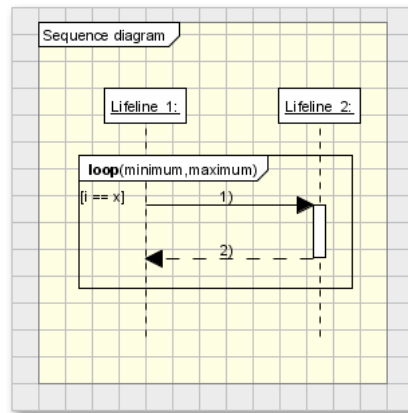


Figure 10 – Sequence Diagram depicting a ‘loop’ combined fragment.

Poseidon for UML makes it simple to implement these new features. Frames are created by first selecting the desired operator in the toolbar, then dragging a box around the lifelines and messages to be included within the frame. As with other elements, inline editing is possible for the most often used properties, and the navigable Properties tabs always contain the full set of these properties.

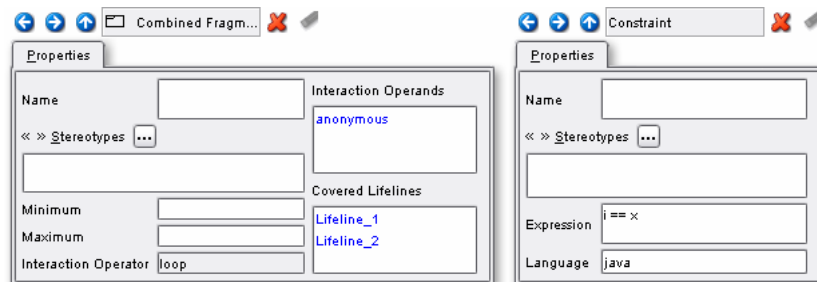


Figure 11 – Properties tabs for combined fragments and constraints.

4.2.2 Inspections

Sequence Diagrams include modeling tips with the addition of inspections. Whenever Poseidon has a suggestion for your diagram, a red exclamation point will appear near the questionable item. Mouse over the exclamation point to reveal the critique.

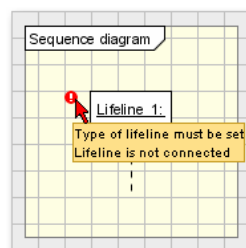


Figure 12 – Sequence Diagram with activated inspection.

5 Features and Benefits

Poseidon for UML is better than ever, with important new features to make your models easier to produce and to share. But you knew that, right? Humor us, and let us reiterate for you exactly what Poseidon will do for you:

Increase Communication

- Unite multiple disciplines into a single team
- Solidify teamwork with real time collaboration tools
- Share work with a wide audience through custom documents

Easy Adoption

- Lower resistance to new tools with the easy and intuitive interface
- Energize development efforts with a rapid ramp-up period

Speed Up Development

- Accelerate the development lifecycle with code generation
- Safeguard requirements through the entire design process
- Eliminate tedious tasks like coding accessors
- Render more readable and maintainable code with freestyle comments
- Boost design excellence with complete and accurate UML models
- Amplify quality while fostering a shorter time-to-market
- Optimize resources in embedded systems
- Augment your Eclipse experience with the Poseidon integration
- Visualize your code through reverse engineering for improved clarity
- Increase comprehension by partitioning problems across diagrams

Watch the Bottom Line

- Constrain costs and development hours
- Expand a small deployment with ease
- Deliver projects on time and on budget
- Sustain focus of software development projects
- Assure long-term coherence to project goals

5.1 Business Analysts

As a liaison between the business and technical sides of a team, it's important to be 'multilingual' in business-speak and tech-talk. UML is the common dialect – software requirements can be abstracted into a format that people without programming knowledge can understand, and business processes can be explicitly presented to the folks who will implement them.

Main Concerns of the Business Analyst:

- Gather requirements
- Communicate needs
- Model high level business processes
- Model business activities
- Model work flow
- Display system behavior

Activity Diagrams - Business Analysts will be mainly interested in using Activity Diagrams in order to model the behavior of a system and the way in which these behaviors relate to the general system flow. Activity Diagrams show a high level view of the general processes rather than specific internal behavior.

Use Case Diagrams – Analysts will also find Use Case Diagrams to be advantageous while mapping the functional requirements and behavior of a system.

Documentation Tab – Diagrams alone can be insufficient to capture the meaning behind use cases. The documentation tab contains both a WYSIWYG editor and a plain-text editor that accepts HTML tags to enable full text descriptions of a diagram or model element. This information can then be included within a generated set of documents.

