ArgoUML User Manual

An introduction to ArgoUML

by Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, and Jeremy Bennett

ArgoUML User Manual: An introduction to ArgoUML

by Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, and Jeremy Bennett

Table of Contents

List of Figures

Preface

Software design is a cognitively challenging task. Designers must manually enter designs, but the primary difficulty is decision-making rather than data-entry. If designers improved their decision-making capabilities, it would result in better designs.

Current CASE tools provide automation and graphical user interfaces that reduce the manual work of entering a design and transforming a design into code. They aid designers in decision-making mainly by providing visualization of design diagrams and simple syntactic checks. Also many CASE tools provide substantial benefits in the area of version control and concurrent design mechanisms. One area of design support that has been not been well supported is analysis of design decisions.

Current CASE tools are usable in that they provide a GUI that allows designers to access all the features provided by the tool. And they support the design process in that they allow the designer to enter diagrams in the style of popular design methodologies. But they typically do not provide process support to guide the designer through the design task. Instead, designers typically start with a blank page and must remember to cover every aspect of the design.

ArgoUML is a domain-oriented design environment that provides cognitive support of object-oriented design. ArgoUML provides some of the same automation features of a commercial CASE tool, but it focuses on features that support the cognitive needs of designers. These cognitive needs are described by three cognitive theories:

1. reflection-in-action;

2. opportunistic design; and

3. comprehension and problem solving.

ArgoUML is based directly on the UML 1.3 specification. In fact, a large part of ArgoUML was generated automatically from the UML specification. ArgoUML is (to the best of our knowledge) the only tool that implements the UML meta-model exactly as specified. In contrast, current commercial tools use tools use basically the same internal representation of the design that they used in previous versions.

Furthermore, it is our goal to provide comprehensive support for OCL (the Object Constraint Language) and XMI (the XML Model Interchange format), which other tools currently do not support.

ArgoUML was originally developed by a small group of people as a research project. ArgoUML has many features that make it special, but it does not implement all the features that commercial CASE tools provide.

The current developer release of ArgoUML implements all the diagram types of the UML 1.3 standard [http:/-/-www.omg.org/-technology/-documents/-formal/-uml.htm]. It is written in Java and runs on every platform with a Java 1.2 (or newer) port being available. It uses the open file formats XMI [http:/-/-www.omg.org/-technology/-documents/-formal/-xmi.htm] (XML Metadata Interchange format) (for model information) and PGML [http:/-/-www.w3.org/-TR/-1998/-NOTE-PGML] (Portable Graphics Markup Language) (for graph information) for storage. A new file format with graphs embedded as SVG [http:/-/-www.w3.org/-TR/-2000/-CR-SVG-20001102/-index.html] (Scalable Vector Graphics) in the XMI is in discussion, but is postponed until a standard for such a file format is defined by the OMG [http:/-/-www.omg.org] (Object Management Group).

This manual is the cumulative work of several people. In addition to new material by the current author, Jeremy Bennett, it includes material from earlier versions by Alejandro Ramirez, Philippe Vanpeperstraete and Andreas Rueckert, from the developers cookbook by Markus Klink and Linus Tolke, the Quick Guide by Kunle Odutola and the FAQ by Dennis Daniels. And of course the review comments of users and developers too many to name.

ArgoUML is available for free and can be used in commercial settings. For terms of use, see the license agreement presented when you download ArgoUML. We are providing the source code for ArgoUML for you to review, customize to your needs, and improve. Over time, we hope that ArgoUML will evolve into a powerful and useful tool for everyone to use.

This User Manual is aimed at the working designer, who wishes to make use of ArgoUML. In its current incar-

nation, it assumes familiarity with UML, but eventually it will support those new to UML.

The ArgoUML project welcomes those who want to get more involved. Look at the project website [http:/-/-argouml.tigris.org/-] to find out more.

Finally tell us what you think about this User Manual. Your comments will help us improve things. Email your thoughts to the ArgoUML Users Mailing List [mailto:users@argouml.tigris.org].

Chapter 1. Introduction

Table of Contents

1.1. Origins and Overview of ArgoUML

1.1.1. Object Oriented Analysis and Design

Over the past decade, Object Oriented Analysis and Design (OOA&D) has become *thethe* dominant software development paradigm. With it has come a major shift in the thought processes of all involved in the software development life cycle.

Programming language support for objects began with Simula 67, but it was the emergence in the 1980's of hybrid languages, such as C++, Ada and Object Pascal that allowed OOA&D to take off. These languages provided support for both OO and procedural programming. Object Oriented *programmingprogramming* became mainstream.

An OO system is designed and implemented as a *simulationsimulation* of the real world using software artifacts. This premise is as powerful as it is simple. By using an OO approach to *designdesign* a system can be designed and tested (or more correctly simulated) without having to actually build the system first.

It is the development during the 1990's of tools to support Object Oriented *analysisanalysis* and *designdesign* that moved this approach into the mainstream. When coupled with the ability to design systems at a very high level, a tool based OOA&D approach has enabled the implementation of more complex systems than previously possible.

The final driver that has propelled OOA&D has been its suitability for modeling graphical user interfaces. The popularity of object based and object oriented graphical languages such as Visual Basic and Java reflect the effectiveness of this approach.

1.1.2. The Development of ArgoUML

During the 1980's a number of OOA&D process methodologies and notations were developed by different research teams. It became clear that were many common themes, and during the 1990's a unified approach for OOA&D notation was developed under the auspices of the Object Management Group [http:/-/-www.omg.org]. This standard became known as the Unified Modeling Language (UML), and is now the standard language for communicating OO concepts.

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 modeling software systems. ArgoUML has a number of very important distinctions from many of these tools.

1. ArgoUML draws on research in cognitive psychology to provide novel features that increase productivity by supporting the cognitive needs object-oriented software designers and architects.

2. ArgoUML supports open standards extensively—UML, XMI, SVG, OCL and others. In this respect, ArgoUML is still ahead of many commercial tools.

3. 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.

4. ArgoUML is an open source project. 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 tool technologies.

In creating the ArgoUML development tool and environment, Jason Robbins and the rest of his research team at the University of California, Irvine leveraged the benefits of UML as the most prevalent OO modeling language and Java as one of the most productive OO development platforms to produce a solid tool for OO systems design and, further a test bed for the evolution of object oriented CASE tools development and research.

A first release of ArgoUML was available in 1998 and more than 100,000 downloads since then show the impact that this project has made, being popular in educational and commercial fields.

1.1.3. Finding Out More About the Project Today

1.1.3.1. The ArgoUML Development Process and the Infrastructure it Uses

Although Jason Elliot Robbins (he's still actively involved, being the maintainer of the Tigris host at CollabNet, where the project is hosted now) has stepped back from project leadership and handed the stab to Tobias Baier, the project is still going strong. There are more than 300 members on the developer mailing list (see http://argouml.tigris.org/servlets/ProjectMailingListList
[http://-/-argouml.tigris.org/-servlets/-ProjectMailingListList]), with 15-20 forming the core group.

This is the place where all the discussion on the latest tasks takes place, and developers discuss the directions the project should take. Although controversial at times, these discussions are always kept nice and friendly (no flame-wars and such), so newbies should not hesitate and participate in them. They'll always get a warm welcome there.

1.1.3.2. More on Infrastructure

Besides the developer mailing list, there's also a mailing for users (see http://argouml.tigris.org/servlets/ProjectMailingListList
[http://-/-argouml.tigris.org/-servlets/-ProjectMailingListList]), where they can discuss problems from a user perspective. Some developers also read this list, so there's a good chance to get qualified help.

Before posting to this list, you should take a look at the user FAQ [http://-/-argouml.tigris.org/-documentation/-defaulthtml/-faq] maintained by Dennis Daniels.

More information on ArgoUML and other UML related topics is also available on the ArgoUML website [http://-/-www.argouml.org], maintained by Markus Klink.

1.1.3.3. Used Development Tools

Note

As a user you don't need to know this, but it's here for interest to explain a little about how ArgoUML is built.

ArgoUML uses Ant [http://-/-jakarta.apache.org/-ant/-index.html] as it's main build tool. The whole build process is controlled by the `build.xml` file in the main source directory.

The parser for Java reverse engineering is generated with Antlr [http:/-/-www.antlr.org] from a customized Java grammar.

Instead of using a self-implemented UML meta-model, ArgoUML for all versions after 0.8 uses a meta-model implementation [http:/-/-nsuml.sourceforge.net] provided by NovoSoft.

OCL is handled by Frank Finger's OCL compiler [http:/-/-dresden-ocl.sourceforge.net], developed at the TU Dresden.

1.2. Scope of This User Manual

1.2.1. Target Audience

The current version of this document is aimed at experienced users of UML in OOA&D (perhaps with other tools) who wish to transfer to ArgoUML.

Future versions will support designers who know OOA&D, and wish to adopt a UML based process.

A long term goal is to support those who are learning design and wish to start with a UML based OOA&D process and people interested in modularized code design with a GUI.

1.2.2. Scope

The intention is that this document will provide a comprehensive guide, enabling designers to use ArgoUML to its full extent. It is in two parts.

- A tutorial manual, showing how to work with ArgoUML

- A complete reference manual, recording everything you can do with ArgoUML.

The current version of the document has achieved the second of these.

In this guide there are some things you will not find

- Descriptions of how ArgoUML works on the inside.

- How to improve ArgoUML with new features and functions.

- A trouble shooting guide.

- A summary quick reference to using ArgoUML.

These are covered elsewhere, particularly in the FAQ [http:/-/-argouml.tigris.org/-documentation/-defaulthtml/-faq/-], Quick Guide [http:/-/-argouml.tigris.org/-documentation/-defaulthtml/-quickguide/-] and Developers Cookbook [http:/-/-argouml.tigris.org/-documentation/-defaulthtml/-cookbook/-].

1.3. Overview of the User Manual

1.3.1. Tutorial Manual Structure

Chapter 2 provides an overview of UML based OOA&D, including a guide to getting ArgoUML up and running.

Chapter 3 through Chapter 6 then step through each part of the design process from initial requirements capture through to final project build and deployment.

As each UML concept is encountered, its use is explained. Its use within ArgoUML is then described. Finally the case study is used to give examples of the concept in use.

1.3.2. Reference Manual Structure

Chapter 7 is an overview of the user interface. Separate chapters Chapter 8 through Chapter 12 describe each of the sub-windows of the user interface, known as *PanesPanes*, and the menu bar.

Chapter 13 gives details of all the cognitive critics within the system. Eventually ArgoUML will link directly to this manual when giving advice on critics.

Chapter 14 is an overview of the artifacts with ArgoUML. The following chapters (Chapter 15 through Chapter 22) describe, the artifacts that can be created through each ArgoUML diagram, and their properties, as well as some standard artifacts provided with the system.

A complete Glossary is provided. Appendix A provides material to supplement the case study used throughout the document. Appendix B and Appendix C identify background information on UML and UML CASE tools. Appendix D describes the plan for creation of this manual. Appendix E gives some guidelines for those writing this manual. Appendix F is a copy of the GNU Free Documentation License.

A future ambition is to provide a comprehensive index

1.3.3. User Feedback

Finally tell us what you think about this User Manual. Your comments will help us improve things. Email your thoughts to the ArgoUML Users Mailing List [mailto:users@argouml.tigris.org].

1.4. Assumptions

The current version of the manual assumes the reader is very familiar with UML already. This is reflected in the sparseness of the description of UML concepts in the tutorial.

The case study is described, but not yet fully realized throughout the tutorial. This will be achieved in future versions of the manual.

Part I. ArgoUML Tutorial Manual

Chapter 2. UML Based OOA&D

Table of Contents

2.1. Introduction

In this chapter, we look at how UML as a notation is used within OOA&D.

2.2. Background to UML

Object orientationObject orientation as a concept has been around since the 1960's, and as a design concept since 1972. However it was in the 1980's that it started to develop as a credible alternative to a *functional approachfunctional approach* in analysis and design. We can identify a number of drivers.

1. The emergence of mainstream OO programming languages like SmallTalk and particularly C++. C++ was a pragmatic OO language derived from C, widely used because of its association with Unix.

2. The development of powerful workstations, and with them the emergence into the mainstream of windowing operating user environments. Graphical User Interfaces (GUI) have an inherent object structure.

3. A number of very public major project failures, suggesting that current approaches were not satisfactory.

A number of researchers proposed OOA&D processes, and with them notations. Those that achieved some success include Coad-Yourdon, Booch, Rumbaugh OMT, OOSE/Jacobson, Shlaer-Mellor, ROOM (for real-time design) and the hybrid Jackson Structured Development.

During the early 1990's it became clear that these approaches had many good ideas, often very similar. A major stumbling block was the diversity of notation, meaning engineers tended to be familiar with one OOA&D methodology, rather than the approach in general.

UML was conceived as a common notation, that would be in the interests of all involved. The original standard was driven by Rational Software (www.rational.com [http://-/-www.rational.com], in which three of the key researchers in the field (Booch, Jacobson and Rumbaugh were involved). They produced documents describing v0.9 and v0.91 during 1996. The effort was taken industry wide through the Object Management Group (OMG), already well known for the CORBA standard. A first proposal, 1.0 was published in early 1997, with an improved version 1.1 approved that autumn.

ArgoUML is based on UML v1.3, which was adopted by OMG in November 1999. UML v1.4 has recently been approved (2001). A major revision, UML v2.0 is planned to be completed in 2002.

2.3. UML Based Processes for OOA&D

It is important to understand that UML is a notation for OOA&D. It does not prescribe any particular process.

Whatever process is adopted, it must take the system being constructed through a number of phases.

1. Requirements Capture. This is where we identify the requirements for the system, using the language of the *problem domainproblem domain*. In other words we describe the problem in the "customer's" terms.

2. Analysis. We take the requirements and start to recast them in the language of a putative solution—the *solution domainsolution domain*. At this stage, although thinking in terms of a solution, we ensure we keep things at a high level, away from concrete details of a specific solution—what is known as *ababstractionstraction*.

3. Design. We take the specification from the Analysis phase and construct the solution in full detail. We are moving from *abstractionabstraction* of the problem to its *realizationrealization* in concrete terms.

4. Build Phase. We take the actual design and write it in a real programming language. This includes not just the programming, but the testing that the program meets the requirements (*verificationverification*), testing that the program actually solves the customer's problem (*validationvalidation*) and writing all user documentation.

2.3.1. Types of Process

In this section we look at the two main types of process in use for software engineering. There are others, but they are less widely used.

In recent years there has also been a move to reduce the effort required in developing software. This has led to the development of a number of lightweight variants of processes (often known as *agile computingagile computing* or *extreme programmingextreme programming*) that are suited to very small teams of engineers.

2.3.1.1. The Waterfall Process

In this process, each stage of the process—requirements, analysis, design and build (code and test) is completed before the next one starts. This is illustrated in Figure 2.1.

Figure 2.1. The waterfall process
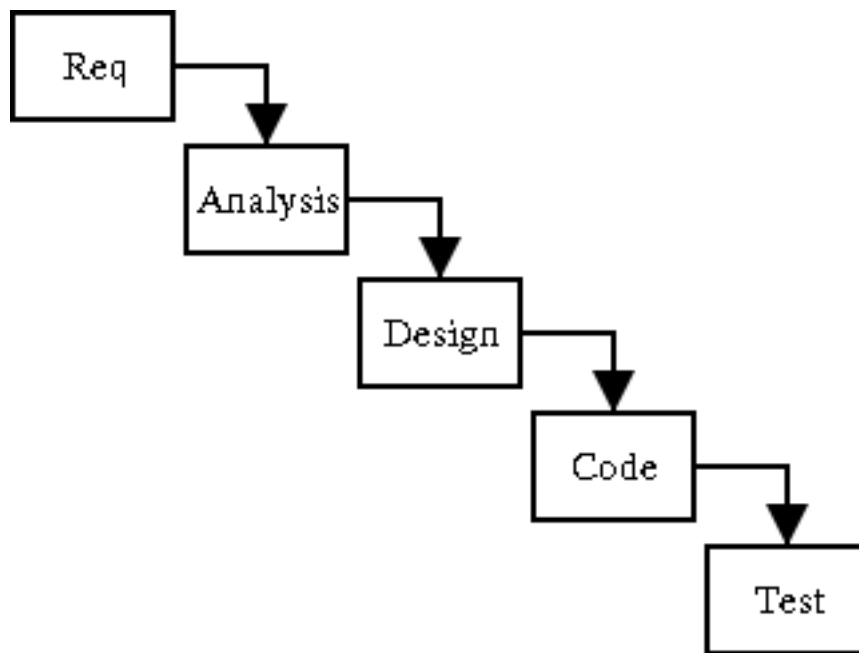
This is a very satisfactory process where requirements are well designed and not expected to change, for example automating a well proven manual system.

The weaknesses of this approach show with less well defined problems. Invariably some of the uncertainties in the requirements will not be clarified until well into the analysis and design, or even code phases, requiring backtracking to redo work.

The worst aspect of this, is that working code does not become available until near the end of the project, and very often it is only at this stage that problems with the original requirements (for example with the user interface) become apparent.

This is exacerbated, by each successive stage requiring more effort, than the previous, so that the costs of late problem discovery are hugely expensive. This is illustrated by the pyramid in Figure 2.2.

Figure 2.2. Effort involved in the steps of the waterfall process
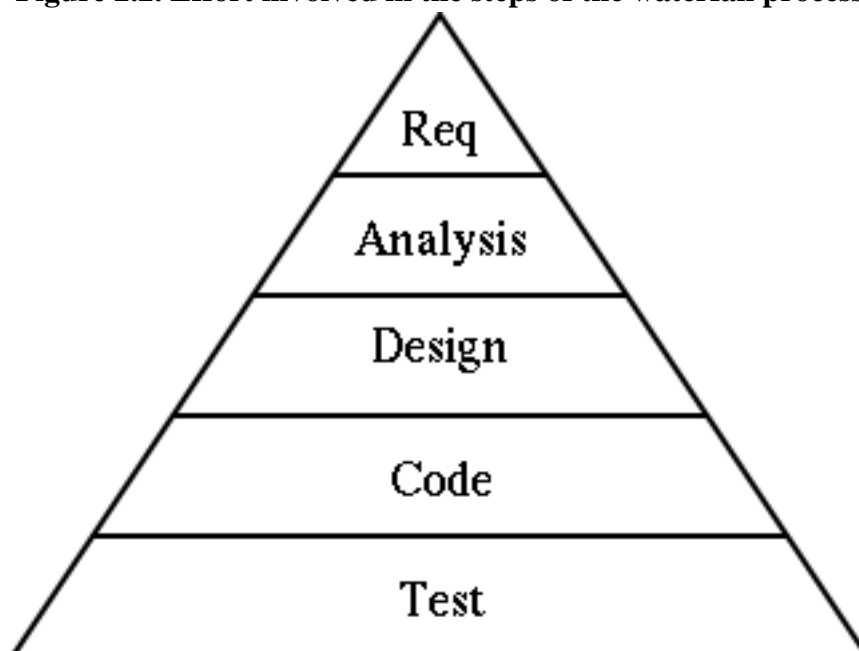
The waterfall process is still probably the dominant design process. However because of its limitations it is increasingly replaced by *iterativeiterative* processes, particularly for projects where the requirements are not well defined.

2.3.1.2. Iterative Development Processes

In recent years a new approach has been used, which aims to get at least part of the code up and running as quickly as possible, to bring discovery of problems forward in the development cycle.

These processes use a series of "mini-waterfalls", defining a few requirements (the most important) first, taking them through analysis, design and build to get an early version of the product, with limited functionality, related to the most important requirements. Feedback from this can then be used to refine the requirements, spot problems etc before more work is done.

The process is then repeated for further requirements to construct a product with a step up in functionality. Again further feedback can be applied to the requirements.

The process is repeated, until eventually all requirements have been implemented and the product is complete. It is this *iterationiteration* that gives these processes their name. Figure 2.3 shows how this process compares to the pyramid structure of the Waterfall Process.

Figure 2.3. Effort involved in the steps of an iterative process
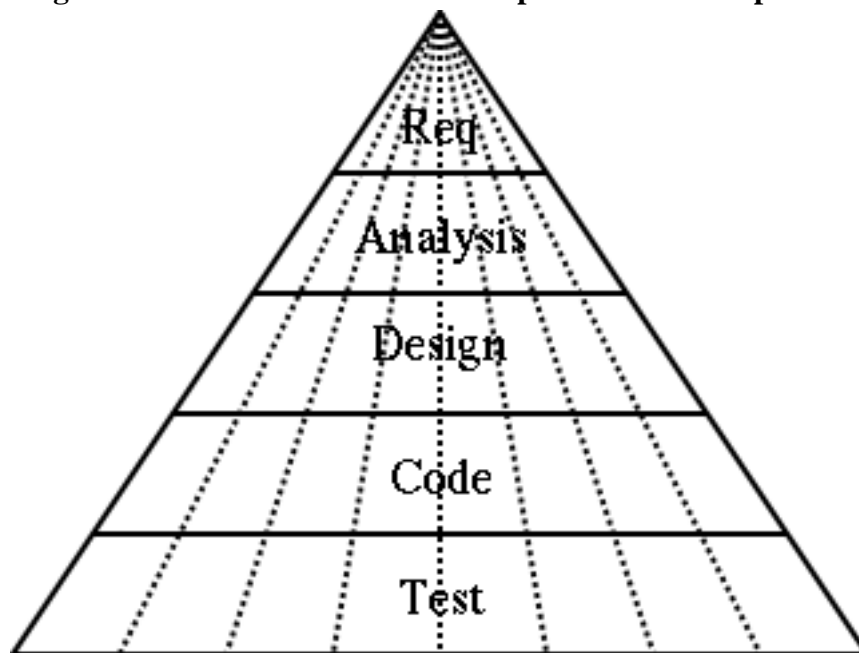
The growth in popularity of iterative processes is closely tied to the growth of OOA&D. It is the clean encapsulation of objects that allows a part of a system to be built with stubs for the remaining code clearly defined.

2.3.1.2.1. The Rational Unified Process

Perhaps the best known Iterative Process is the Rational Unified Process (RUP) from Rational Software (www.rational.com [http:/-/-www.rational.com]).

This process recognizes that our pyramid view of even slices of the waterfall is not realistic. In practice the early iterations tend to be heavy on the requirements end of things (you need to define a reasonable amount even to get started), whilst the later iterations have more of their effort in the design and build areas.

RUP recognizes that iterations can be grouped into a number of *phasesphases* according to their stage in the overall project. Each phase may have one or more iterations.

- In the *inception phaseinception phase* iterations tend to be heavy on the requirements/analysis end, whilst any build activity may be limited to emulation of the design within a CASE tool.

- In the *elaboration phaseelaboration phase* iterations tend to be completing the specification of the requirements, and starting to focus on the analysis and design, and possibly the first real built code.

- In the *construction phaseconstruction phase* iterations are more or less completed with the requirements and analysis, and the effort is mostly in design and build.

- Finally, in the *deployment phasedeployment phase* iterations are largely about build activity, and in particular the testing of the software.

Note

It should be clear that testing is an integral part of all phases. Even in the early phases the requirements and design should be tested, and this is facilitated by a good CASE tool.

We shall use an iterative process in this manual, that is loosely based on the RUP.

2.3.1.2.2. How Big is an Iteration?

A good rule of thumb is that an iteration should take between six and ten weeks for typical commercial projects. Any longer and you have probably bitten off too many requirements to do in one go. You also lose focus on getting the next working iteration completed. Any shorter and you probably haven't got enough requirements to make a significant advance, and will struggle to get all the work done.

This then begs the question of how many iterations in total. This depends on the size of project. Take the estimated time (and working out/guessing that is a whole subject on its own), and divided it into 8 week chunks. Experience seems to suggest that the iterations will divide in the ratio of around 1:2:3:3 into RUP style inception, elaboration, construction and deployment phases. A project that has great vagueness in its specification (some advanced research projects for example) will tend to be heavier on the early phases.

When building a product to contract for a customer the end point is well defined. However when developing a new product for the market place, a strategy that can be used is to decide the product launch date, and hence the end date for completion of engineering (some time before). The time is then divided into iterations, and as much of the product as can be built in that time developed. The iterative process is very effective where time to market is more important than the exact functionality.

2.3.1.3. Recursive Development Processes

Very few software systems are conceived as monolithic artifacts. They are broken down into subsystems, modules etc.

Software processes are the same, with early parts of the process defining a top level structure, and the process reapplying to parts of the structure in turn to define ever greater details.

For example the initial design of a telephone system might identify objects to i) handle the phone lines, ii) process the calls, iii) manage the system and iv) bill the customer. The software process can then be reapplied to each of these four components to identify their design.

OOA&D with its clean boundaries to objects, naturally supports this approach. Such OOA&D with recursive development is sometimes abbreviated as OOA&D/RD.

Recursive development can be applied equally well to waterfall or iterative processes. It is not an alternative to them.

2.3.2. A Development Process for This Manual

For the purpose of this manual we will use a stripped down iterative process with recursive development, loosely akin to RUP. The case study will take us through the first iteration, although at the end of the tutorial

section of the manual we will look at how the project will develop to completion.

Within that first iteration, we will tackle each of the requirements capture, analysis, design and build activities in turn. Not all parts of the process are based on UML or ArgoUML. We will look at what other material is needed outside.

2.3.2.1. Requirements Capture

Our requirements capture will use the UML concept of *Use CasesUse Cases*. Starting with a *Vision DocumentVision Document* we will see how Use Cases can be developed to describe all aspects of the system's behavior in the problem domain.

2.3.2.2. Analysis

During the analysis stage, we will introduce the UML concept of *classesclasses* to allow us to build a top level view of the objects that will make up the solution—sometimes known as a *concept diagramconcept diagram*.

We will introduce the UML *sequence diagramsequence diagram* and *state diagramstate diagram* to capture requirements for the overall behavior of the system.

Finally we will take the Use Cases from the requirements capture stage, and recast them in the language of the solution domain. This will illustrate the UML ideas of *stereotypingstereotyping* and *realizationrealization*.

2.3.2.3. Design

We use the UML *package diagrampackage diagram* to organize the components of the project. We then revisit the class diagram, sequence diagram and state diagram, to show how they can be used recursively to design the complete solution.

During this part of the process, we need to develop our system architecture, to define how all the components will fit together and operate.

Although not strictly part of our process, we'll look at how the UML *collaboration diagramcollaboration diagram* can be used as an alternative to, or to complement the *sequence diagramsequence diagram*. Similarly we will look at the UML *activity diagramactivity diagram* as an alternative or complement to the state diagram.

Finally we shall use the UML *deployment diagramdeployment diagram* to specify how the system will actually be realized.

2.3.2.4. Build

UML is not really concerned with code writing. However at this stage we will show how ArgoUML can be used for code generation.

We will also look at how the UML Use Case Diagram and Use Case Specification are invaluable tools for a test program.

2.4. Why ArgoUML is Different

In the introduction, we listed the four key things that make ArgoUML different: i) it makes use of ideas from cognitive psychology, ii) it is based on open standards; iii) it is 100% pure Java; and iv) it is an open source project.

2.4.1. Cognitive Psychology

2.4.1.1. Theory

ArgoUML is particularly inspired by three theories within cognitive psychology: i) reflection-in-action, ii) opportunistic design ii) and comprehension and problem solving.

2.4.1.1.1. Reflection-in-Action

This theory observes that designers of complex systems do not conceive a design fully-formed. Instead, they must construct a partial design, evaluate, reflect on, and revise it, until they are ready to extend it further.

As developers work hands-on with the design, their mental model of the problem situation improves, hence improving their design.

2.4.1.1.2. Opportunistic Design

A theory within cognitive psychology suggesting that although designers plan and describe their work in an ordered, hierarchical fashion, in reality, they choose successive tasks based on the criteria of cognitive cost.

Simply stated, designers do not follow even their own plans in order, but choose steps that are mentally least expensive among alternatives.

2.4.1.1.3. Comprehension and Problem Solving

A design visualization theory within cognitive psychology. The theory notes that designers must bridge a gap between their mental model of the problem or situation and the formal model of a solution or system.

This theory suggests that programmers will benefit from:

1. Multiple representations such as program syntactic decomposition, state transitions, control flow, and data flow. These allow the programmer to better identify elements and relationships in the problem and solution and thus more readily create a mapping between their situation models and working system models.

2. Familiar aspects of a situation model, which improve designers' abilities to formulate solutions.

2.4.1.2. Practical Application in ArgoUML

The implementation of these theories within ArgoUML is through a number of techniques.

1. The design of a user interface which allows the user to view the design from a number of different perspectives, and allows the user to achieve goals through a number of alternative routes.

2. The the use of processes running in parallel with the design tool, evaluating the current design against models of how "best practice" design might work. These processes are known as *design criticsdesign critics*.

3. The use of *to-do liststo-do lists* to convey suggestions from the design critics to the user, as well as allowing the user the record areas for future action.

4. The use of checklists, to guide the user through a complex process.

2.4.2. Open Standards

UML is itself an open standard. ArgoUML throughout has tried to use open standards for all its interfaces.

The key advantage of open standards is that it permits easy inter-working between applications, and the ability to move from one application to another as necessary.

It is this very flexibility that means some commercial software is not built to use standards. In a monopolistic strategy, non-standard interfaces lock your customer to you.

This is a two-edged sword. Customers are not stupid, and are reluctant to buy non-standard software, because of the danger of lock-in.

The open source movement, is inherently about avoiding monopolistic control of software. Open standards are a natural direction for such software—including ArgoUML.

2.4.2.1. XML Metadata Interchange (XMI)

XML Metadata Interchange (XMI)XML Metadata Interchange (XMI) is the standard for saving the meta-data that make up a particular UML model. In principle this will allow you to take the model you have created in ArgoUML and import it into another tool.

This clearly has advantages in allowing UML to meet its goal of being a standard for communication between designers.

The reality is not quite this good. XMI is a recent standard, and ArgoUML is one of the few tools to implement it. Furthermore it says nothing about the graphical representation of the models, so diagram layout is lost. ArgoUML gets round this by saving graphical information with the model (see Section 2.5.3.1).

2.4.2.2. Standard Vector Graphics (SVG)

Standard Vector Graphics (SVG)Standard Vector Graphics (SVG) is a recent standard proposed to the W3C Standardization body (www.w3c.org [http:/-/-www.w3c.org]) by Adobe (www.adobe.com [http:/-/-www.adobe.com]). As yet it is not widely adopted, but is likely to become the standard for graphical data that is based on vectors (i.e. lines) rather than bitmaps.

For now ArgoUML saves diagrams using an earlier proposed standard, *Portable Graphics Markup Language (PGML)Portable Graphics Markup Language (PGML)*. However it has the option to export graphical data as SVG for those who can make use of it.

2.4.2.3. Object Constraint Language (OCL)

Object Constraint Language (OCL)Object Constraint Language (OCL) is the UML standard for expressing constraints within diagrams that express the dynamic behavior of a design.

At present OCL is quite new and not widely available. ArgoUML is one of the few CASE tools to provide comprehensive support.

2.4.3. 100% Pure Java

Java was conceived as an interpreted language. It doesn't have a compiler to produce code for any particular target machine. It compiles code for its own target, the *Java Virtual Machine (JVM)Java Virtual Machine (JVM)*.

Writing an interpreter for a JVM is much easier than writing a compiler, and such machines are now incorporated into almost every Web Browser. As a result most machines can run Java, with no further work.

(In case you wonder why all languages aren't like this, it is because interpreted languages tend to be slower than compiled languages. However with the high performance of modern PCs, the trade-off for portability is worthwhile for many applications. Furthermore modern multi-level caches can mean that interpreted languages, which produce denser code, may actually not be that much slower anyway.)

By choosing to write ArgoUML in pure Java, it is immediately made available to the maximum number of users with the minimum amount of effort.

2.4.4. Open Source

ArgoUML is an *open sourceopen source* project. That means anyone can have a free copy of the source code, change it, use it for new purposes and so on. The only (major) obligation is that you pass your code on in the

same way to others. The precise nature of what you can and can't do varies from project to project, but the principle is the same.

The advantage is that a small project like ArgoUML suddenly is open to a lot of additional help from those who can chip in their ideas for how the program might be improved. At any one time their may be 10, 15, 20 or more people making significant contributions to ArgoUML. To do that commercially would cost $1m+ per year.

Its not just a spirit of pure altruism. Contributing is a way of learning "hands-on" about leading edge software. Its a way of getting a lot of visibility (over 100,000 people had downloaded ArgoUML by the spring of 2001). That's a lot of good experience on a resumé and a lot of potential employers seeing you!

And its great for the ego!

Open Source doesn't preclude making money. Gentleware www.gentleware.com [http:/-/-www.gentleware.com] sell a commercial version of ArgoUML, Poseidon. Their value proposition is not a piece of private code. Its the commercial polish and support that take risk out of using ArgoUML in a commercial development, allowing customers to take advantage of ArgoUML's leading edge technology.

2.5. ArgoUML Basics

The aim of this section is to get you started with ArgoUML. It takes you through obtaining the code and getting it running.

2.5.1. Getting Started

You have three options for obtaining ArgoUML.

1. Run ArgoUML directly from the Web Site using Java WebStart (JWS). This is the easiest option.

2. Download the binary code. This is the right option if you intend using ArgoUML regularly and is not that difficult.

3. Download the source code using CVS and build your own version. Choose this option if you want to look at the internal workings of ArgoUML, or want to join in as a developer.

All three options are freely available through the project web site, argouml.tigris.org [http:/-/-argouml.tigris.org].

If you choose to download the binary, you will have a choice of downloading the latest stable version of the code (which will be more reliable, but not have all the latest features), or the current version (which will be less reliable, but have more features). Choose according to your own situation.

This documentation was produced using ArgoUML 0.9.8, which at the time of writing was the version about to be issued.

Warning

The previous paragraph is a statement of "anticipation". The preparation of 0.9.8 and the writing of this edition of the User Manual are running in parallel. So this manual really reflects the current (unstable) development of the previous version!

2.5.1.1. ArgoUML using JWS

There are two steps to this.

1. Install JavaWebStart (JWS) on your machine. This is available from java.sun.com/products/javawebstart [http:/-/-java.sun.com/-products/-javawebstart], or via the `JavaWebStart` link on the ArgoUML home page [http:/-/-argouml.tigris.org].

2. Click on the `Launch ArgoUML 0.9.8` link on the ArgoUML home page [http:/-/-argouml.tigris.org].

JWS will download ArgoUML, cache it and start it the first time, then on subsequent starts, check if ArgoUML is updated and only download any updated parts and then start it. The ArgoUML home page [http:/-/-argouml.tigris.org] also provides details on starting ArgoUML from the JWS console.

2.5.1.2. Downloading ArgoUML

Since ArgoUML is written in 100% pure Java, it should run on any machine with a Java Engine. Java2, version 1.2 or later is needed. You may have this in place, but if not it can be downloaded free from Sun MicroSystems (www.sun.com [http:/-/-www.sun.com]). Note that you only need the Java Runtime Environment (JRE), there is no need to download the whole Java Development Kit (JDK).

ArgoUML needs a reasonable amount of computing resource. A PC with 300MHz processor, 64Mb RAM and 10Mb of disk should be adequate. Download the code from Download section of the project website argouml.tigris.org [http:/-/-argouml.tigris.org]. Choose the version that suits your needs as described in the section above.

ArgoUML comes in `.zip` or `tar.gz` flavors. Choose the former if you are a Microsoft Windows user, and the latter if you are running some flavor of Unix. Unpacking is as follows.

- On Windows. Unzip the `.zip` file with WinZip, or on later versions of Windows (ME, XP) copy the files out of the compressed folder and put them into a directory of your choosing.

- Use GNU tar to unzip and break out the files to a directory of your choice `tar zxvf <file>.tar.gz`. If you have an older version of tar, the `z` option may not be available, so use `gunzip < file.tar.gz | tar xvf -`.

You should have a directory containing the following files: `antlrall.jar`, `gef.jar`, `nsuml.jar`, `README.txt`, `argouml.jar`, `log4j.jar`, `ocl-argo.jar` and `xerces.jar`. To run ArgoUML depends on whether you are a Microsoft Windows or Unix user.

- On Windows. Start a MSDOS shell window. Use Start/Run with the default "command". In the window change to the directory holding your ArgoUML files and type `java -jar argouml.jar`

- Start a shell window and type `java -jar argouml.jar`

2.5.1.2.1. Problems Downloading

If you get completely stuck and you have no local assistance, try the web site, particularly the FAQ [http:/-/-argouml.tigris.org/-documentation/-defaulthtml/-faq/-]. If this still doesn't solve the problem, try the ArgoUML users' mailing list.

You can subscribe through the mailing lists section of the project web site argouml.tigris.org [http:/-/-argouml.tigris.org], or send an empty message to users@argouml.org [mailto:users@argouml.org] with the subject line `subscribe`.

You can then send your problem to users@argouml.org [mailto:users@argouml.org] and see how other users are able to help.

The users' mailing list is an excellent introduction to the live activity of the project. If you want to get further involved there are additional mailing lists that cover the development of the product and issues in the current and future releases.

2.5.1.2.2. Problems Running ArgoUML

It's unusual to encounter problems if you have made a successful download. Most common issue is not having a new enough Java Runtime Environment (it must be 1.2 or later).

If you can't solve the problem. Try the users' mailing list (see Section 2.5.1.2.1).

2.5.2. The ArgoUML User Interface

Figure 2.4 shows the main ArgoUML window.

Figure 2.4. Overview of the ArgoUML window
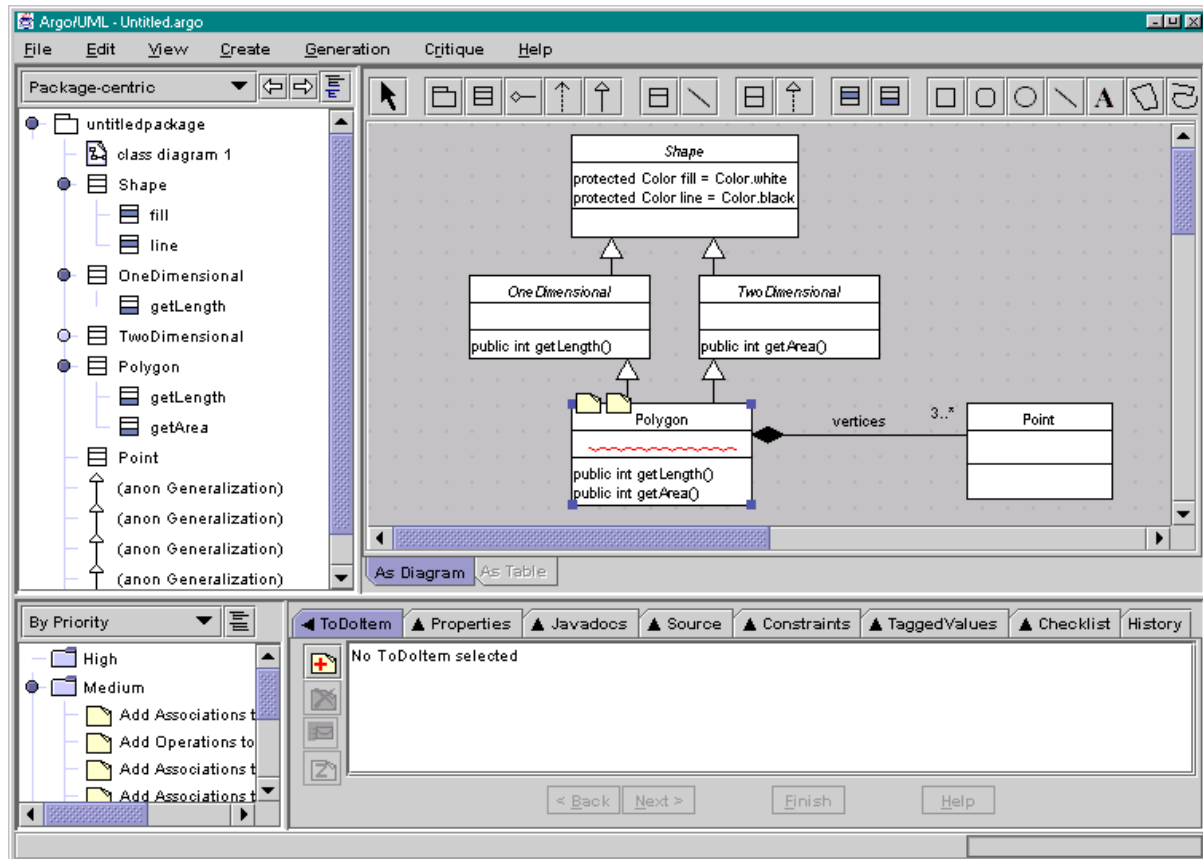
At the top of screen is a *menu barmenu bar*. The bulk of the window comprises four sub-windows or *PanesPanes*. Clockwise from top left these are the *Navigator PaneNavigator Pane*, *Editing PaneEditing Pane*, *Details PaneDetails Pane* and *To-Do PaneTo-Do Pane*. At the top of the details Pane is a *tool bartool bar*. Finally at the bottom of the window is a *status barstatus bar*.

2.5.2.1. The Navigator Pane

Figure 2.5 shows the Navigator Pane.

Figure 2.5. Overview of the Navigator Pane
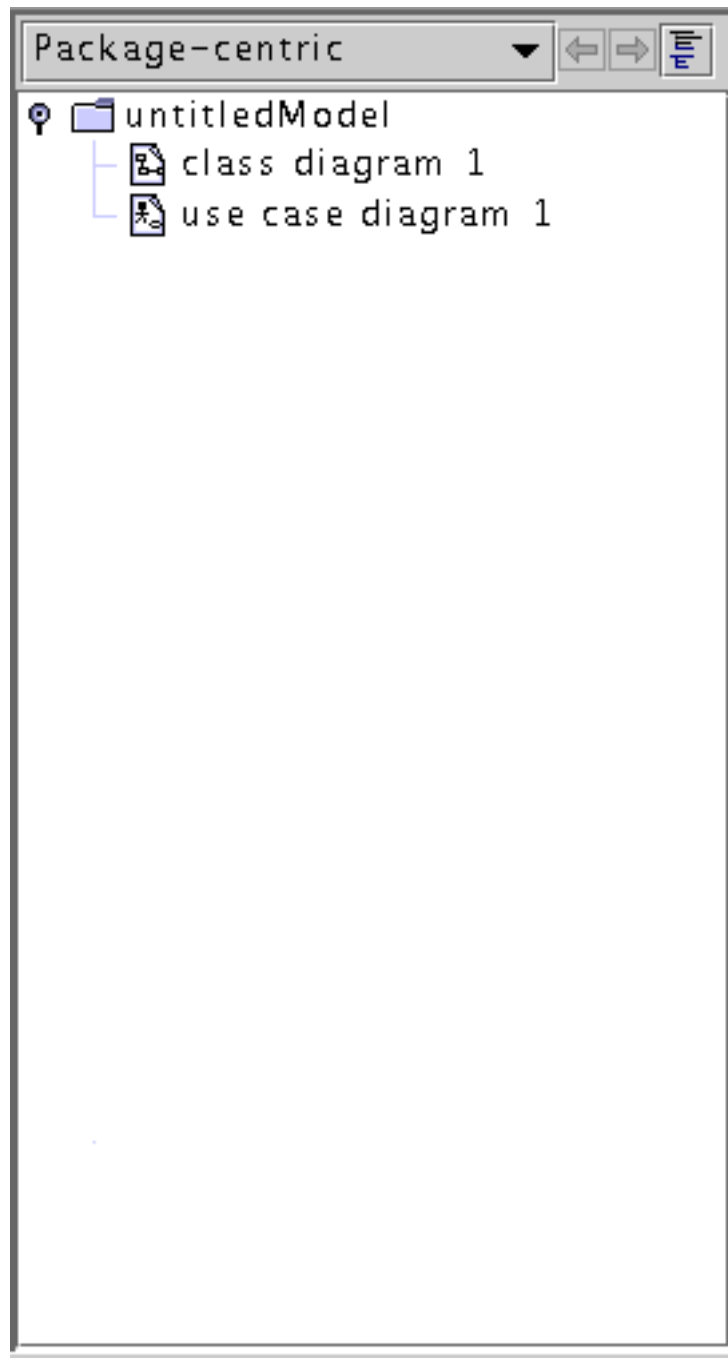
This is the navigator pane, where you can navigate through your model. This pane list all the classes, interfaces and data types of your model as a tree view.

2.5.2.2. The Editing Pane

Figure 2.6 shows the Editing Pane.