Requirements Engineering Plug-In – Add additional requirements to standard Use Case Diagrams easily with the Requirements Engineering plug-in. The requirements description is stored within your model; there is no need to manually synchronize a separate requirements document with your UML model.

5.2 Programmers

Programmers today are required to develop higher quality code in a shorter time and with fewer resources. The right tools are essential for meeting these goals – tools that are powerful and easy to use, like Poseidon for UML.

Main Concerns of the Programmer:

- Code generation
- Documentation generation
- Integration into Eclipse

Forward Engineering – Forward engineering refers to the process by which Class Diagrams are used as the blueprint to generate source code in any of the supported target languages. This allows the developer to edit a model and have these changes implemented in the source code.

Reverse Engineering – Reverse engineering allows developers to view Class Diagrams that are automatically generated from existing Java source. The developer is then able to use the resulting Class Diagrams to demonstrate the static design view of the system. Class Diagrams are composed principally of classes, interfaces, and the relationships between them, which can have corresponding counterparts in the resulting implementation.

Roundtrip Engineering – The ability to roundtrip engineer software from existing source code to UML diagrams and back again gives programmers peerless flexibility. Roundtrip engineering refers to the process in which both forward and reverse engineering is used to keep the model and code synchronized.

UML 2.0 State Machine Diagrams – Programmers can use State Machine Diagrams to map out the system state. State Machine Diagrams illustrate how elements transition between states and capture system changes over time.

Class Diagrams as Package Diagrams – It is extremely useful to be able to visualize the package structure of the system. Package Diagrams describe the system architecture and are used to organize the diagrams and elements as well as their dependencies.

Activity Diagrams – Activity Diagrams are used to illustrate the dynamic nature of the system and discern the flow of control between activities, resulting in increased understanding of the changes in the state of the system.

Eclipse Integration – Running Poseidon for UML within the Eclipse IDE combines the familiar interface and UML features of Poseidon with the power of Eclipse. Changes made in the source code within Eclipse can update your model, and your model can generate code directly into your Eclipse project.

5.3 Project Managers

Project management involves a lot of risk management – risks of not meeting targets, budgets, quality marks, etc. Poseidon for UML is the easiest tool to learn and to use, letting your team realize the many benefits of UML as quickly, easily, and inexpensively as possible.

Main concerns of the Project Manager:

- Assure consistent and correct use of UML throughout the team
- Utilize a tool with a short learning curve
- Assure high quality output with minimal resource investment

Use Case Diagrams – User stories are easily and accurately captured by Use Case Diagrams, which can then be used to create and communicate acceptance tests.

Focus – Let UML do what it does best: capture a system. Project management “features” in other UML tools are inadequate, so a separate project management system is inevitably required. Poseidon avoids this muddling with the single goal of being a simple and elegant modeling tool.

Enforced Semantic Rules – Lots of tools on the market are glorified drawing tools that draw pretty pictures without underlying logic. Poseidon builds models that are based on the UML metamodel and therefore enforce inherent rules, resulting in well-formed models.

5.4 Software Architects

Software architects require a deep knowledge of planned and existing software architectures, as well as the interfaces between them. Poseidon for UML gives you the tools you need to represent your system, including security, communication protocols, simultaneous processing, data structures, etc.

Main concerns of the Software Architect:

- Analysis and high-level design
- Abstract and invent elegantly simple models of complex systems
- Ensure compliance with or enhancement of the existing architecture or design
- Studying the problem and the aim of the software to be developed as per the specification

Use Case Diagrams – Use Case Diagrams can be used to map the functional requirements of the system and expand upon the high-level business processes as defined by the Business Analyst. Each use case describes a single discrete segment of the proposed functionality of the system.

Once a Use Case Diagram has been created, it can be used as a map to develop detailed classes, keeping in mind the objectives delineated by the use cases.

Deployment Diagrams – Deployment Diagrams provide a static overview of the runtime configuration of processing nodes and their components, as well as the connections between hardware, software and middleware in a system.

Component Diagrams – Physical aspects of a system, such as executables, libraries, and data files, can be modeled using the Component Diagram.

Sequence Diagrams – Sequence Diagrams illustrate the interactions between classes by detailing the messages that are exchanged between objects and the lifetimes of those objects.

Class Diagrams – Class Diagrams can also be used to create an overview of the package structure of the software, to organize diagrams and elements into manageable groups, and to declare dependencies between these groups.