Figure 2.6. Overview of the Editing Pane
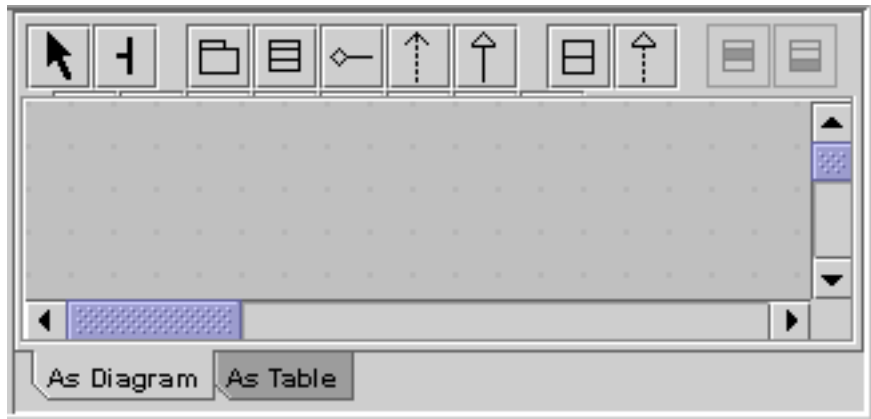
This is the Editing Pane, where you can edit your diagram. This pane lets the user edit her diagram in a graphical way.

2.5.2.3. The Details Pane

Figure 2.7 shows the Details Pane.

Figure 2.7. Overview of the Details Pane

This is the Details Pane, that allows you to edit various details of your model. The Details Pane lets the user edit various details of the model.

2.5.2.4. The To-Do Pane

Figure 2.8 shows the To-Do Pane.

Figure 2.8. Overview of the To-Do Pane
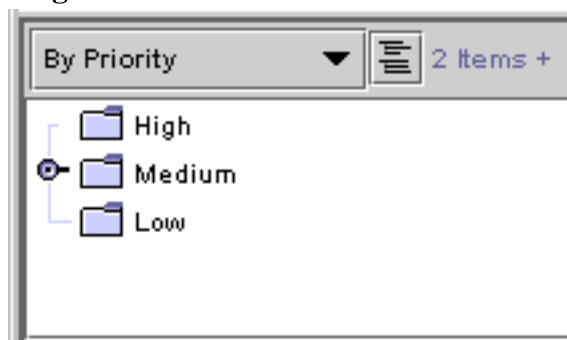
This is the to-do pane, that shows all the items on your to-do list. The to-do pane list the items on the users to-do list.

2.5.2.5. The Menu Bar

The menu bar gives access to all the main features of ArgoUML. As is conventional, menu options that are not available are grayed out and menu items that invoke a dialog box are followed by an ellipsis (…).

- *FileFile* menu. This allows you to create a new project, save and open projects, import sources from elsewhere, load and save the model to and from a database, print the model, save the graphics of the model, save the configuration of the model and exit ArgoUML

- *EditEdit* menu. This allows you to select one or more UML items on a diagram, undo and redo edits, remove items from diagrams or the whole model, empty the trash and change settings.

- *ViewView* menu. This allows you to switch between diagrams, find artifacts in the model, zoom in a diagram, select a particular diagram representation (although at present there is only one), select a particular tab in the details menu, adjust the grid, view buttons on a selection, and switch between UML and Java notation.

- *Create DiagramCreate Diagram* menu. This allows you to create any one of the seven UML diagram types (class, use case, state, activity, collaboration, deployment and sequence) supported by ArgoUML.

   ### Warning

   State and activity diagrams can only be created when a class or actor is selected, even though the relevant menu entries are *notnot* grayed out if this has not been done (nothing will happen under this circumstance).

- *ArrangeArrange* menu. This allows you to align, distribute, reorder and nudge artifacts on a diagram and set the layout strategy for the diagram.

- *GenerationGeneration* menu. This allows you to generate Java code for selected classes or all classes.

- *CritiqueCritique* menu. This allows you to toggle the auto-critique on and off, set the level of importance of design issues and design goals and browse the critics available.

- *ToolsTools* menu. This menu appears to be permanently grayed out.

- *HelpHelp* menu. This menu gives access to details of those who authored the system, and where additional help may be found.

2.5.2.6. The Mouse

The mouse and mouse buttons (or their equivalent with alternative tracking devices) are used in a wide variety of ways. In this section we look at the common modes of use.

ArgoUML assumes a two button mouse. We will refer to the buttons as "button 1" and "button 2". Button 1 is the leftmost button on a right-handed mouse, and sometimes referred to as the "select" button. Button 2 is sometimes referred to as the "adjust" button.

1. *Button 1 clickButton 1 click*. This action is generally used to select an item for subsequent operations. If the item is an artifact in the navigation pane or the editing pane it will be highlighted.

   In the case of the tool bar of the editing pane, that artifact is selected as the next to be added to the diagram (but only once—see double clicking for adding multiple artifacts). The adding to the diagram is achieved by moving the mouse to the editing area and clicking again.

2. *Button 1 double clickButton 1 double click*. This action is identical to a single click *exceptexcept*, when it is used with the editing pane tool bar. Under these circumstances the selected artifact will be added multiple times to the drawing area, once for each further button click, until the tool is again selected or another tool

chosen.

3. *Button 2 clickButton 2 click*. When used over text items in the the navigation or details panes, or graphical artifacts in the editing pane, this will display a context dependent drop-down menu.

   If the item has not yet been selected it will also be selected.

4. *Button 1 motionButton 1 motion*. Where button 1 click has been used to pull down a menu from the menu bar, button 1 motion is used to select items on that menu.

   Button 1 motion also has an effect in the editing pane. Over graphical artifacts it will move the artifact to a new position. Graphical artifacts that are selected show handles, and these can be used for re-sizing.

   Where the artifact is some form of connector between other items, button 1 motion other than at a handle will cause a new handle to be created, allowing the connector to be articulated at that point. Such new handles can be removed by moving them to the end of the connector.

5. *Button 2 motionButton 2 motion*. This is used to select items in a context sensitive menu popped up by use of button 2 click.

There are other more specific behaviors that will be encountered under the specific cases where they are used.

2.5.2.7. Drawing Diagrams

In general diagrams are drawn by using the editing pane tool bar to select the artifact desired and clicking in the diagram at the position required as described in Section 2.5.2.6. That section also explains the use of the mouse for re-sizing artifacts.

Artifacts that are already in the model, but not on a diagram, may be added to a diagram by selecting the artifact in the navigation pane, using `Add to Diagram` from the drop down menu (button 2) over that artifact, and then clicking button 1 at the desired location on the diagram.

As well as UML artifacts, the tool bar provides for general drawing artifacts (rectangles, circles, lines, polygons, curves, text) to provide supplementary information on diagrams.

2.5.2.7.1. Moving Diagram Elements

There are several ways to move diagram elements.

2.5.2.7.1.1. Using the Mouse Keys

Select the elements you want to move. By holding down the control key you can select several.

Now hit your arrow keys. Your elements move a little with every key stroke.

If you also hold down the shift key, they move a bit faster.

2.5.2.7.1.2. Using the Tool bar

Click on the broom button on the tool bar. Move your mouse to the diagram pane, right click and hold. Now moving your mouse will align elements.

2.5.2.7.2. Arranging Elements

The menu item `Arrange` allows you to align, group, or nudge elements.

2.5.2.8. Working with Projects

2.5.2.8.1. The Start-up Window

Figure 2.9 shows the ArgoUML main window as it appears as right after start-up

Figure 2.9. ArgoUML window at start-up
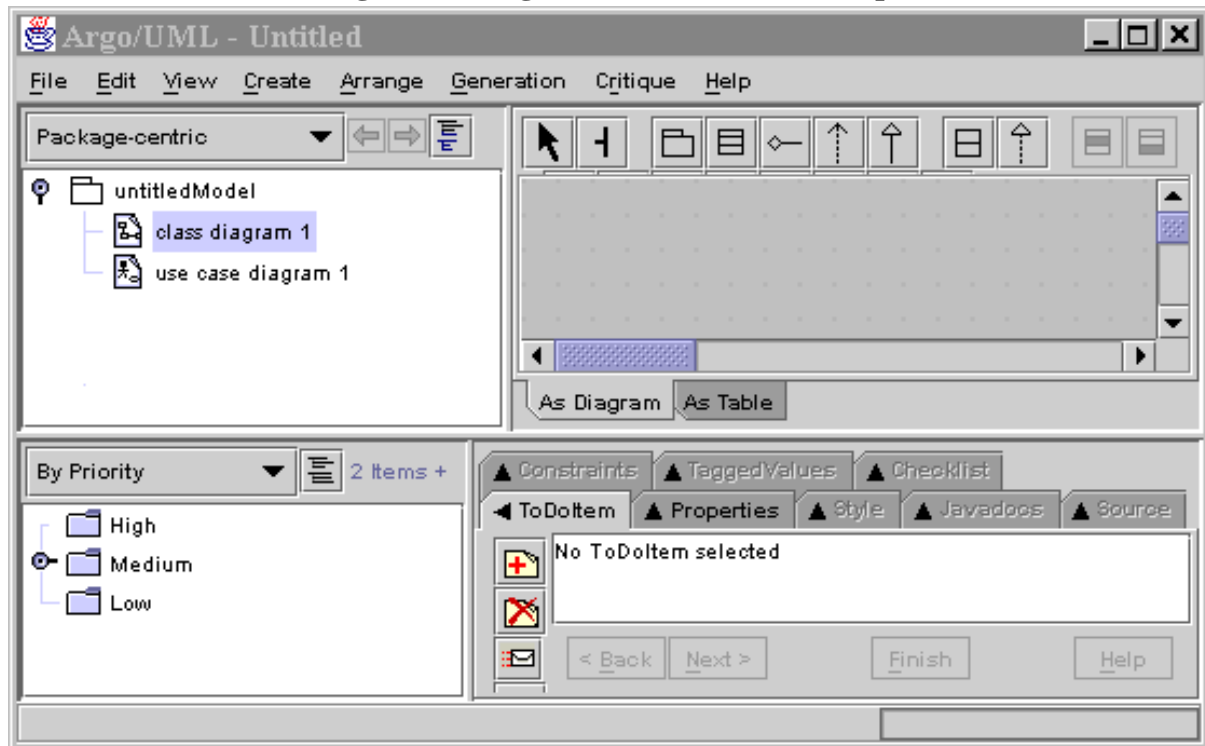
The main window's client area, below the menu bar, is subdivided into four panes. Starting at the leftmost top pane, and working around the clock, you can see the Navigation Pane, showing an explorer-like tree view of your UML model, the Editing Pane with its tool bar, two scroll bars and gray drawing area, the Details Pane with the ToDoItem tab selected, and the To-Do Pane with a tree view of the to do items, ranked into three priority levels (High, Medium and Low).

Notice that the title bar says `ArgoUML - Untitled`. The "Untitled" means that the current project has not yet been saved. After saving the current project, the "Untitled" will be replaced by the file name of the current project.

Each time ArgoUML is started up without a project file as an argument, a new blank project is created. This project contains a model called `untitledModel`. This model contains a blank Class Diagram, called `class diagram 1`, and a blank Use Case Diagram called `use case diagram 1`.

The model and both empty diagrams can be seen in the Navigation Pane, which is the main tool for you to navigate through your model.

Let's assume for a moment that this is the point where you want to start modeling a new purchasing system. You want to give the name "purchasingmodel" to your model, and you want to store it in a file called `FirstProject`.

2.5.2.8.2. Saving a project - the file menu

First, let's save the model in it's current (empty and unnamed) state. On the menu bar, click on `File`, then on `Save Project As...` as shown in Figure 2.10.

Figure 2.10. Invoking Save Project As…
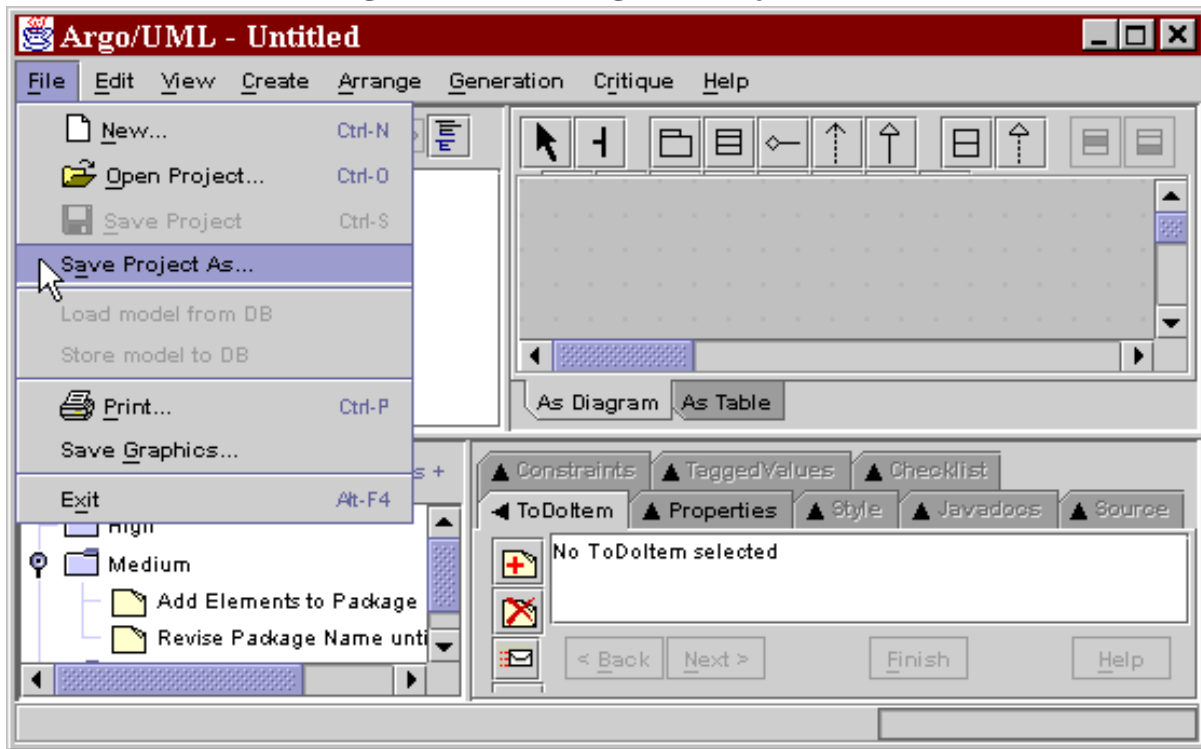
Please notice that the File menu contains the usual options for creating a new project, for opening an existing project, for saving a project under a new name, for printing the currently displayed diagram (which of course is empty at this moment), for saving the currently displayed diagram as a file, and for program Exit.

Some of these menu commands can be invoked by pressing key combinations, as indicated on the drop-down menu. For instance, holding down the "Ctrl" key, and pressing "N", will create a new project.

In the current version, ArgoUML can only contain one active project at a time. In addition, a project can only contain one UML model. Since an UML model can contain an unlimited number of elements and diagrams, this should not present any serious limitations, even for modeling quite large and complex systems.

Also take notice of a number of disabled commands, grayed out on the drop-down menu. First, the `Save Project` command (Ctrl-S), which obviously cannot be executed since we haven't entered a file name yet to store our project. Second, you can see the two commands `Load model from DB` and `Store model to DB`. This is a unique feature of ArgoUML that allows you to save and retrieve your model into a relational database. Currently, the only database supported is MySQL, but support for other database engines will undoubtedly be introduced in the future.

2.5.2.8.3. The File Chooser Dialog

But let's go back to saving our project. After clicking on the `Save Project As…` menu command, we get the file chooser dialog to enter the file name we wish to use as shown in Figure 2.11.

Figure 2.11. File chooser dialog
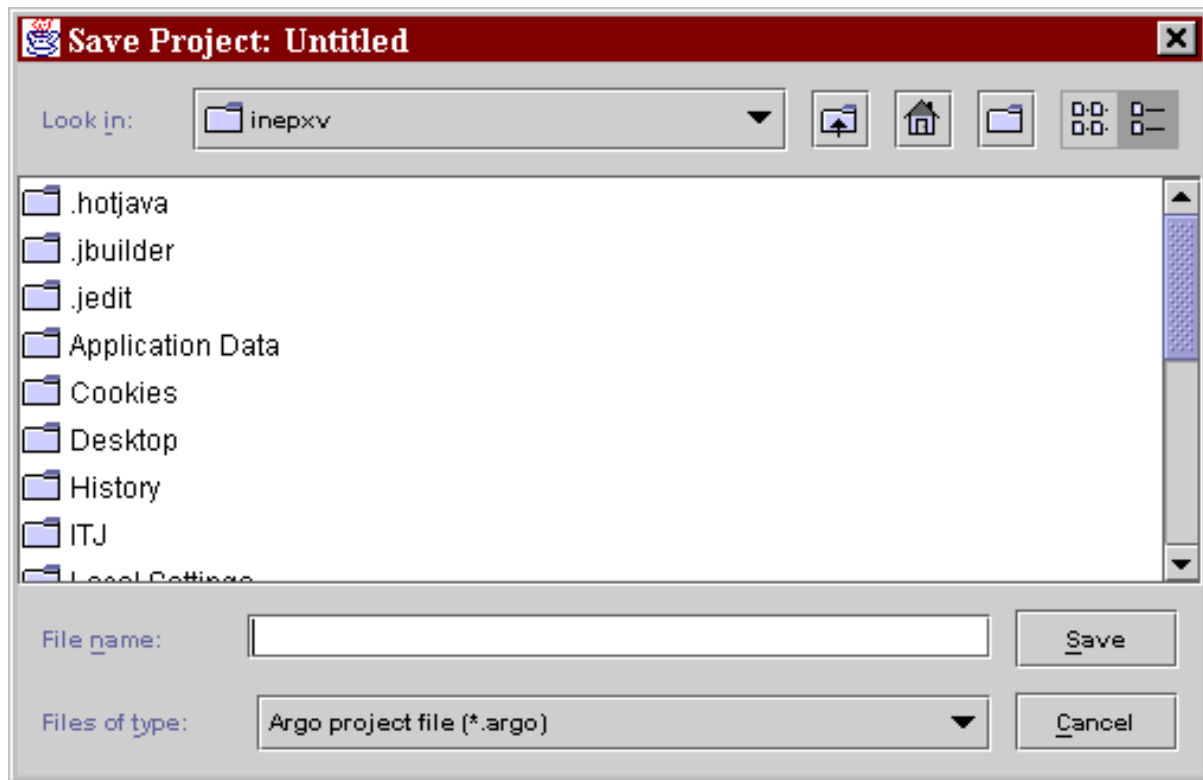
This is a standard Java FileChooser. Let's go over it in some detail.
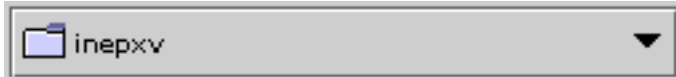
The main, outstanding feature, is the scrollable folders list in the center of the dialog. By using the scroll bar on the right, you can move up and down in the list of folders contained inside the currently selected folder.

Double-clicking on one of the displayed folders navigates you into that folder, allowing you to quickly navigate down into the folders hierarchy on your hard disk.

Notice that only folder names, and no file names are displayed in the scrollable area. Indeed, the dialog is currently set up in order to show only ArgoUML project files with an extension of .argo, as can be seen on the lower drop-down control labeled Files of type:.
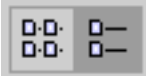
Also notice that the currently selected folder's name is displayed in the upper drop-down control labeled Look in:. A single click on a folder inside the scrollable area does select that folder on screen but does not select the folder for saving.

At the top of the dialog, above the scrollable folder chooser area, there are a few more folder navigation tools.

- The Folder drop-down control. Clicking on the down-arrow displays a tree view of the folder hierarchy, allowing you to navigate quickly up the hierarchy, and at the same time to quickly determine where in the hierarchy you are currently positioned.

- The Folder-Up icon. Clicking on this icon will bring you to the parent folder of the current folder.

- The Home Folder icon. Clicking on this icon will bring you to your home directory

- The New Folder icon. Clicking on this icon will create a new folder called "New Folder" under the

23

current folder. (I haven't been able to create a folder with a specific name, using this button. This forces me to use my file manager to rename the newly created folder. Any suggestions on how this can be done would be welcome)

- The Folders Presentation Icon. This button is not functional at this time.

OK, now navigate to the directory where you want to save your ArgoUML project, fill in the `File name:` with an appropriate name, such as "FirstProject" and click on the `Save` button.

2.5.2.8.4. File formats used by ArgoUML

Notice that the file name of your project (`FirstProject.argo`) is now reflected in the title bar of ArgoUML. If you select the java console, you will notice that three files have been written to the hard disk.

- A file `FirstProject.xmi`. This is a representation of your (still empty) model in *XML Metadata Interchange (XMI)XML Metadata Interchange (XMI)* format. If you want to know more about XMI, you can start at the following rather extensive XMI page [http:/-/-www.oasis-open.org/-cover/-xmi.html]. In short, XMI allows your models created with ArgoUML to be used by other applications. That is, only the contents of your model, without the graphic representation of the diagrams.

- A file `FirstProject_classdiagram1.pgml`, containing the graphical representation of the (still empty) class diagram `classdiagram1` in Precision Graphics Markup Language (PGML) format. PGML is probably the first proposal for sending vector graphics over the Web. You can find more information at the PGML specification page [http:/-/-www.w3.org/-TR/-1998/-NOTE-PGML].

- A file `FirstProject_usecasediagram1.pgml`, containing a graphical representation of the (still empty) use case diagram `usecasediagram1`.

You have now an active project called `FirstProject`, connected to the file `FirstProject.argo`.

2.5.3. Output

2.5.3.1. Loading and Saving

2.5.3.1.1. Saving XMI files in ArgoUML

ArgoUML saves the diagram information in a PGML file (with extension `.pgml`, the model information in an XMI file (with extension `.xmi` and information about the project in a file with extension `.argo`. See Section 2.5.3.2.2 and Section 2.5.3.3 for more about PGML and XMI respectively.

All of these are then zipped to a file with extension `.zargo`. You can easily extract the `.xmi` file from the `.zargo` file using any old generic ZIP application. Give it a try and look into the magic of Argo.

> ### Warning
>
> Be aware that double clicking will launch a `ZIP utility`, if one is installed, and NOT Argo.

2.5.3.2. Graphics and Printing

2.5.3.2.1. The Graph Editing Framework (GEF)

GEF, the thing which is hidden in gef.jar, is responsible for displaying the graphs and which gives you the the ability to move things around. GEF used to be closely interwoven with Argo, but is now more or less separate. The sources are also available via www.tigris.org [http:/-/-www.tigris.org].

2.5.3.2.2. Precision Graphics Markup Language (PGML)

PGML is the saving format for diagram information used in ArgoUML. Although this is outdated a little, it shows why we chose PGML in the first place. In the future, PGML will be replaced by an extended XMI format which will include graphical representation.

In April 1998 Adobe Systems proposed a new Standard for Vector Graphics to the World Wide Web Consortium (W3C), which should facilitate the creation and usage of image material in the World Wide Web. It will be replaced by SVG. We will support SVG by generating it from the XMI in the future.

2.5.3.2.3. Applications Which Open PGML

PGML is a predecessor of SVG (see Section 2.5.3.2.5. It was dropped by the W3C Consortium.

Currently there are no other tools that we know of working on PGML. We will move to SVG soon. And *thatthat* will be read by many tools.

2.5.3.2.4. Printing Diagrams

Select a diagram, then go to `File#Export Diagrams`. You can generate GIF, PostScript, Encapsulated PostScript or SVG format.

2.5.3.2.5. Standard Vector Graphics (SVG)

A super cool vector graphics format (like MM Flash) that is totally open and is gaining enormous ground in the graphics world. Find out more at www.w3c.org [http:/-/-www.w3c.org].

As well visit adobe.com [http:/-/-www.adobe.com] for the plugin for Netscape Navigator and Internet Explorer.

2.5.3.2.6. Saving Diagrams as SVG

1. Select `.svg` as the file type.

2. Type the name of the file as you like with the `.svg` tag at the end. Example `myumldiagram.svg`

Warning

ArgoUML does not do this automatically. See the to-do list in Chapter 14.

Et viola! SVG! Give it a try and zoom around a little… They are not pretty though, so if you know anything about rendering beautiful SVG let us know!

Oh, and by the way, you'll need a browser that can read SVG. See adobe.com [http:/-/-www.adobe.com] for the plugin for Netscape Navigator and Internet Explorer

There are a few browsers that read SVG natively: Amaya, X-Smiles, and a rudimentary version of Mozilla.

Note

You will not have scroll bars for your SVG unless it is embedded into an HTML! Good luck and let us know what you find!

2.5.3.3. XMI

There is now a tool available which supports various dialects of XMI and their interconversion. This should make portability of different models in to ArgoUML considerably easier. For more information, see http://umltool.d-a-t.com [http:/-/-umltool.d-a-t.com/-]. Any work on using this with ArgoUML would be much

appreciated.

There is also a tool that converts XMI to HTML. For more information, see http://www.objectsbydesign.com/projects/xmi_to_html_2.html [http:/-/-www.objectsbydesign.com/-projects/-xmi_to_html_2.html].

2.5.3.3.1. Using XMI from Rational Rose

It is probably not possible to take a model generated by Rational Rose in XMI format and put it into ArgoUML directly.

The conversion tool identified above will probably solve the problem. If you can document how this is done, you'd make a lot of friends in a hurry!

2.5.3.3.2. Using models created by Gentleware

Gentleware is a commercial development based on ArgoUML. It diverged a little while ago and the XMI representations are no longer full compatible.

Eugenio Alvarez suggests the following procedure to convert a Gentleware project for use with ArgoUML. It works if there are no sequence diagrams in the project.

1. Make sure to backup your project first.

2. Unpackage the Gentleware project Project.zargo. For example `unzip  Project.zargo` or `jar xvf Project.zargo`.

3. Then try reading and then saving the project with http://umltool.d-a-t.com [http:/-/-umltool.d-a-t.com].

4. Repackage the project. For example `zip  NewProject.zargo  *.xmi  *.pgml  *.argo` or `jar cvf NewProject.zargo *.`

5. Try reading the project with ArgoUML.

2.5.3.3.3. XMI Interchangeability between ArgoUML 0.8.1a and 0.9.x

This depends on the version of NSUML (the UML engine underling ArgoUML) you are using. You can find out about this i your start-up messages on console). Using version 0.4.19 is OK, should be interchangeable.

2.5.3.3.4. Importing Other XMI Formats into ArgoUML

Because of the variety of XMI formats being generated you will encounter problems using XMI files generated from other applications. ArgoUML's native storage format is another implementation of XMI (Curt Arnold believes that it has been hacked so it can also read the Unisys variant). He recommends that if you are comfortable with XSLT, you can typically transform one variant of UML and XMI to another, plus MetaIntegrations www.metaintegrations.net [http:/-/-www.metaintegrations.net] makes a version of their Bridge product available to transform between XMI flavors.

2.5.3.3.5. Generating XMI Format

Unzip the `.zargo` file. Open it and Viola! It's tucked inside.

2.5.3.4. Code Generation

2.5.3.4.1. Code generated by ArgoUML

It is possible to compile your generated code with ArgoUML, you still need to implement method bodies,

though, to get usable results.

2.5.3.4.2. Generating Code for Methods

At the moment you cannot write code for methods (operations) within ArgoUML. The source pane is editable, but the changes are ignored. ArgoUML is a pure design tool for now, no IDE functionality but the desire is there. You might consider using Forte and ArgoUML together—it's a good work around!

You can help us out there if you'd like!

2.5.4. Working With Design Critics

2.5.4.1. The To-Do Pane—Messages From the Design Critics

Where do we stand now? A new project has been created, and is connected to the file FirstProject.argo. Figure 2.12 shows how your ArgoUML window should look at this stage.

Figure 2.12. ArgoUML window having save FirstProject.argo
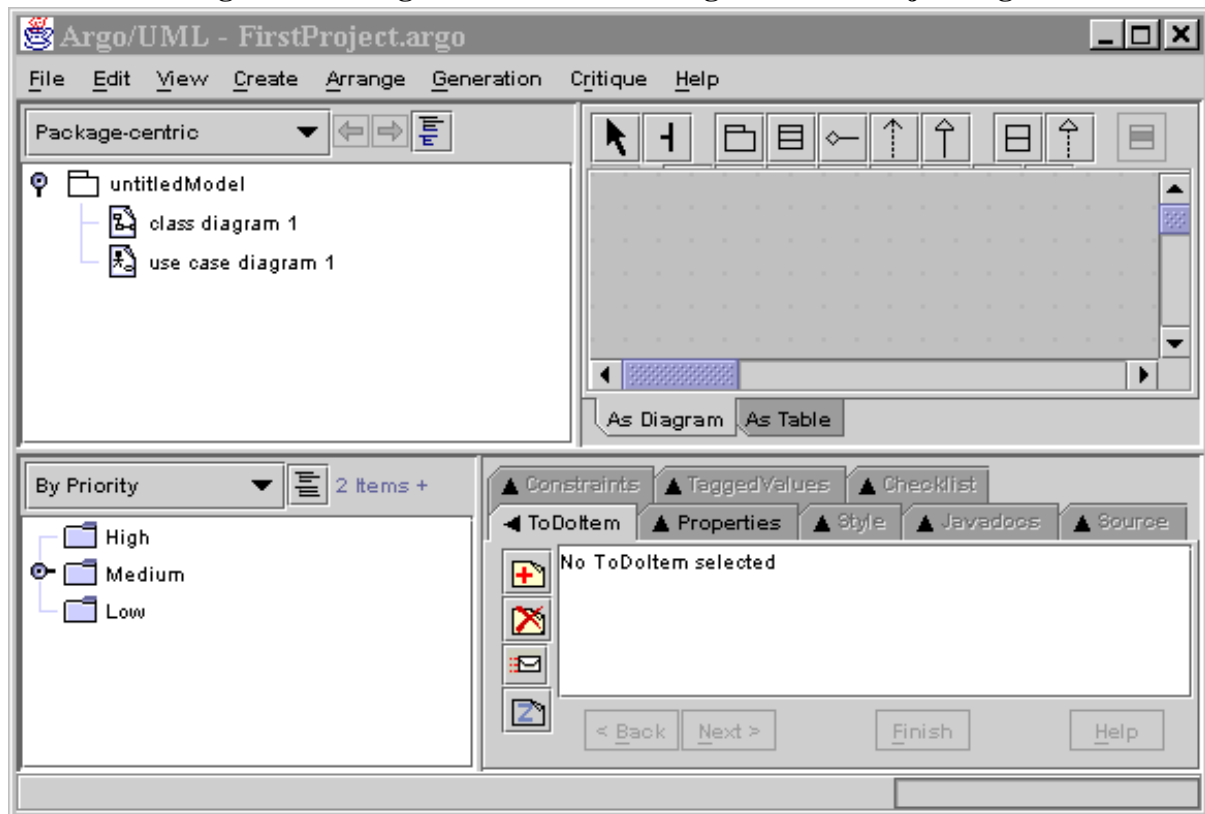
The project filename is displayed in the title bar, the project contains a top-level package, called untitled-Model, which contains a class diagram and a use case diagram.

If we look carefully at the screen, we can see that the "Medium" folder in the To Do pane (the lower left pane) must contain some items, since its activation icon ⊙━ is displayed.

Clicking on this icon will open the "Medium" folder. An open folder is indicated by the ⊙ icon.

But what is this "To-Do" Pane anyway. You haven't recorded anything yet that has to be done, so where do

these to do items originate.

The answer is simple, and is at the same time one of the strong points of ArgoUML. While you are working on your UML model, your work is monitored continuously and invisibly by a piece of code called a *design criticdesign critic*. This is like a personal mentor that watches over your shoulder and notifies you each time he sees something questionable in your design.

Critics are quite unobtrusive. They give you a friendly warning, but they do not force you into design principles that you don't want or like to follow. Let us take a look at what the critics are telling us. Click on the [icon] icon next to the `Medium` folder, and click on the `Revise Package Name UntitledModel` item.

Figure 2.13 shows how your screen should now look.

Figure 2.13. ArgoUML window showing the critic item
Revise Package Name UntitledModel
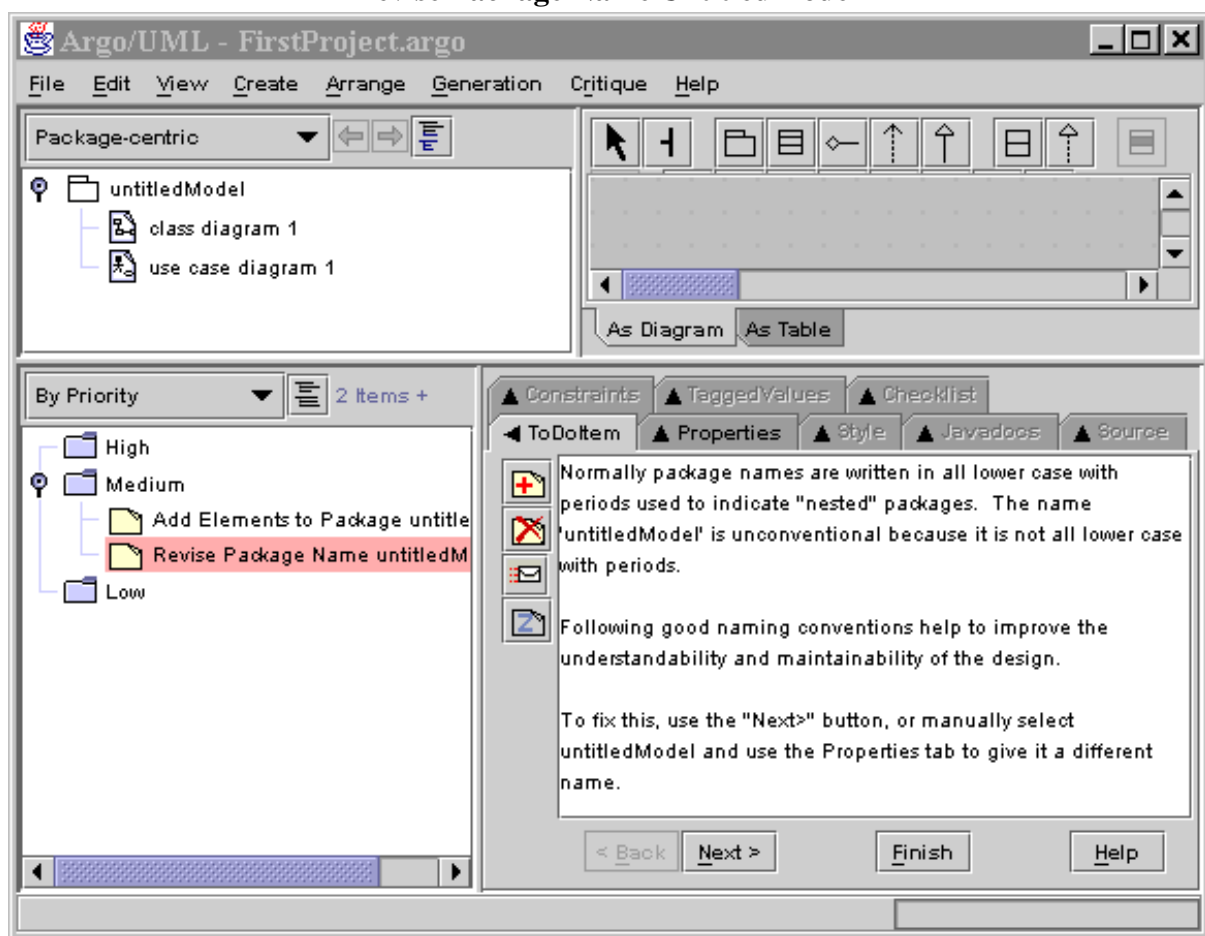
Notice that your selection is highlighted in red in the To-Do Pane, and that a full explanation appears now in the Details Pane (the lower right pane). You may have to re-size your Details Pane or to scroll down in order to see the full message as displayed in our example.

What ArgoUML is trying to tell you is that usually, package names are written in lower cases. The default top level package created by ArgoUML is called `untitledModel` and therefore violates a sound design principle. (Actually, this could be considered as a bug within ArgoUML, but it comes in handy to demonstrate the working of critics).

At this point, you can choose to change the package name manually, to impose silence on the design critic for some time or permanently, or to request a more comprehensive explanation by Email from an expert.

We will do nothing of this (we'll come back to it when we talk about the design critics in more detail) but we'll use another handy feature of ArgoUML—an auto-correct feature.

In order to do that, just click on the `Next` button on the Details Pane. This will cause a renaming wizard to be displayed inside the properties panel, proposing to use the name `untitledmodel` (all in lower case).

2.5.4.2. Design Critics at Work: The Rename Package Wizard

Replace the name `untitledmodel` with `purchasingmodel`, and click on the `Finish` button. Figure 2.14 shows how the ArgoUML window will now look.

Figure 2.14. ArgoUML window showing the critic wizard to rename the package
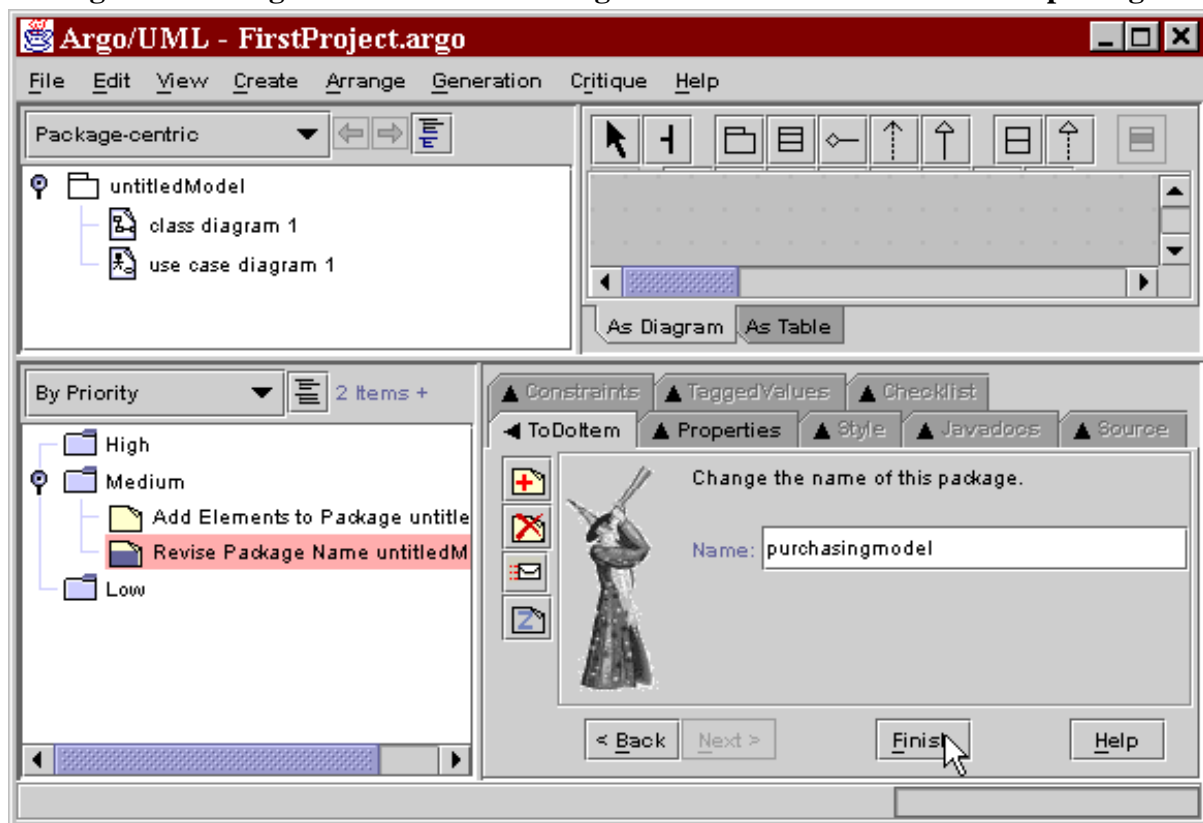
Watch now how the design critic note in the To Do panel disappears, leaving only the `Add Elements to Package purchasingmodel` note in the To-Do list.

If this doesn't happen at once, wait for a few seconds. ArgoUML makes heavy use of several threads of execution that execute in parallel. This can cause delays of a few seconds before the information gets updated on the screen.

Normally, the package name change should also be reflected in the navigation pane, in the top left corner of your ArgoUML window.

Actually, that doesn't happen until you click on one of the items in the navigator pane. This is also a small bug in ArgoUML that will need to be fixed in following versions.

Don't let this discourage you, but remember that ArgoUML is still pretty much "work in progress", and that still quite a bit can be done to improve the user interface.

We are now ready to create our first UML diagram, a Use Case diagram, but first let's save what we've done so far.

Click on the `File` menu item, and select `Save Project`. You can now safely exit ArgoUML without losing your work so far, or go on creating your first diagram.

2.6. The Case Study

To be written...

Chapter 3. Requirements Capture

Table of Contents

3.1. Introduction

Requirements capture is the process of identifying what the "customer" wants from the proposed system.

The key at this stage is that we are in the problem domain. At this stage we must describe everything from the "customer" perspective and in the language of the "customer".

The biggest risk we have in requirements capture is to start thinking in terms of possible solutions. That must wait until the *Analysis PhaseAnalysis Phase* (see Chapter 4). One of the steps of the Analysis Phase will be to take the output of the Requirements Phase and recast it in the language of a putative solution.

Remember we are using both a *recursiverecursive*, and an *iterativeiterative* process.

We may well come back to the requirements process again as we break down the problem into smaller chunks, each of which must have its requirements captured.

We will certainly come back through the requirements phase on each iteration as we seek to define the requirements of more and more of the system

3.2. The Requirements Capture Process
To be written...

3.2.1. Process Steps
To be written...

3.2.2. Output of the Requirements Capture Process
To be written...

3.2.2.1. Use Case Diagram

To be written...

3.2.2.2. Documents

Vision, Use Case Specifications and Supplementary Requirements Specifications. To be written...

3.3. Use Cases

Use Cases model the interaction between "users" (animate or inanimate) and the system. In creating Use Cases we define the system boundaries.

3.3.1. The Use Case Diagram

To be written...

3.3.2. The Use Case Specification

To be written...

3.3.3. Advanced Use Case Diagrams

To be written...

3.3.3.1. Includes

To be written...

3.3.3.2. Extends

To be written...

3.3.3.3. Generalization

To be written...

3.4. Creating Use Cases in ArgoUML

3.4.1. Actors

To be written...

3.4.2. Use Cases

To be written...

3.4.3. Associations

To be written...

3.4.4. Advanced Use Case Features

3.4.4.1. Includes

To be written...

3.4.4.2. Extends
To be written...

3.4.4.3. Generalization
To be written...

3.4.4.4. Stereotypes
To be written...

3.5. Case Study

3.5.1. Vision Document
To be written...

3.5.2. Identifying Actors and Use Cases
To be written...

3.5.3. Associations
To be written...

3.5.4. Advanced Diagram Features
To be written...

3.5.5. Use Case Specifications
To be written...

3.5.6. Supplementary Requirements Specification
To be written...

Chapter 4. Analysis

Table of Contents

4.1. Introduction

Analysis is the process of taking the "customer" requirements and re-casting them in the language of, and from the perspective of, a putative solution.

We are not actually trying the flesh out the detailed solution at this stage. That occurs in the *Design De-Phasesign Phase* (see Chapter 5).

Unlike the boundary between Requirements and Analysis Phases, the boundary between Analysis and Design Phases is inherently blurred. The key is that analysis should define the solution no further than is necessary to specify the requirements in the language of the solution. The artifacts in Analysis generally represent a high level of abstraction.

Once again the *recursiverecursive*, and *iterativeiterative* nature of our process means we will come back to the Analysis phase many times in the future.

4.2. The Analysis Process

Include CRC cards. To be written...

4.2.1. Output of the Analysis Process

To be written...

4.2.1.1. Concept Diagram

To be written...

4.2.1.2. System Sequence Diagram

To be written...

4.2.1.3. System State Diagram

To be written...

4.2.1.4. Realisation Use Case Diagram

To be written...

4.2.1.5. Documents

Use Case Specifications and Supplementary Requirements Specifications recast in solution language. To be written...

4.3. Class Diagrams

To be written...

4.3.1. The Class Diagram

To be written...

4.3.2. Advanced Class Diagrams

To be written...

4.3.2.1. Association Classes

To be written...

4.4. Creating Class Diagrams in ArgoUML

4.4.1. Classes

Identifying class diagrams from existing materials (Vision, Use Cases etc). To be written...

4.4.1.1. Using the Note Icon in the Tool Bar

Click on your target class. Then click on the note icon. ArgoUML will generate the link automatically.

You can also right click to add a note as well! Be aware that you can add an undefined number of notes to any one class!

Warning

Be aware that your note will not appear in the source code documentation tab.

4.4.2. Associations

To be written...

4.4.2.1. Aggregation

To be written...

4.4.3. Class Attributes and Operations

To be written...

4.4.3.1. Entering Data Into Attributes and Methods Windows

Click directly in the class artifact and start typing. Do not use the properties window dialog fields—they are not fully functional and liable to cause you a little frustration.

In fact, it would be interesting to see if you can type stereotypes write in the class attribute box for generating XML diagrams.

4.4.3.2. Class Attributes

To be written...

4.4.3.3. Class Operations

To be written...

4.4.4. Advanced Class Features

4.4.4.1. Association Classes

To be written...

4.4.4.2. Stereotypes

To be written...

4.5. Sequence Diagrams

To be written...

4.5.1. The Sequence Diagram

To be written...

4.5.2. Identifying Actions

To be written...

4.5.3. Advanced Sequence Diagrams

To be written...

4.6. Creating Sequence Diagrams in ArgoUML

4.6.1. Sequence Diagrams

4.6.1.1. Creating a Sequence Diagram

Normally, you can just start a sequence diagram right away. On the `Create Diagram` menu choose `Sequence`.

4.6.1.2. Known Issues with Sequence Diagrams

The following problems are currently identified.

- Checklist is not enabled with Sequence artifacts.

- It is not possible (as yet) to create call backs.

- You cannot change the order of the connectivity of a link artifact in ArgoUML yet.

- StimulusDestroy is not dependable.

4.6.2. Actions

To be written...

4.6.3. Advanced Sequence Diagrams

To be written...

4.7. State Diagrams

To be written...

4.7.1. The State Diagram

Types of state diagram (Moore, Mealy); Hierarchical diagrams. To be written...

4.7.2. Advanced State Diagrams

To be written...

4.7.2.1. Hierarchical State Diagrams

To be written...

4.8. Creating State Diagrams in ArgoUML

4.8.1. State Diagrams

To be written...

4.8.1.1. Creating a State Diagram

Select a class, then you can create a state diagram.

4.8.2. States

To be written...

4.8.2.1. Editing a Composite State

When editing a composite state, how do you provide do and event for a composite state?

The answer is to select a class, then you can create a state diagram.

4.8.3. Transitions

To be written...

4.8.4. Actions

To be written...

4.8.5. Advanced State Diagrams

To be written...

4.8.5.1. Hierarchical State Diagrams

To be written...

4.9. Realisation Use Cases

To be written...

4.10. Creating Realisation Use Cases in ArgoUML

To be written...

4.11. Case Study

4.11.1. Concept Class Diagrams

To be written...

4.11.1.1. Identifying classes

To be written...

4.11.1.2. Identifying associations

To be written...

4.11.2. System Sequence Diagrams

To be written...

4.11.2.1. Identifying actions

To be written...

4.11.3. System State Diagrams

To be written...

4.11.4. Realisation Use Cases

To be written...

Chapter 5. Design

Table of Contents

5.1. Introduction

We now have the problem we are trying to solve specified in the language of a putative solution. In the Design Phase, we construct all the details of that solution.

The blurred boundary between Analysis and Design is reflected in their use of many of the same UML tools. In this chapter we will mostly be reusing UML technology we have already met once. The big step is casting everything into concrete terms. We move from the abstract concepts of analysis to their concrete realisation.

Once again the *recursiverecursive*, and *iterativeiterative* nature of our process means we will come back to the Design phase many times in the future.

5.2. The Design Process

To be written...

5.2.1. Output of the Design Process

To be written...

5.2.1.1. Package Diagram

To be written...

5.2.1.2. Realization Class Diagrams

To be written...

5.2.1.3. Sequence Diagrams and Collaboration Diagrams

To be written...

5.2.1.4. State Diagrams and Activity Diagrams

To be written...

5.2.1.5. Deployment Diagram

To be written...

5.2.1.6. Documents

System Architecture. To be written...

5.3. Package Diagrams

To be written...

5.3.1. The Package Diagram

To be written...

5.3.2. Advanced Package Diagrams

To be written...

5.3.2.1. Subpackages

To be written...

5.3.2.2. Adding DataTypes

To be written...

5.3.2.3. Adding Stereotypes

To be written...

5.4. Creating Package Diagrams in ArgoUML

5.4.1. Packages

How to work out what goes in packages. To be written...

5.4.1.1. Subpackages

To be written...

5.4.2. Relationships between packages

To be written...

5.4.2.1. Dependency

To be written...

5.4.2.2. Generalization

To be written...

5.4.2.3. Realization and Abstraction

To be written...

5.4.3. Advanced Package Features

To be written...

5.4.3.1. Creating New Datatypes

To be written...

5.4.3.2. Creating New Stereotypes

To be written...

5.5. More on Class Diagrams

To be written...

5.5.1. The Class Diagram

To be written...

5.5.1.1. Class Attributes

To be written...

5.5.1.2. Class Operations
To be written...

5.5.2. Advanced Class Diagrams
To be written...

5.5.2.1. Realization and Abstraction
To be written...

5.6. More on Class Diagrams in ArgoUML

5.6.1. Classes
More on identifying classes from existing materials and use of stereotypes. To be written...

5.6.2. Class Attributes and Operationss
To be written...

5.6.2.1. Class Attributes
To be written...

5.6.2.2. Class Operations
To be written...

5.6.3. Advanced Class Features

5.6.3.1. Operations on Interfaces

5.6.3.1.1. Interfaces that extend interfaces

Add a unnamed interface to the current classdiagram by single-clicking on the interface icon in the tool bar and then clicking at the diagram pane (see Figure 5.1).

Figure 5.1. Selecting the Interface tool
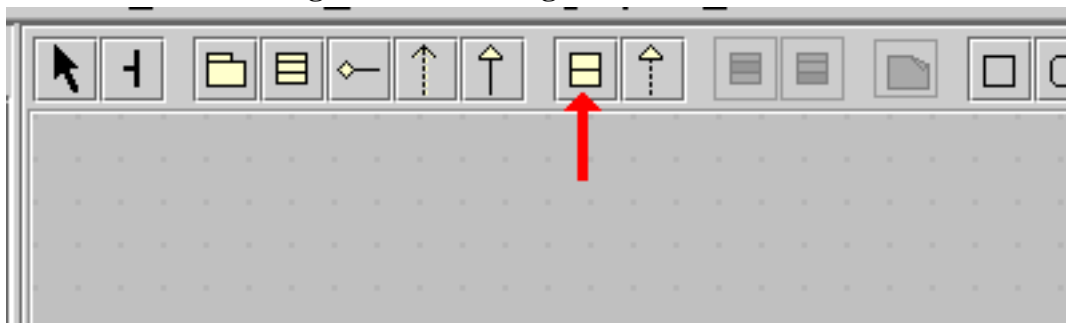
Then double click on the interfaces name field to change it's name as shown in Figure 5.2.

Figure 5.2. Interface artifact on the Class Diagram
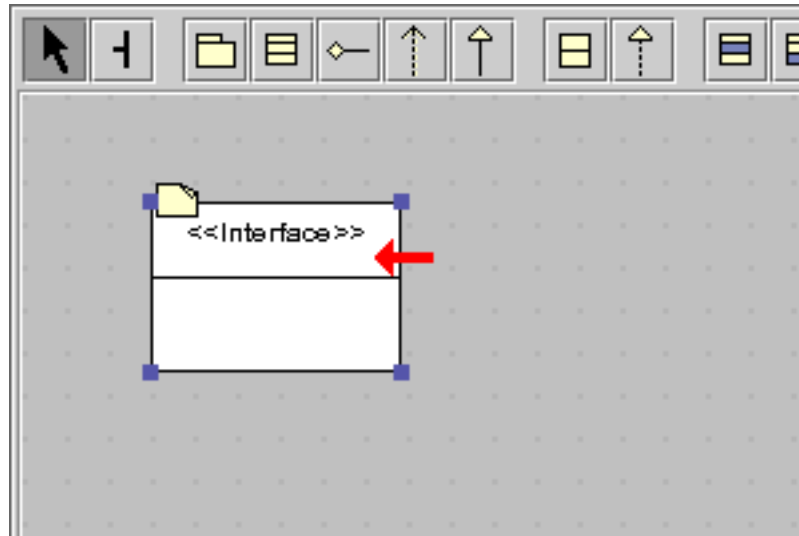
and type a name for it (like `TestInterface` in this case). Press "Enter" when the name is complete. (You could also enter the name by going to the Properties Tab in the Details Pane after adding the interface.)

Add another interface with a different by repeating the last 2 steps. Then single-click on the Generalization icon in the tool bar as shown in Figure 5.3.

Figure 5.3. Generalization on the Class Diagram tool bar
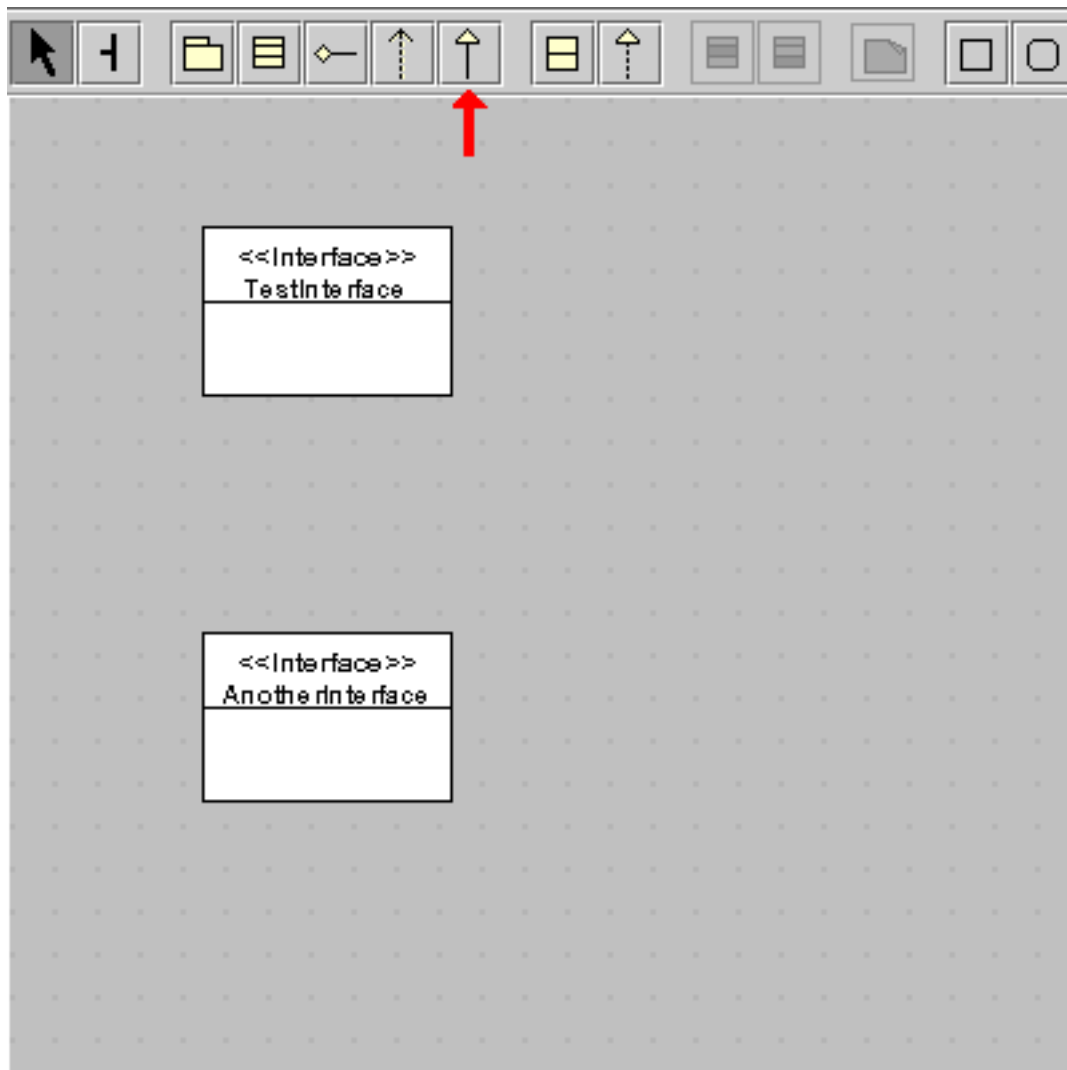
move the mouse pointer to the subinterface, press the left mouse button and drag the generalization to the super-interface, where you release the mouse button. Figure Figure 5.4 shows how your diagram should look now.

Figure 5.4. Generalization between two Interfaces.
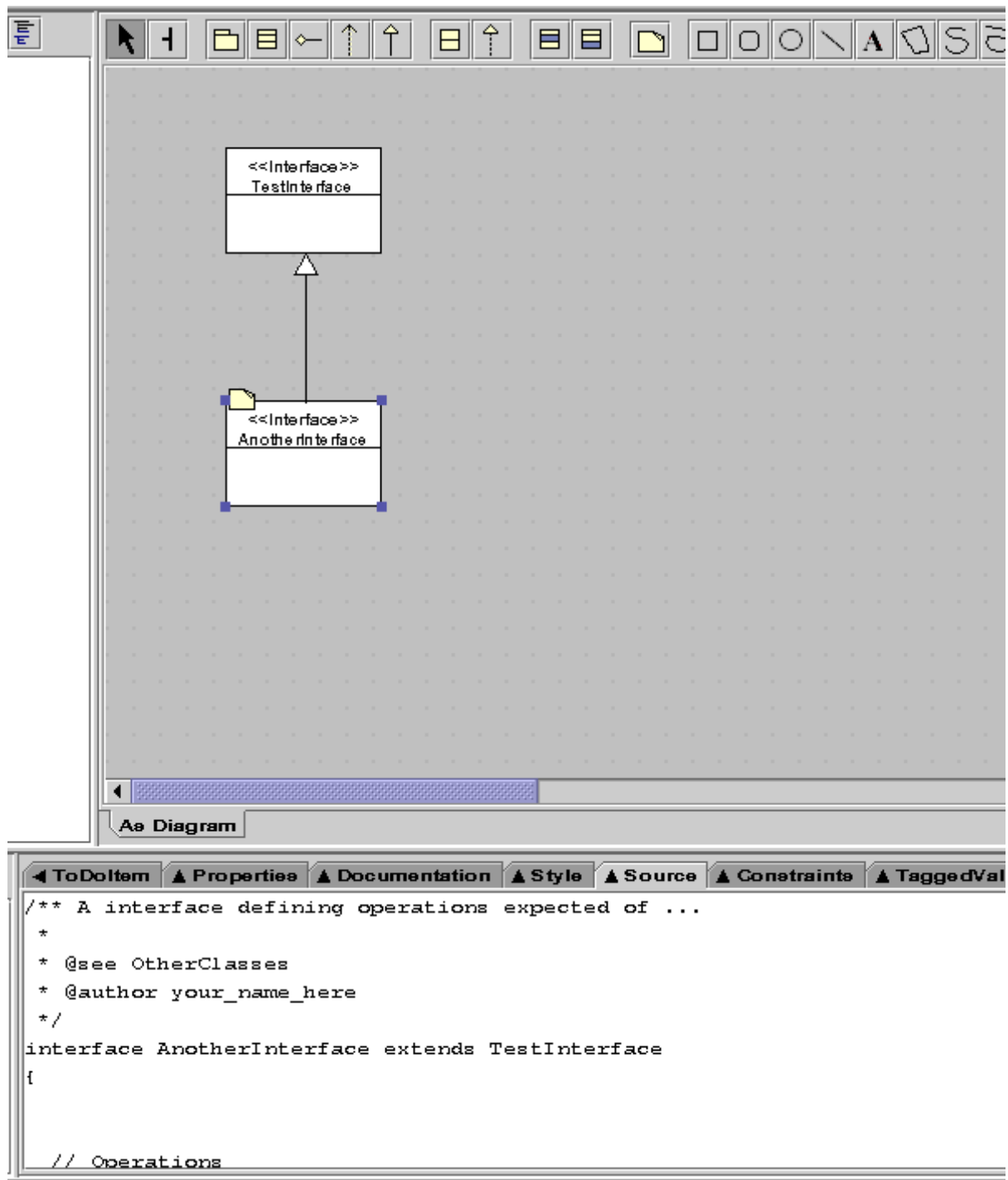
By clicking on the subinterface and the properties pane, you can see that the interface actually extends it's super-interface.

5.6.3.2. Stereotypes

To be written...

5.7. Sequence and Collaboration Diagrams

To be written...

5.7.1. More on the Sequence Diagram

To be written...

5.7.2. The Collaboration Diagram

To be written...

5.7.2.1. Messages

To be written...

5.7.2.2. Actions

To be written...

5.7.3. Advanced Collaboration Diagrams

To be written...

5.8. Creating Collaboration Diagrams in ArgoUML

5.8.1. Collaboration Diagrams

To be written...

5.8.2. Messages

To be written...

5.8.2.1. Actions

To be written...

5.8.3. Advanced Collaboration Diagrams

To be written...

5.9. State Diagrams

To be written...

5.9.1. The State Diagram

More on this. To be written...

5.9.2. Advanced State Diagrams

To be written...

5.9.2.1. Actions

To be written...

5.9.2.2. Transitions

To be written...

5.9.2.2.1. Triggers

To be written...

5.9.2.2.2. Guards

To be written...

5.9.2.2.3. Effectss

To be written...

5.9.2.3. Pseudo States

To be written...

5.9.2.3.1. Branch

To be written...

5.9.2.3.2. Fork and Join

To be written...

5.9.2.4. Hierarchical State Machines

To be written...

5.9.2.5. Models for State History

Shallow v Deep. To be written...

5.10. Creating State Diagrams in ArgoUML

5.10.1. State Diagrams

To be written...

5.10.2. States

To be written...

5.10.3. Transitions

To be written...

5.10.4. Actions

To be written...

5.10.5. Advanced State Diagrams

To be written...

5.10.5.1. Transitions

To be written...

5.10.5.1.1. Triggers

To be written...

5.10.5.1.2. Guards

To be written...

5.10.5.1.3. Effectss

To be written...

5.10.5.2. Pseudo States

To be written...

5.10.5.2.1. Branch

To be written...

5.10.5.2.2. Fork and Join

To be written...

5.10.5.3. Hierarchical State Machines

To be written...

5.10.5.4. History

Shallow v Deep. To be written...

5.11. Activity Diagrams

To be written...

5.11.1. The Activity Diagram

More on this. To be written...

5.11.1.1. Action States

To be written...

5.12. Creating Activity Diagrams in ArgoUML

5.12.1. Activity Diagrams

To be written...

5.12.1.1. Creating an Activity Diagram

Select a use case or class, then you can create an activity diagram.

5.12.2. Action States

To be written...

5.13. Deployment Diagrams

To be written...

5.13.1. The Deployment Diagram

To be written...

5.14. Creating Deployment Diagrams in ArgoUML

5.14.1. Nodes

To be written...

5.14.1.1. Node Instances

To be written...

5.14.2. Components

To be written...

5.14.2.1. Component Instances

To be written...

5.14.3. Relationships between nodes and components

To be written...

5.14.3.1. Dependency

To be written...

5.14.3.2. Associations

To be written...

5.14.3.3. Links

To be written...

5.15. System Architecture

To be written...

5.16. Case Study

5.16.1. Packages

To be written...

5.16.1.1. Identifying Packages

To be written...

5.16.1.2. Datatypes and Stereotypes

To be written...

5.16.2. Class Diagrams

To be written...

5.16.2.1. Identifying classes

To be written...

5.16.2.2. Identifying associations

To be written...

5.16.2.3. Specifying Attributes and Operations

To be written...

5.16.3. Sequence Diagrams

To be written...

5.16.3.1. Identifying actions

To be written...

5.16.4. Collaboration Diagrams

To be written...

5.16.4.1. Identifying Messages

To be written...

5.16.5. State Diagrams

To be written...

5.16.6. Activity Diagrams

To be written...

5.16.7. The Deployment Diagram

To be written...

5.16.8. The System Architecture

To be written...

Chapter 6. Build

Table of Contents

6.1. Introduction

We now have out design fully specified. With the right simulator we could actually execute the design and see if it works. ArgoUML does not provide such functionality, but this functionality has been provided in alternative tools.

ArgoUML does allow you to generate code from the design. In the Build Phase take the design we have and supplement it with code for the detailed actions of the classes and their states.

Central to this will be our System Architecture. It is this specification that provides the framework against which all our code will operate.

Once again the *recursiverecursive*, and *iterativeiterative* nature of our process means we will come back to the Design phase many times in the future.

This chapter is a little different to those that went before. UML itself is a notation for design. There is no new UML notation to understand here.

6.2. The Build Process

To be written...

6.2.1. Output of the Build Process

To be written...

6.2.1.1. Code

To be written...

6.2.1.2. Documents

To be written...

6.3. Adding Code to ArgoUML

6.3.1. Classes

To be written...

6.3.2. Actions

To be written...

6.4. Case Study

6.4.1. Coding for Classes

To be written...

6.4.2. Coding Actions

To be written...

6.5. Case Study

To be written...

Part II. ArgoUML Reference Manual

Chapter 7. User Interface Reference

Table of Contents

7.1. Introduction

This chapter describes the overall behavior of the user interface. Description of the various component parts—the menu bar, panes and various diagrams— is in separate chapters.

7.2. Overview of the Window

Figure 7.1 shows the main ArgoUML window.

Figure 7.1. Overview of the ArgoUML window
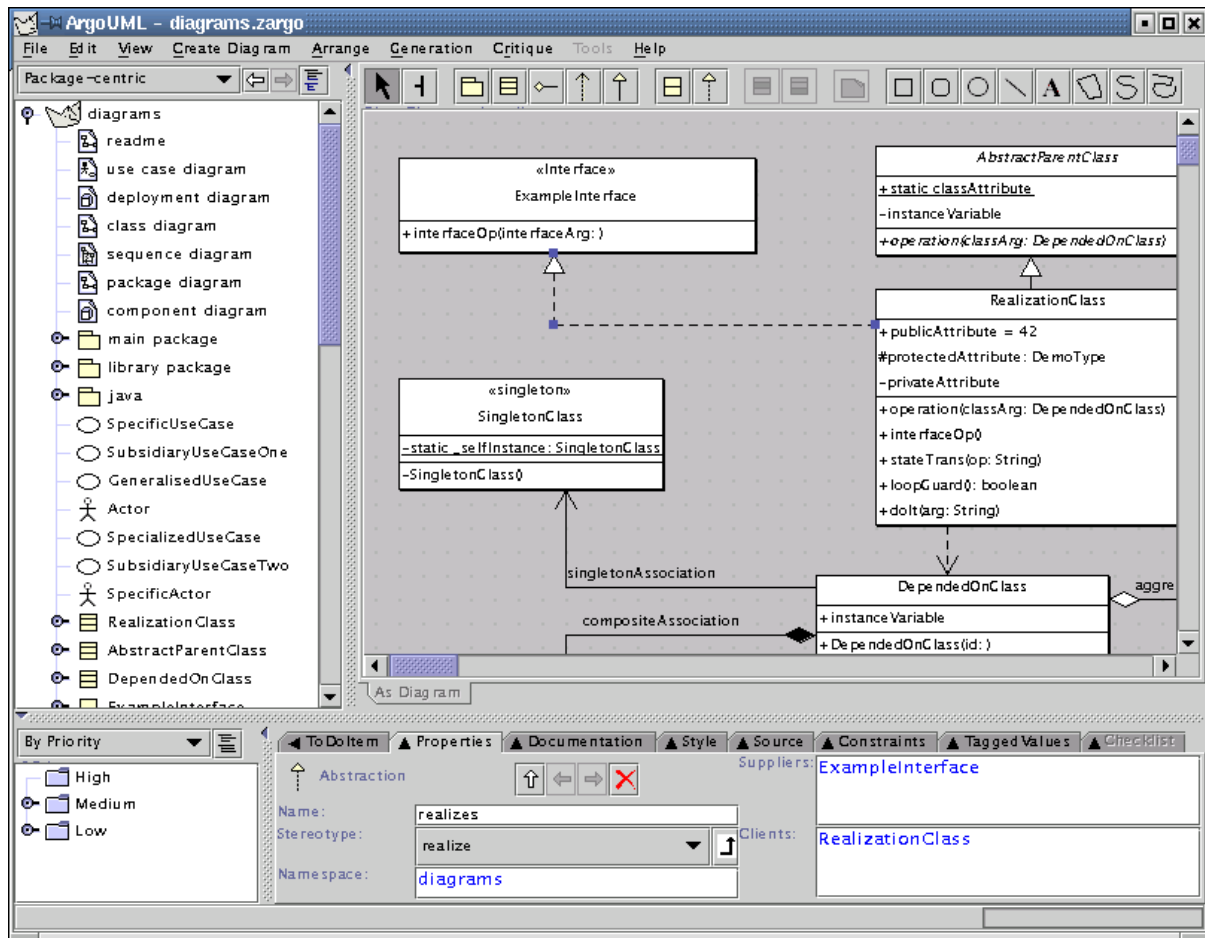
At the top of screen is a *menu barmenu bar* described in Chapter 12. The bulk of the window comprises four sub-windows or *panespanes*. Clockwise from top left these are the *navigator panenavigator pane* (see Chapter 8), *editing paneediting pane* (see Chapter 9), *details panedetails pane* (see Chapter 10) and *to-do paneto-do pane* (see Chapter 11). At the top of the editing pane is a *tool bartool bar*. An overview of the panes is given in Section 7.4. Finally at the bottom of the window is a *status barstatus bar* described in Section 7.5.

7.3. General Mouse Behavior in ArgoUML

Mouse behaviour that is specific to the various panes of ArgoUML (see Section 7.4) or the menu bar, is discussed in the chapters covering those panes and the menu bar. In this section we cover behaviour that is general across all of ArgoUML.

In a number of places in ArgoUML text may be directly edited (for example the constraint editor—see Section 10.8.1). The behaviour of the mouse when handling text is discussed in the sections that follow.

7.3.1. Mouse Button Terminology

ArgoUML assumes a two button mouse. We will refer to the buttons as "button 1" and "button 2". Button 1 is the leftmost button on a right-handed mouse, and sometimes referred to as the *selectselect* button. Button 2 is the rightmost button on a right-handed mouse, and is sometimes referred to as the *adjustadjust* button.

A single depress and release of a mouse button whilst the mouse is still is referred to as a *clickclick*. Two clicks in quick succession is referred to as a *double clickdouble click*. Moving the mouse while holding a button down is referred to as *button motionbutton motion* with the starting point being at *button downbutton down* and the end point at *button upbutton up*.

7.3.2. Button 1 Click

This action is generally used to select an item for subsequent operations. If the item is an artifact in the navigation pane or the editing pane it will be highlighted.

7.3.2.1. General Behaviour When Editing Text

Here button 1 is used to select the point within the text at which operations (text entry and deletion) will take place.

7.3.3. Button 1 Double Click

The behaviour of button 1 double click varies betweens panes and is discussed in their chapters.

7.3.3.1. General Behaviour When Editing Text

Here button 1 double click is used to select a complete word, or other syntactic unit within the text. Subsequent operations (text entry and deletion) will replace the selected text.

7.3.4. Button 1 Motion

7.3.4.1. General Behaviour When Editing Text

Here button 1 motion is used to select a range of text. Subsequent operations (text entry and deletion) will replace the selected text.

7.3.5. Shift and Control with Button 1

7.3.5.1. Within Lists

This behavior applies where there is a list of things that may be selected. This includes various dialog boxes, and the to-do pane, where there is a list of to-do items to be selected.

Where selections are to be made, the SHIFT key is used to with button 1 to *extendextend* from the original button 1 selection to the current position.

Similarly the CONTROL key with button 1 is used to add individual items to the current selection. Where Ctrl-button 1 is used on an item already selected, that item is removed from the selection.

Caution

Users of Microsoft Windows will be familiar with the use SHIFT and CONTROL together to add sub-lists to an existing selection. ArgoUML does not support this. SHIFT-CONTROL will behave as CONTROL.

7.3.5.2. General Behaviour When Editing Text

In a number of places in ArgoUML text may be directly edited (for example the constraint editor—see Section 10.8.1). Here SHIFT button 1 is used to select a range of text from the previously selected point. Subsequent operations (text entry and deletion) will replace the selected text.

7.3.6. Button 2 Actions

Button 2 actions are all dependent on the pane or menu bar, and discussed in their various chapters.

7.3.7. Button 2 Double Click

Button 2 actions are all dependent on the pane or menu bar, and discussed in their various chapters.

7.3.8. Button 2 Motion

Button 2 actions are all dependent on the pane or menu bar, and discussed in their various chapters.

7.4. General Information About Panes

The four sub-windows of the main ArgoUML window are called *panespanes*. Clockwise from top left these are the *navigator panenavigator pane* (see Chapter 8), *editing paneediting pane* (see Chapter 9), *details panedetails pane* (see Chapter 10) and *to-do paneto-do pane* (see Chapter 11). At the top the editing pane is a *tool bartool bar*.

7.4.1. Re-sizing Panes

You can re-size panes by dragging on the divider bars between them. The mouse cursor changes to a vertical or horizontal bar with an arrow at right-angles when over the divider bars.

In addition you will see there are two small left pointing arrows within the vertical divider bars, one at the top of the vertical divider bar between navigation pane and editing pane and one at the top of the vertical divider bar between to-do pane and details pane. Button 1 click on the first of these will expand the editing pane to the full width of the window, button 1 click on the second will expand the details pane to the full width of the window.

There is also a small downward pointing arrow within the horizontal divider bar at its leftmost end. Clicking on this will expand the navigation and editing panes to the full depth of the window.

By using both the top arrow on the vertical divider and the arrow on the horizontal divider, it is possible to expand the editing pane to use the entire window.

7.4.2. Looking at Different Diagrams Simultaneously

ArgoUML provides the ability to spawn copies of key windows. This is possible for the editing pane and for the tabs of the details pane. These can then be manipulated as separate windows and placed as desired on the screen.

This is achieved through use of button 1 double click as described in sections Section 9.2 and Section 10.2.

Note

There is no equivalent function for the navigation or to-do panes.

7.4.3. Reducing the Number of Panes Visible

This function is not implemented directly. However you can spawn a pane (see Section 7.4.2) and maximize this to be the whole window, which achieves the same thing.

7.5. The status bar

The status bar is at the very bottom of the ArgoUML window and is used to display short advisory messages. In general such messages are self explanatory. It is also used for debugging messages for the system in development—these messages can be ignored.

Chapter 8. The Navigation Pane

Table of Contents

8.1. Introduction

Figure 8.1 shows the ArgoUML window, with the navigation pane highlighted.

Figure 8.1. Overview of the navigation pane
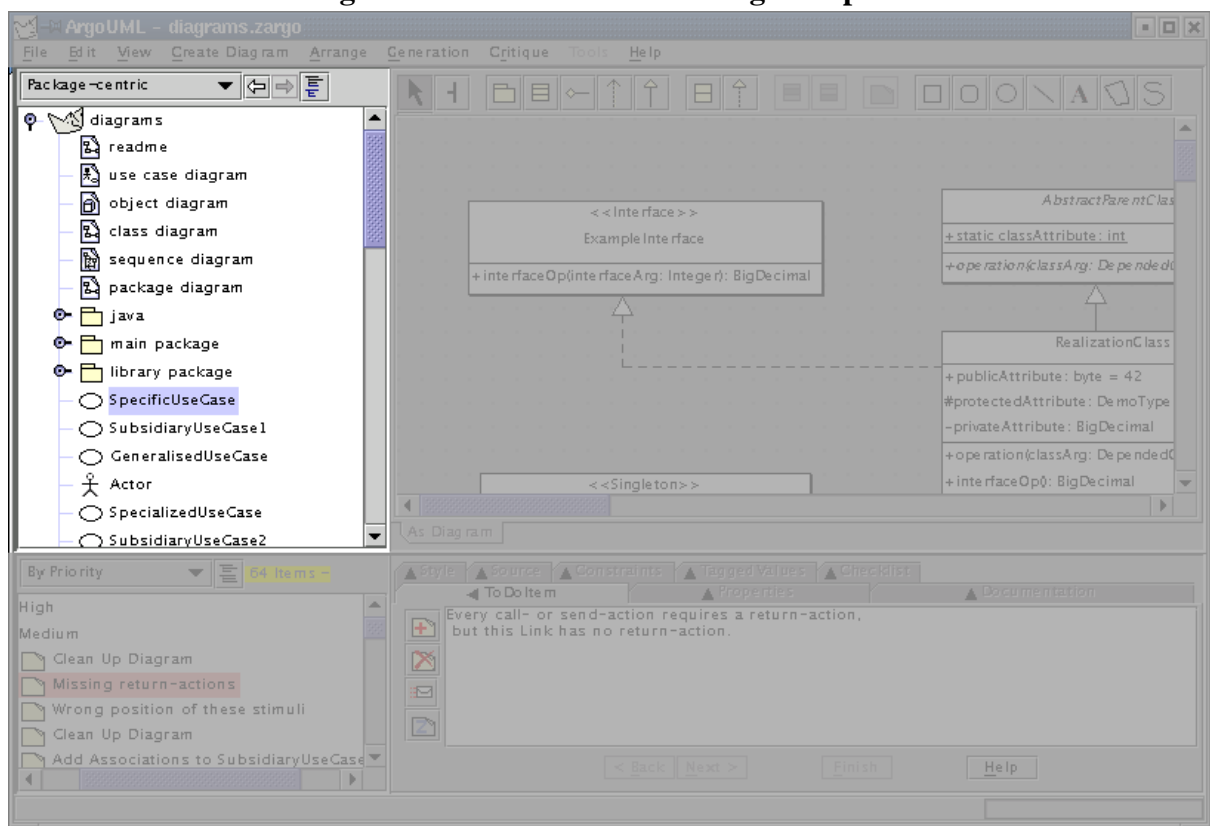
The navigation pane allows the user to view the structure of the model from a number of predefined perspectives. It also allows the user to define their own perspetives for custom navigation of the model.

An important feature, related to the cognitive psychology ideas behind ArgoUML is that not all artifacts are necessarily shown in all perspectives. Rather, the perspectives are used to implement hiding of uninteresting parts of the model.

8.2. Mouse Behavior in the Navigation Pane

Behaviour of the mouse in general, and the naming of the buttons is covered in the chapter on the overall user interface (see Chapter 7).

8.2.1. Button 1 Click

Within the hierarchical display, elements which have sub-hierarchies are indicated by when the hierarchy is hidden and when the hierarchy is open.

Button 1 click over the name of any diagram artifact will cause the diagram to be selected and displayed in the editing pane. Its details will also be displayed in the details pane.

Button 1 click over the name of any artifact other than a diagram in the main area of the navigation pane will cause it to be selected, and its details shown in the details pane. If the artifact is part of a diagram currently displayed in the editing pane, it will be highlighted there.

Note

If the artifact is part of a diagram other than that currently displayed in the Editing Pane, there will be *nono* change of diagram in the Editing Pane.

Where button 2 click has been used to bring up a context sensitive pop-up menu (see below), button 1 click is used to select the menu entry required. button 1 click outside the menu area will remove it.

8.2.2. Button 1 Double Click

This has no effect other than that of button 1 single click.

8.2.3. Button 2 Actions

When used over text items in the the navigation pane, this will display a context dependent pop-up menu. Menu entries are highlighted (but not selected) and sub-menus exposed by subsequent mouse motion (without any buttons). Menu entry selection is with button 1

If the artifact generating the menu has not yet been selected it will also be selected.

8.2.4. Button 2 Double Click

This has no effect other than that of button 2 single click.

8.3. Perspective Selection

The artifacts in the ArgoUML model may be ordered from a number of perspectives. This drop-down allows selection of the perspective to be used in the navigation pane.

Note

Within the hierarchy, there is no particular ordering of artifacts at a particular level. They tend to be in order of creation. This is a known limitation of the current release of ArgoUML.

- `Package-centric` (the default). The navigation hierarchy is organized by package hierarchy. The top level shows the model. Under this are all the top level packages in the model and all the artifacts that are directly in the namespace of the model.

  Beneath each package are all the artifacts that sit within the namespace of that package, including any further sub-packages (which in turn have their own sub-hierarchies).

- `Diagram-centric`. In this view the top level comprises all the diagrams in the model. Beneath each diagram is a flat listing of all the artifacts on the diagram. Artifacts that have sub-artifacts that do not appear on the diagram have their own hierarchy (for example attributes and operations of classes).

- `Inheritance-centric`. In this view the top level shows the model. Beneath this are all artifacts that have no generalization in the model. Those artifacts that have specializations have a sub-hierarchy showing the specializations.

- `Class Associations`. In this view the top level shows the model. Beneath this are all diagrams and all classes. All classes that have associations have a hierarchy tracking through the associated classes.

- `Navigable Associations`. This is identical to the Class Associations ordering, but includes only navigable associations.

- `Association-centric`. In this view the top level shows the model. Beneath this are all diagrams and all associations. All associations have then have a sub-hierarchy listing the classes they connect. Where classes have associated state diagrams these are shown beneath the class.

- `State-centric`. In this view the top level shows all the state machines and all activity graphics associated with classes. The state machine for *ClassClass* is named *ClassClass*`StateMachine` and the activity graph for *ClassClass* is named *ClassClass*`ActivityGraph`.

  Beneath each state machine is a hierarchy showing the state diagram and all of the states. Beneath each state is a list of the transitions in and out of the state.

  Beneath each activity graph is a hierarchy showing the activity diagram and all of the action states. Beneath each action state is a list of the transitions in and out of the action state.

- `Transitions-centric`. This is very similar to `State-centric` view, but under each state machine is listed the diagrams and all transitions on the diagram, with states being shown as sub-hierarchies under their connected transitions.

  Similarly under each activity graph is listed the diagrams and all transitions on the diagram, with action states being shown as sub-hierarchies under their connected transitions.

- `Transitions-paths`. This is again similar to `State-centric` view. However under each state machine is shown the state diagram and the initial state. Under the initial state is a hierarchy showing the states to which it has transitions, and so on for each of those states. A similar structure is used for activity diagrams.

- `Collaboration-centric`. This should show all collaborations in the view with artifacts in hierarchies under their collaborations.

Warning

In the current release of ArgoUML this view is broken, and will display nothing.

- `Dependency-centric`. In this view the top level shows the model. Beneath this are all diagrams and top level artifacts. A hierarchy is shown for any dependency beneath any top level artifact.

Caution

Dependencies include realizations, use case includes and extends relationships (which is not strictly correct under UML 1.3) and implicit dependencies such as inner classes and state machines on their owning classes.

8.4. Navigating Back and Forward

Two buttons are provided allowing the user to navigate back and forward through diagrams in the model.

- ⇐ `Navigate Back`. ArgoUML keeps a record of the diagrams that you have been viewing whilst navigating the model. This button moves you back to the previous one viewed. If there are no more previous diagrams, the button is grayed out.

- ⇒ `Navigate Forward`. ArgoUML keeps a record of the diagrams that you have been viewing whilst navigating the model. This button moves you forward to the next one viewed (after you have used the Navigate Back button to move back). If there are no more next diagrams, the button is grayed out.

Warning

In the current release of ArgoUML there is believed to be a problem, where the same diagram appears several times in a row so that Navigate Back/Forward does not appear to work.

8.5. Configuring Perspectives

The navigation pane is designed to be user configurable, to allow the designer to view in his or her preferred way.

8.5.1. Perspectives Tag

button 1 click on the NavConfig icon ( ) brings up the navigation perspectives dialog (see Figure 8.2).

Figure 8.2. Navigation pane perspectives dialog box
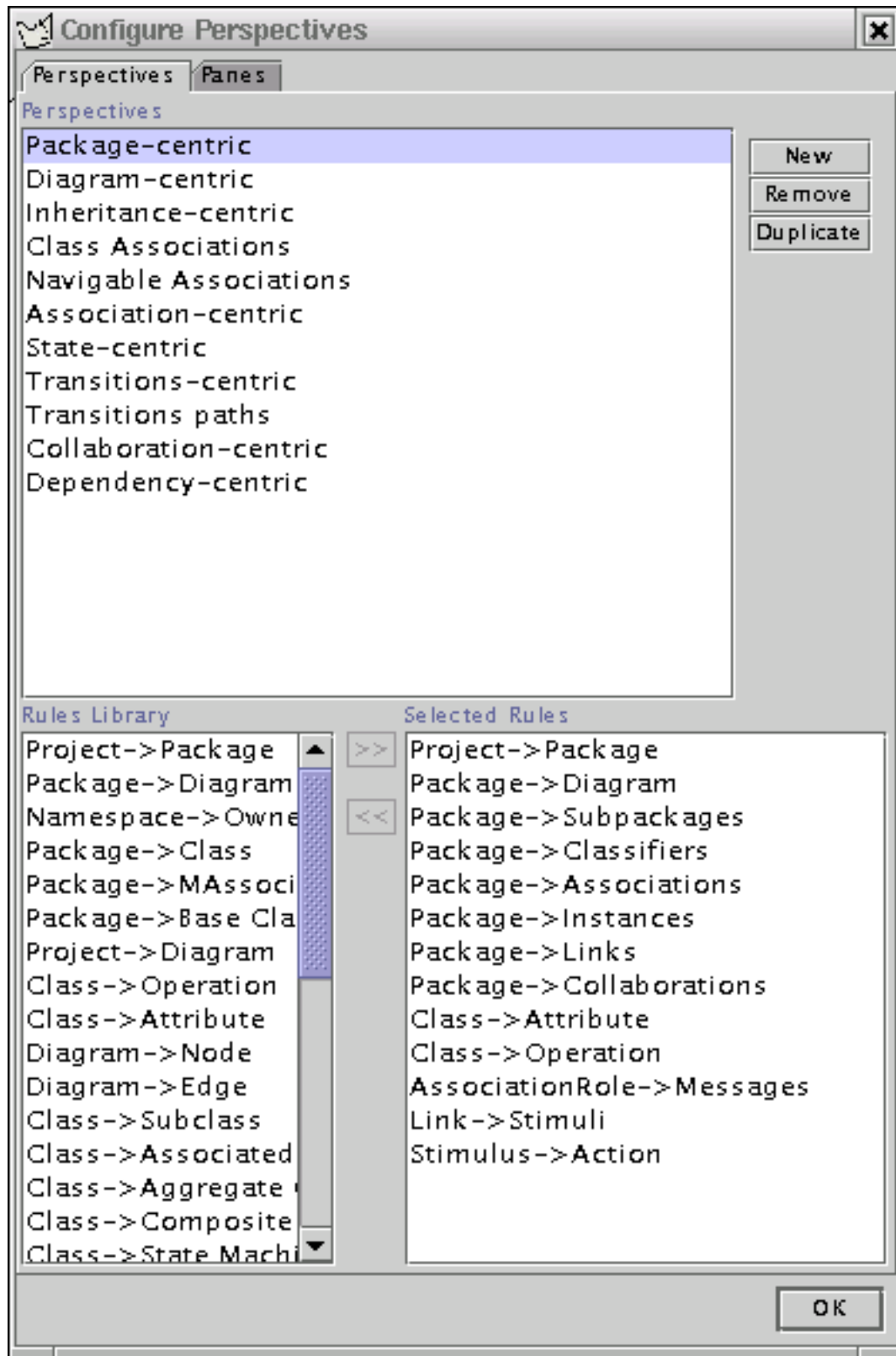
The dialog box has at the top two tabs, labeled `Perspectives` and `Panes`. The first of these is the default, but can be selected explicitly by button 1 click on the tab.

The top half of the dialog contains a text area with a list of all the currently defined perspectives and to the right three buttons stacked vertically. Button 1 click can be used to select a perspective, with CONTROL and SHIFT keys used for multiple perspectives.

- `New`. This creates a new perspective from scratch with no rules selected.

- `Remove`. This removes the selected perspective(s).

- `Duplicate`. This should copy the selected perspective(s) so it/they can be used as the basis of a new perspective.

Warning

In the current release of ArgoUML this function, although not grayed out, has no effect.

The lower half of the dialog contains two text areas. The one on the left, labeled `Rules Library` contains possible expansion rules to be expanded in turn to create the hierarchy. The one on the right contains the actual rules chosen for the hierarchy.

Separating the two areas are buttons labeled `>>` and `<<`. The first of these transfers rules selected on the left to the text area on the right—i.e. it adds rules to the perspective. The second transfers rules selected on the right to the text area on the left—i.e. it removes rules from the perspective.

The rules are applied in turn to the overall project (UML model) to create the top level list. They are then reapplied in turn to each of the elements in the top level list to create the next level hierarchy, and so on until no further expansion is possible.

At the very bottom right is a button labeled `OK` to indicate that all changes are complete. button 1 click on this button will close the dialog window.

Note

There is no `Cancel` button, because all changes take immediate effect.

8.5.2. Panes Tag

This dialog is obtained by using button 1 to select the `Panes` tab after button 1 click on the NavConfig icon ( ) has brought up the navigation perspectives dialog (see Figure 8.3).

Figure 8.3. Navigation panes dialog box
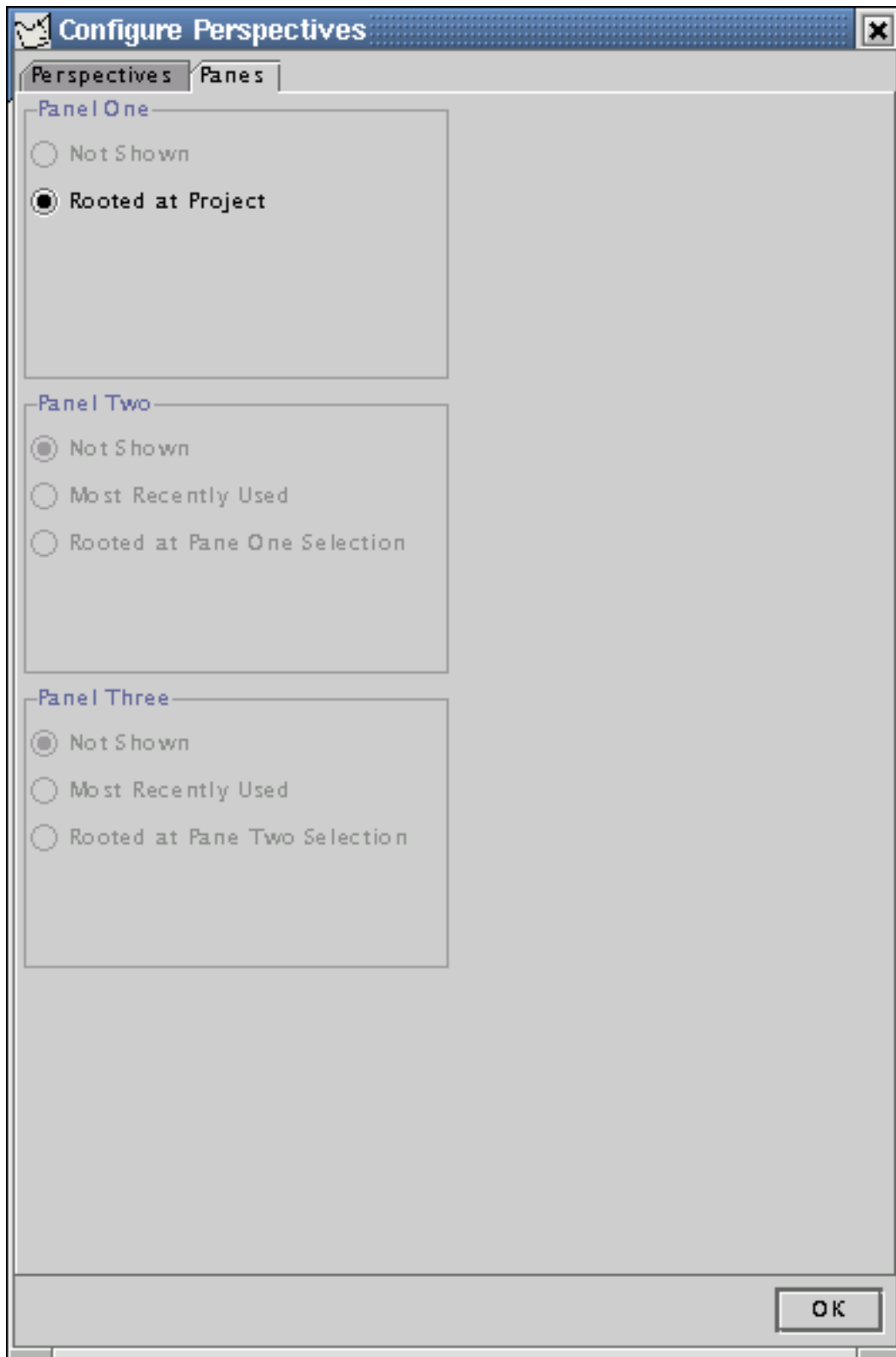
The navigation pane was conceived as being able to offer multiple views simultaneously, with up to three panels of hierarchy. This dialog offers control of these panels, with an option to show or not show each panel, and the option of the second and third panels to be rooted at the selection on the previous panel, or to show the previously used hierarchy.

Caution

This functionality is not yet implemented in ArgoUML. Only the first panel is every shown, which is always rooted at project. All other options are grayed out and unavailable.

When using the navigation pane, it is worth bearing in mind that this is a useful visualization of the model, it is not *solelysolely* for navigation and selection. It is also useful for users to easily (meaning a few mouse clicks without having to arrange a diagram) visualize the model structured according to some perspective. For example, show me the state nesting, or show me the class inheritance tree, or show me the package nesting, or show me the list of actors.

The idea with the second and third nav tree panes, is that sometimes it is easier to explore the tree to a certain level in one tree and then continue expanding in a second tree . Since each tree will not be so deep, it will look more like a list.

For example, look at the way javadocs are viewed with frames: the packages are selected from one list, and interfaces, classes, and exceptions in that package are shown in a second pane. In the case of javadocs, the second pane has a tree with three roots that are always expanded, but it is visually presented as three lists with three headers.

The other suggested use of a second or third nav panel is to list recently visited model elements. This is simliar to the recently visited files listed on the `File` menu of most applications, but for model elements rather than files.

The motivtion is the fact that designers frequently "interrupt" themselves: they are thinking of working on one part of the design, and then they make a decision that needs a change in some other part, and then that has an implication on some other part, and… and… and… the designer forgets what it was they were originally trying to accomplish. They need to "pop their mental stack" to come back to finish what the started. If they fail to do that, they leave half-finished fragments all over the design which always seem to come out as soon as it is presented to someone else.

A recently visited list should be LIFO (stack) ordered, or FIFO (queue) ordered. The goal would be to help designers come back to finish what they were doing, or simply to help them switch between two or three related parts of the design. The `Navigate Back` and `Navigate Forward` buttons in the navigation pane serve the same basic purpose and use a lot less screen space.

ArgoUML's tear-off tabs (see Section 7.4.2 also help with alternating between views, but sometimes users dont want to work with some many separate windows.

This is the theory that motivates the idea of multiple nav panes. For more information see Jason Robbins' PhD dissertation http://argouml.tigris.org/docs/robbins_dissertation/ [http://-/-argouml.tigris.org/-docs/-robbins_dissertation/-]. One practical usability concern is available screen space and another is visual complexity. For this reason the multiple panes have yet to be implemented in ArgoUML while more pressing issues are addressed.

8.6. Context Sensitive Menu

Button 2 Click over any selected artifact in the main area of the navigation pane will cause a pop-up menu to appear.

8.6.1. Properties Pop-Up

This entry on the pop-up menu causes the properties tab of the selected artifact or diagram to be displayed in the details pane.

8.6.2. Add to Diagram

This entry on the pop-up menu appears for any artifact that could be added to a diagram in the editing pane. This could be the diagram currently displayed in editing pane, or it could be a diagram in a "spawned" window (see Section 9.2.2 for an explantion of how to do this).

The item can be placed in a diagram by moving the cursor to the editing pane or a spawned editing pane window (where it will appear as a cross) and clicking button 1.

Caution

This menu entry appears and is not grayed out, even for some artifacts that can not be placed on the currently selected diagram in the editing pane.

This is because there may be separate spawned editing pane windows on which the artifact can validly be placed. ArgoUML cannot track which windows are available, and can only know if an artifact can be validly placed, until a window is selected with the cursor *afterafter* the Add to Diagram is chosen.

The only way to find out if an artifact can be placed in a particular window is to try the operation and see if the artifact is placed.

Caution

A separate known limitation is that the current release of ArgoUML will not let you place more than one copy of any particular artifact on a diagram.

8.6.3. Delete From Model

This entry on the pop-up menu appears for any artifact that could be deleted from the model.

Warning

This deletes the artifact from the model completely, not just from the diagram. To remove just from the diagram use the edit menu (see Section 12.4.7).

Caution

You cannot delete a diagram from the model. This function will just delete all the items on the diagram.

The only way to delete a diagram is to remove it from the saved project file (a project file with the suffix .zargo is just a ZIP archive of all the files that describe the project. There is one .pgml file for each diagram that must be removed from the archive to delete the diagram. Use this technique with caution!

Chapter 9. The Editing Pane

Table of Contents

9.1. Introduction

Figure 9.1 shows the ArgoUML window with the editing pane highlighted.

Figure 9.1. Overview of the editing pane
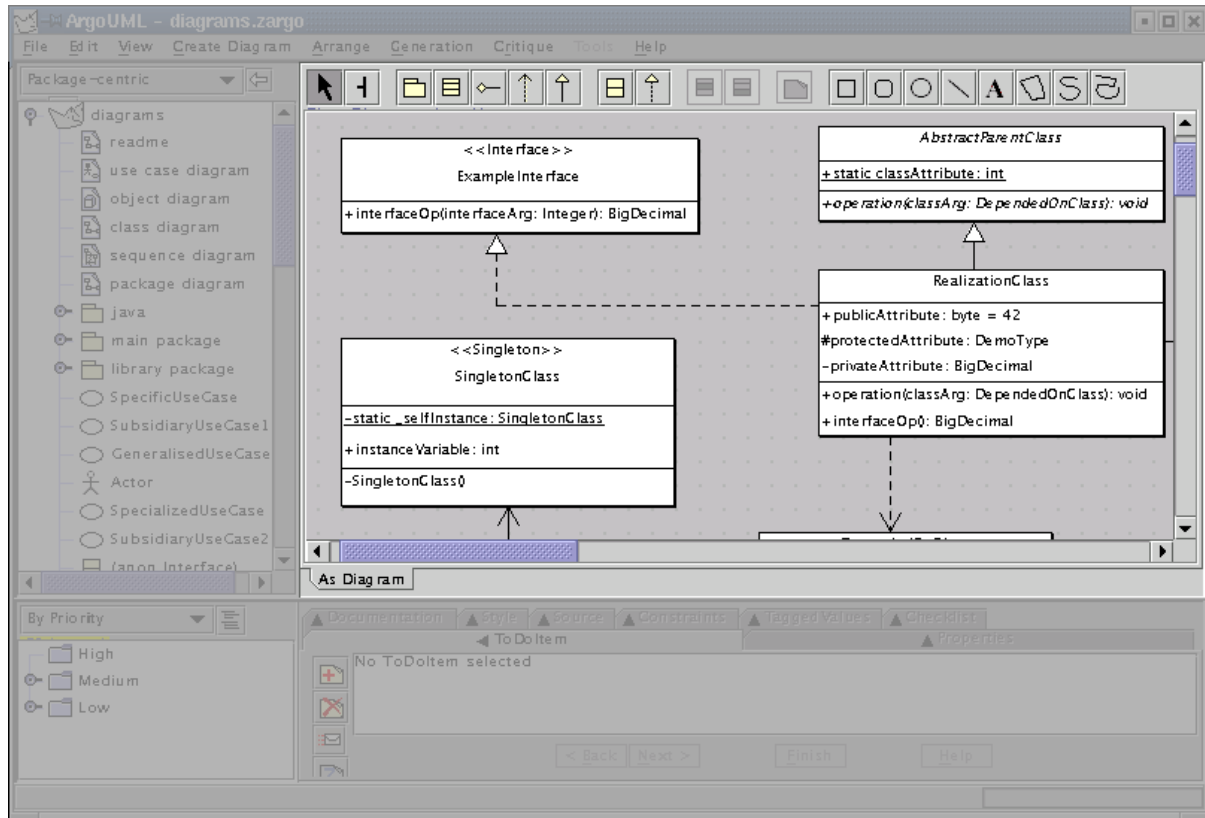
This is where all the diagrams are drawn. In earlier versions of ArgoUML this pane went under a variety of names. You may encounter "drawing pane", "diagram pane" or "multi-editor pane" in other documentation that is still being updated.

The pane has a tool bar at the top, and a single tab labeled As Diagram at the bottom. So long as your window is big enough, you will see the name of the diagram in the top right corner at the end of the tool bar.

9.2. Mouse Behavior in the Editing Pane

Behaviour of the mouse in general, and the naming of the buttons is covered in the chapter on the overall user interface (see Chapter 7).

9.2.1. Button 1 Click

In the tool bar of the editing pane, button 1 click is used to select the next tool for adding artifacts to the diagram (but only once—see double clicking for adding multiple artifacts). Adding to the diagram is achieved by moving the mouse to the editing area and clicking again.

In the main editing area button 1 click is used to select an individual artifact.

Where button 2 click has been used to bring up a context sensitive pop-up menu (see below), button 1 click is used to select the menu entry required. button 1 click outside the menu area will remove it.

There are various more detailed effects, which are discussed under the descriptions of the various tools (see Section 9.3).

9.2.2. Button 1 Double Click

When used on the tool bar with a tool to add an artifact, the selected artifact will be added multiple times to the

drawing area, once for each further button click, until the tool is again selected or another tool chosen.

When used within the drawing area on an artifact that has sub-components, double click will select the sub-component (creating it if necessary).

For example double clicking over an operation compartment of a class will select the operation. Or create one if there is not one there.

A special use is with package artifacts on the class diagram. A double click on a package will take you to the class diagram associated with a package (the first created if there is more than one), or will offer to create one for you if there is none.

You can "spawn" a copy of the editing pane by double-clicking on the `As Diagram` tab at the bottom of the pane. It will then show as an individual window and you can put it where you will on the screen.

9.2.3. Button 1 Motion

When adding an artifact using the toolbar, button 1 motion can generally be used to define the size of the new artifact. Its dimensions (line ends, diagonal corners of a rectangular artifact etc) are defined by button 1 down and button 1 up.

Where the artifact being added is some form of connector its termination point is shown with button 1 up over the terminating artifact. button 1 click may be used in the space between artifacts to create articulation points in the connector. This is particularly useful where connectors must loopback on themselves.

Over graphical artifacts button 1 motion will move the artifact to a new position. Graphical artifacts that are selected show handles, and these can be used for re-sizing.

Some artifacts (e.g. actor, class) show special handles at the sides, top and bottom, which can be dragged to form types of relationship with other artifacts.

Where the artifact is some form of connector between other items, button 1 motion other than at a handle will cause a new handle to be created, allowing the connector to be articulated at that point. Such new handles can be removed by moving them to the end of the connector.

There are various more detailed effects, which are discussed under the descriptions of the various tools (see Section 9.3).

9.2.4. Shift and Control with Button 1

Where selections are to be made, the SHIFT key is used to with button 1 to *addadd* unselected artifacts to the current selection. Where an artifact is already selected, it is removed from the current selection.

Using the CONTROL key invokes the broom tool, causing the selected artifacts (and any others swept up with them) to be moved with the broom tool (see Section 9.3.1).

9.2.5. Button 2 Actions

When used over artifacts in the the editing pane, this will display a context dependent pop-up menu. Menu entries are highlighted (but not selected) and sub-menus exposed by subsequent mouse motion (without any buttons). Menu entry selection is with button 1

If the artifact generating the menu has not yet been selected it will also be selected.

9.2.6. Button 2 Double Click

This has no effect other than that of button 2 single click.

9.2.7. Button 2 Motion

This is used to select items in a context sensitive menu popped up by use of button 2 click.

9.3. The tool bar

The toolbar at the top of the editing pane provides the main functions of the pane. The default tool is the `Select` tool (▶ . In general button 1 on any tool click selects a tool for one use, before reverting to the default tool, button 1 double click selects a tool for repeat use until another tool is selected.

The tools fall into four categories.

- *Layout tools.Layout tools.* Provide assistance in laying out artifacts on the diagram.

- *Annotation tools.Annotation tools.* Used to annotate artifacts on the diagram.

- *Drawing tools.Drawing tools.* Used to add general graphics artifacts to diagrams.

- *Diagram specific tools.Diagram specific tools.* Used to add UML artifacts specific to a particular diagram type to the diagram.

9.3.1. Layout Tools

Two tools are provided in all diagrams in this category.

- ▶ `Select`. This tool provides for general selection of artifacts on the diagram. Button 1 click will select an artifact. SHIFT with button 1 can be used to select (or deselect multiple artifacts. Button 1 motion will move selected 2D items or add and move a new control point on a link. Button 1 motion on a selected component's control point will stretch that component's shape.

- ⊣ *BroomBroom.* Button 1 motion with this tool provide a "broom" which will sweep all artifacts along. This is very shortcut way of lining things up.

  Broom can also be invoked by using CONTROL with button 1 motion when the `Select` tool is in use.

  ### Tip

  Additional control of artifact layout is provided through the `Arrange` menu (see Section 12.7).

9.3.2. Annotation Tools

Only one annotation tool is provide, `Note` (▱). This is used to add a note to a selected UML artifact.

Caution

Unlike most other tools you use the `Select` tool to select an artifact, and then button 1 click on `Note` to create the note.

The note is created alongside the selected artifact, with the default contents `New note`. This text can be selected with button 1 and edited from the keyboard.

Warning

The UML standard allows notes to be attached to any artifact. However in the current version of Ar-goUML there is only a partial implementation. `Note` is only provided on class and state diagrams, and can only be attached to classes, interfaces, states and pseudostates.

Tip

You can add a note also to an object in a deployment diagram by using its context sensitive `Add` pop-up menu (see Section 9.7).

Tip

If you need to add a note to some other artifact, try using a text box from the graphics drawing tools (see Section 9.3.3).

9.3.3. Drawing Tools

These are a series of tools for providing graphical additions to diagrams. Although they are not UML artifacts, the UML standard provides for such decoration to improve the readability of diagrams.

Tip

These drawing tools provide a useful way to partially support some of the UML features (such as general purpose notes) that are missing from the current release of ArgoUML.

Eight tools are provided. button 1 click on the diagram will place an instance of the graphical item of the same size as the last one placed. The size can be controlled by button 1 motion during placement. One side or end of the element will be at button 1 down, the other side or end at button 1 up. In general after they are placed on the diagram, graphical elements can be moved with the `Select` tool and button 1 and re-sized by button 1 motion on the handles after they have been selected.

- `Rectangle`. Provides a rectangle.

- `Rounded Rectangle`. Provides a rectangle with rounded corners. There is no control over the degree of rounding.

- `Circle`. Provides a circle.

- `Line`. Provides a line.

- `Text`. Provides a text box. The text is entered by selecting the box and typing. Text is in blue and centered and after typing, the box will shrink to the size of the text. However it can be re-sized by dragging on the corners.

- `Polygon`. Provides a polygon. The points of the polygon are selected by button 1 click and the polygon closed with button 1 double click (which will link the final point to the first point).

- `Spline`. Provide an open spline. The control points of the spline are selected with button 1 and the last point selected with button 1 double click.

- `Ink`. Provide a polyline. The points are provided by button 1 motion.

9.3.4. Use Case Diagram Specific Tools

Seven tools are provided specific to UML artifacts on use case diagrams. The detailed properties of these artifacts are described in the section on use case diagram artifacts (see Chapter 15).

- Actor. Add an actor to the diagram. For convenience, when the mouse is over a selected actor it displays two handles to left and right which may be dragged to form association relationships.

- Use Case. Add a use case to the diagram. For convenience, when the mouse is over a selected use case it displays two handles to left and right which may be dragged to form association relationships and two handles top and bottom which may be dragged to form generalization and specialization relationships respectively.

- Association. Add an association between two artifacts selected using button 1 motion (for the first artifact to the second).

- Generalization. Add a generalization between two artifacts selected using button 1 motion (from the child to the parent).

- Extend. Add an extend relationship between two artifacts selected using button 1 motion (from the extended to the extending use case).

- Include. Add an include relationship between two artifacts selected using button 1 motion (from the including to the included use case).

- Dependency. Add a dependency between two artifacts selected using button 1 motion (from the dependent artifact).

Caution

ArgoUML incorrectly uses stereotyped dependencies to represent UML extends and includes relationships.

9.3.5. Class Diagram Specific Tools

Nine tools are provided specific to UML artifacts on class diagrams. The detailed properties of these artifacts are described in the section on class diagram artifacts (see Chapter 16).

- Package. Add a package to the diagram.

- Class. Add a class to the diagram. For convenience, when the mouse is over a selected class it displays two handles to left and right which may be dragged to form association relationships and two handles top and bottom which may be dragged to form generalization and specialization relationships respectively.

- Association. Add an association between two artifacts selected using button 1 motion (from the first artifact to the second artifact).

- Dependency. Add a dependency between two artifacts selected using button 1 motion (from the de-

pendent artifact).

- ⬆ Generalization. Add a generalization between two artifacts selected using button 1 (from the child to the parent).

- ▤ Interface. Add an interface to the diagram. For convenience, when the mouse is over a selected interface it displays a handle at the bottom which may be dragged to form a realization relationship (the target being the realizing class).

- ⬆ Realization. Add a realization between a class and an interface selected using button 1 motion (from the realizing class to the realized interface).

- ▤ Add Attribute. Add an attribute to the currently selected class. The attribute is given the default name `newAttr` of type `int` and may be edited by button 1 double click and using the keyboard, or by selecting with button 1 click (after the class has been selected) and using the property tab.

Note

This tool is grayed out except when a class or interface is selected.

Caution

In the current release of ArgoUML this tool is not grayed out for intefaces as it should be (attributes are not permitted for interfaces in the UML 1.3 standard). However the tool has no effect if selected for an interface.

- ▤ Add Operation. Add an operation to the currently selected class or interface. The operation is given the default name `newOperation` with no arguments and return type `void` and may be edited by button 1 double click and using the keyboard, or by selecting with button 1 click (after the class has been selected) and using the property tab.

Note

This tool is grayed out except when a class or interface is selected.

9.3.6. Sequence Diagram Specific Tools

Six tools are provided specific to UML artifacts on sequence diagrams. The detailed properties of these artifacts are described in the section on sequence diagram artifacts (see Chapter 17).

- ▭ Object. Add an object to the diagram.

- → Stimulus Call. Add a call stimulus between two objects selected using button 1 motion (from the originating object to the receiving object).

- →▭ Stimulus Create. Add a create stimulus between two objects selected using button 1 motion (from the originating object to the receiving object).

- →✕ Stimulus Destroy. Add a destroy stimulus between two objects selected using button 1 motion (from the originating object to the receiving object).

-

Stimulus Send. Add a send stimulus between two objects selected using button 1 motion (from the originating object to the receiving object).

- Stimulus Return. Add a return stimulus between two objects selected using button 1 motion (from the originating object to the receiving object).

9.3.7. State Diagram Specific Tools

Ten tools are provided specific to UML artifacts on state diagrams. The detailed properties of these artifacts are described in the section on state diagram artifacts (see Chapter 18).

- State. Add a state to the diagram.

- Composite State. Add a composite state to the diagram. All artifacts that are subsequently placed on the diagram on top of the composite state will form part of that composite state.

- Transition. Add a transition between two states selected using button 1 motion (from the originating state to the receiving state).

- Initial. Add an initial pseudostate to the diagram.

Caution

There is nothing to stop you adding more than one initial state to a diagram or composite state. However to do so is meaningless, and one of the critics will complain.

- Final State. Add a final state to the diagram.

- Branch. Add a branch pseudostate to the diagram.

Caution

A well formed branch should have one incoming transition and two or more outgoing. ArgoUML does not enforce this, but an ArgoUML critic will complain about any branch that does not follow this rule.

- Fork. Add a fork pseudostate to the diagram.

Caution

A well formed fork should have one incoming transition and two or more outgoing. ArgoUML does not enforce this, but an ArgoUML critic will complain about any fork that does not follow this rule.

- Join. Add a join pseudostate to the diagram.

Caution

A well formed join should have one outgoing transition and two or more incoming. ArgoUML does not enforce this, but an ArgoUML critic will complain about any join that does not follow this rule.

- Shallow History. Add a shallow history pseudostate to the diagram.

- Deep History. Add a deep history pseudostate to the diagram.

9.3.8. Collaboration Diagram Specific Tools

Three tools are provided specific to UML artifacts on collaboration diagrams. The detailed properties of these artifacts are described in the section on collaboration diagram artifacts (see Chapter 19).

- Classifier Role. Add a classifier role to the diagram.

- Association Role. Add an association role between two classifier roles selected using button 1 motion (from the originating classifier role to the receiving classifier role).

- Add Message. Add a message to the selected association role.

Note

This tool is grayed out except when an association role is selected.

9.3.9. Activity Diagram Specific Tools

Seven tools are provided specific to UML artifacts on activity diagrams. The detailed properties of these artifacts are described in the section on activity diagram artifacts (see Chapter 20).

- Action State. Add an action state to the diagram.

- Transition. Add a transition between two action states selected using button 1 motion (from the originating action state to the receiving action state).

- Initial. Add an initial pseudostate to the diagram.

Caution

There is nothing to stop you adding more than one initial state to a diagram. However to do so is meaningless, and one of the critics will complain.

- Final State. Add a final state to the diagram.

- Branch. Add a branch pseudostate to the diagram.

Caution

A well formed branch should have one incoming transition and two or more outgoing. ArgoUML does not enforce this, but an ArgoUML critic will complain about any branch that does not follow this rule.

- Fork. Add a fork pseudostate to the diagram.

Caution

A well formed fork should have one incoming transition and two or more outgoing. ArgoUML does

not enforce this, but an ArgoUML critic will complain about any fork that does not follow this rule.

- ⊥⊥ Join. Add a join pseudostate to the diagram.

Caution

A well formed join should have one outgoing transition and two or more incoming. ArgoUML does not enforce this, but an ArgoUML critic will complain about any join that does not follow this rule.

9.3.10. Deployment Diagram Specific Tools

Ten tools are provided specific to UML artifacts on deployment diagrams. The detailed properties of these artifacts are described in the section on deployment diagram artifacts (see Chapter 21).

Note

Remember that ArgoUML's deployment diagrams are also used for component diagrams.

- Node. Add a node to the diagram. For convenience, when the mouse is over a selected node it displays four handles to left, right, top and bottom which may be dragged to form association relationships.

- Node Instance. Add a node instance to the diagram. For convenience, when the mouse is over a selected node instance it displays four handles to left, right, top and bottom which may be dragged to form link relationships.

- Component. Add a component to the diagram. For convenience, when the mouse is over a selected component it displays four handles to left, right, top and bottom which may be dragged to form dependency relationships.

- Component Instance. Add a component instance to the diagram. For convenience, when the mouse is over a selected component instance it displays four handles to left, right, top and bottom which may be dragged to form dependency relationships.

- Dependency. Add a dependency between two artifacts selected using button 1 motion (from the dependent artifact).

- Class. Add a class to the diagram. For convenience, when the mouse is over a selected class it displays two handles to left and right which may be dragged to form association relationships.

Note

Unlike a class diagram, two handles top and bottom which may be dragged to form generalization and specialization relationships are not shown. Such relationships may not be described on a deployment or component diagram.

- Interface. Add an interface to the diagram.

Caution

In the current release of ArgoUML, when the mouse is over a selected interface it displays a handle at the bottom. This suggests this may be dragged to form a realization relationship as in a class diagram. Such relationships may not be displayed on a deployment or component diagram, and this handle has

no effect.

- ◇— Association. Add an association between two artifacts (node, component, class or interface) selected using button 1 motion (from the first artifact to the second artifact).

  ### Caution

  The constraint that associations between classes and interfaces must not be navigable *fromfrom* the interface still applies on deployment diagrams.

- ☐ Class. Add an object to the diagram. For convenience, when the mouse is over a selected class it displays four handles to left, right, top and bottom, which may be dragged to form link relationships.

- ↘ Link. Add a link between two artifacts (node instance, component instance or object) selected using button 1 motion.

9.4. Clarifiers

A key feature of ArgoUML are the critics, which run in parallel with the main ArgoUML tool. When they find a problem, they typically raise a to-do item, and also highlight the problem on the editing pane. The graphical techniques used for highlighting are called *ClarifiersClarifiers*

- Note icon (▱). Displayed at the top left of an artifact indicates a critic of that artifact. Moving the mouse over the icon will pop up the critic headline.

- Colored wavy line (〰〰〰〰). Used for critics specific to sub-components of graphical artifacts. For example to underline attributes with a problem within a class.

- Solid colored line (▬▬▬▬). Not seen in ordinary editing, but used when a to-do item is highlighted from the to-do pane (see Chapter 11) by button 1 double click. The solid line is used to show all the artifacts affected by the critic, for example all stimuli that are out of order.

9.5. The Drawing Grid

The editing pane is provided with a background grid which can be set in various styles or turned off altogether through the Edit menu (see Section 12.5.6).

Whatever grid is actually displayed, placement of items on the diagram is always controlled by the setting for grid snap, which ranges from 4 to 32 pixels (see Section 12.5.7).

9.6. The Diagram Tab

At the bottom of the editing pane is a small tab labeled as As Diagram. The concept is that a UML diagram can be displayed in a number of ways, for example as a graphical diagram or as a table. Each representation would have its own tab and be selected by button 1 click on the tab.

Earlier versions of ArgoUML did implement a tabular representation, but the current release only supports a diagram representation, so this tab is of null function.

In addition you can use button 1 double click on this tab to spawn a copy of the current diagram (see Section 7.4.2 for more details).

9.7. Pop-Up Menus

Within the editing pane, button 2 click over an artifact will bring up a pop-up menu with up to six main entries, each with a sub-menu.

9.7.1. Critiques

This sub-menu gives list of all the critics that have triggered for this artifact. Selection of a menu entry causes that entry to be highlighted in the to-do pane and its detailed explanation to be placed in the ToDoItem tab of the details pane.

9.7.2. Ordering

This menu controls the ordering of overlapping artifacts on the diagram. It is equivalent to the Reorder sub-menu of the Arrange menu (see Section 12.7.3). There are four entries.

- Forward. The selected artifacts are moved one step forward in the ordering hierarchy with respect to other artifacts they overlap.

- Backward. The selected artifacts are moved one step back in the ordering hierarchy with respect to other artifacts they overlap.

- To Front. The selected artifacts are moved to the front of any other artifacts they overlap.

- To Back. The selected artifacts are moved to the back of any other artifacts they overlap.

9.7.3. Add

This sub-menu only appears for artifacts that can have notes attached (class, interface, object, state, pseudostate) or have operations or attributes added (class, interface). There are at most three entries.

- Add Attribute. Only appears where the selected artifact is a class or interface. Adds an attribute to the artifact

- Add Operation. Only appears where the selected artifact is a class or interface. Adds an operation to the artifact

- Note. Attaches a note to the selected artifact.

9.7.4. Show

This sub-menu only appears with class or object artifacts. It is used to control whether the attribute and operation compartments of the artifact are displayed. There are six possible entries, but at most three show at any one time.

- Hide All Compartments. Only appears when both attribute and operation compartments are displayed. Hides both compartments.

- `Show All Compartments`. Only appears when both attribute and operation compartments are hidden. Displays both compartments.

- `Hide Attribute Compartment`. Only appears when the attribute compartment is displayed. Hides the compartment.

- `Show Attribute Compartment`. Only appears when the attribute compartment is hidden. Displays the compartment.

- `Hide Operation Compartment`. Only appears when the operation compartment is displayed. Hides the compartment.

- `Show Operation Compartment`. Only appears when the operation compartment is hidden. Displays the compartment.

9.7.5. Modifiers

This sub-menu only appears with class artifacts. It is used to set or clear the values of the various modifiers available.

- `Public`. Set if this class is visible outside its package.

- `Abstract`. Set for an abstract class.

- `Final`. Set for a final class, i.e. one with no subclasses.

- `Root`. Set for a root class, i.e. one with no superclasses.

- `Active`. Set for a class with dynamic behavior.

Note

This really ought to be set automatically for classes with state machines or activity diagrams.

9.7.6. Multiplicity

This sub-menu only appears with association artifacts. It is used to control the multiplicity at the end of the association nearest the mouse click point. There are only four entries, a sub-set of the range of multiplicities that are available through the property sheet of a association end (see Section 15.5).

- `1`

- `0..1`

- `1..*`

- `0..*`

9.7.7. Aggregation

This sub-menu only appears with association artifacts. It is used to control the aggregation at the end of the association nearest the mouse click point. There are three entries.

- `none`. Remove any aggregation.

- `aggregate`. Make this end a shared aggregation (loosely known as an "aggregation").

- `composite`. Make this end a composite aggregation (loosely known as a "composition").

Warning

UML requires that at most one end of an association has any aggregation relationship. The current release of ArgoUML has a known problem where it believes any end whose aggregation has not been explicity set is in some form of aggregation (although this is not shown on the diagram or property sheet). This will trigger a critic. The solution is to change the aggregation to one of the aggregate forms and then back to none to explicitly set no aggregation.

Caution

UML requires that an end with a composition relationship must have a multiplicity of 1 (the default).

9.7.8. Navigability

This sub-menu only appears with association artifacts. It is used to control the navigability of the association. There are three entries.

- `bidirectional`. Make the association navigable in both directions.

- `<class1> to <class2>`. Make the association navigable only from <class1> to <class2>. In other words <class1> can reference <class2> but not the other way round.

- `<class2> to <class1>`. Make the association navigable only from <class2> to <class1>. In other words <class2> can reference <class1> but not the other way round.

Note

UML does permit an association to be non-navigable in both directions. If you want this, you will have to set each of the association ends navigation property, reached from the property tab of the association.

This is considered bad design practice (it will trigger a critic in ArgoUML), so is only of theorertical interest.

Note

UML does not permit navigability from an interface to a class. This drop down is disabled for associations between interfaces and classes.

9.7.9. Properties

This entry has no sub-menu. Selecting the entry will cause the property tab for the selected artifact to be displayed in the details pane.

Chapter 10. The Details Pane

Table of Contents

10.1. Introduction

Figure 10.1 shows the ArgoUML window, with the details pane highlighted.

Figure 10.1. Overview of the details pane
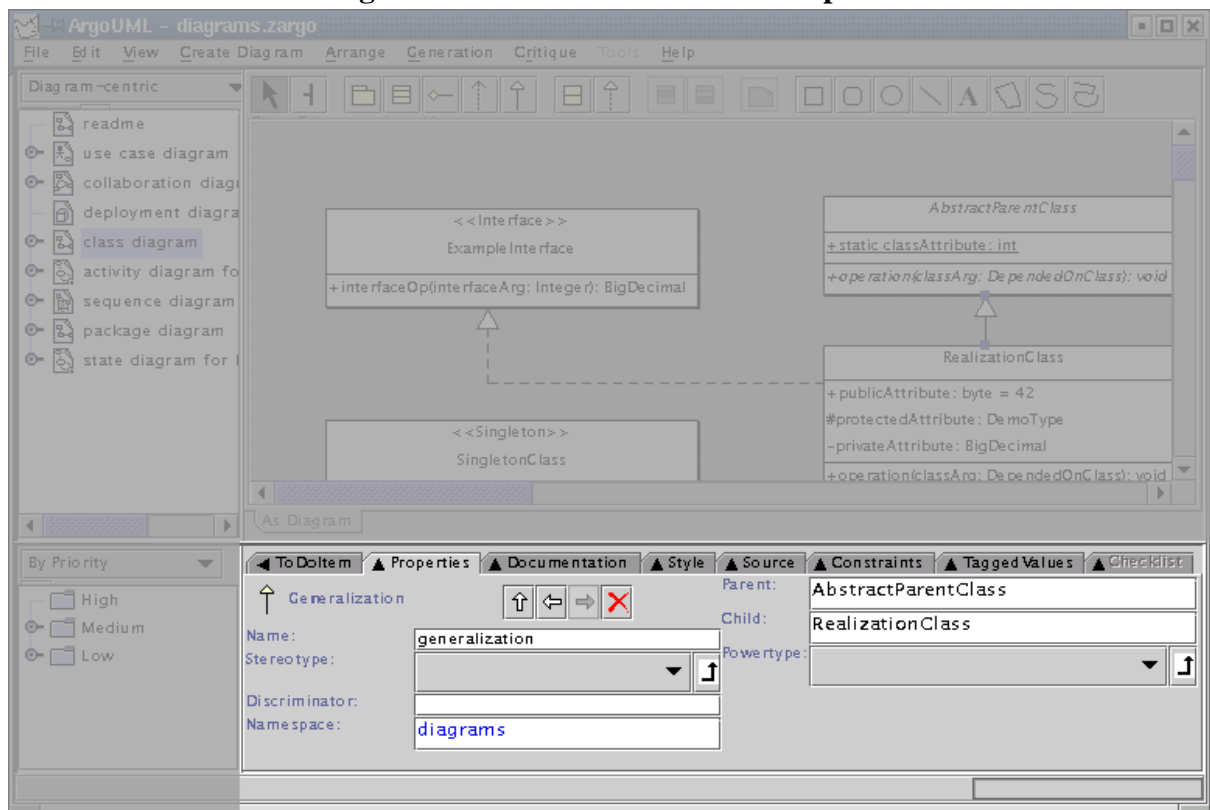
For any artifact within the system, this pane is where all its associated data is viewed and entered.

The Pane has a series of tabs at the top, which are selected by button 1 click, through the `View > Details Tabs` menu or through the menu shortcuts. The body of a tab is a menu of items to be checked, selected or entered specific to the particular tab selected.

Of these, the `Properties` Tab is by far the most complex, with a different presentation for each artifact within the system. The detailed descriptions of the properties tab for each artifact are the subject of separate chapters covering the artifacts that may appear on the various diagrams (see Chapter 14 through Chapter 21).

10.1.1. Spawning Details Tabs

Tabs in the details pane may be spawned as separate stand alone windows, by button 1 double click on the tab. See Section 7.4.2 for more information.

10.2. Mouse Behavior in the Details Pane

Behaviour of the mouse in general, and the naming of the buttons is covered in the chapter on the overall user interface (see Chapter 7).

10.2.1. Button 1 Click

This action is generally used to select an item for subsequent operations. For fields of the property tab that are references to other artifacts this will navigate to that artifact.

Where button 2 click has been used to bring up a context sensitive pop-up menu (see below), button 1 click is used to select the menu entry required. button 1 click outside the menu area will remove it.

10.2.2. Button 1 Double Click

In general this has no effect other than that of button 1 single click.

However you can "spawn" a copy of any tab by double-clicking on the tab. It will then show as an individual window and you can put it where you will on the screen.

10.2.3. Button 2 Actions

When used over text field entries in the the properties tab, this may display a context dependent pop-up menu (dependin on the field). Menu entries are highlighted (but not selected) and sub-menus exposed by subsequent mouse motion (without any buttons). Menu entry selection is with button 1

If the entry generating the menu has not yet been highlighted it will also be highlighted.

10.2.4. Button 2 Double Click

This has no effect other than that of button 2 single click.

10.2.5. Button 2 Motion

This is used to select items in a context sensitive menu popped up by use of button 2 click.

10.3. To Do Item Tab

This tab provides control over the various to-do items raised by the ArgoUML critics (discussed in more detail in the section on the `Critique` menu—see Section 12.9). Figure 10.2 shows a typical pane. The to-do item is selected with button 1 in the to-do pane (see Chapter 11) or by using the `Critiques` context sensitive pop-up menu on the editing pane.

Figure 10.2. Example of the To Do Item tab on the properties pane
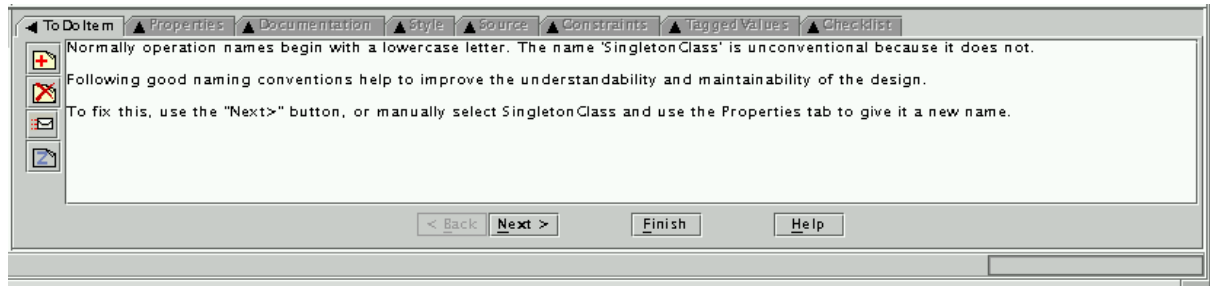
The body of the tab describes the problem found by the critic and outlines how it can be fixed. To the left are four buttons.

- `New To Do Item`… This launches a dialog box (see Figure 10.3), which allows you to create your own to-do item, with its own headline (which appears in the to-do pane), priority for the to-do pane, reference URL and detailed description for further information.

Figure 10.3. Dialog box for New To Do Item

85
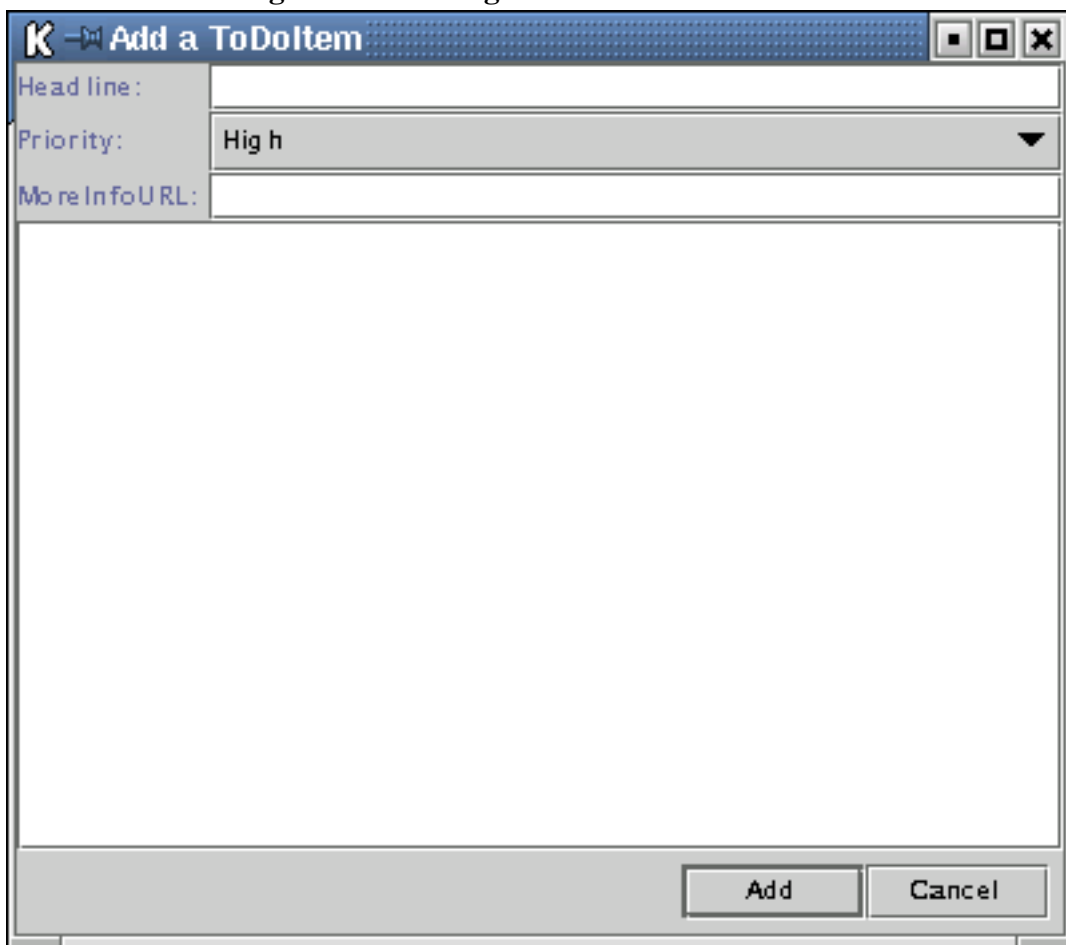
Warning

In the current release of ArgoUML these to-do notes are not saved, and so will be lost. They should not be used at this stage.

• ![icon] `Resolve Item`... This pops up a dialog allowing the user to resolve the selected to-do item (see Figure 10.4). This is an important dialog, because it allows you to deal with to-do items in ways other than the recommendation of the to-do item (which is the whole point of their being advisory).

Figure 10.4. Dialog box for New To Do Item
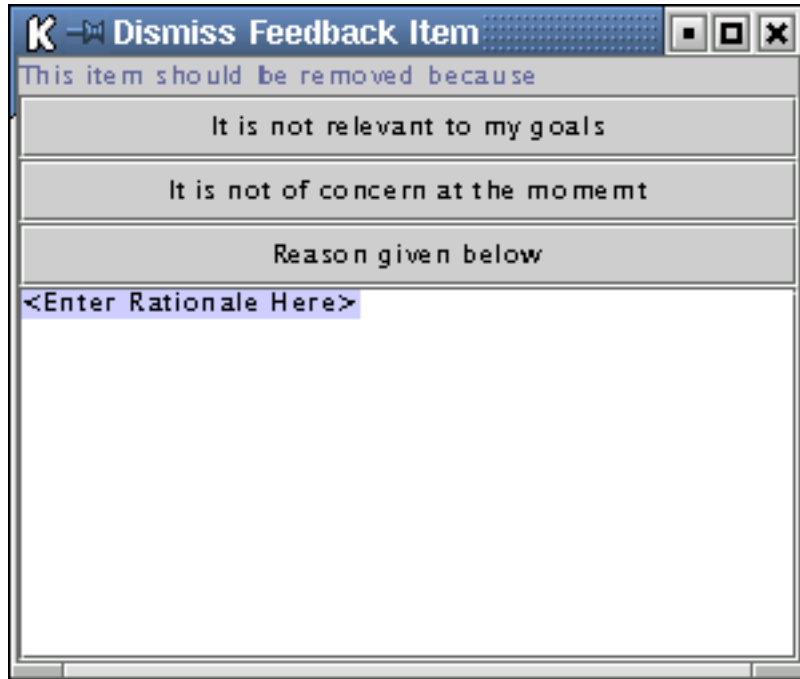
At the top are three buttons, which will dismiss the selected item, labeled `It is not relevant to my goals`, `It is not of concern at the moment` and `Reason given below`. If you choose the third of these you should *firstfirst* enter a reason in the main text box.

Tip

If you wish to resolve a to-do item by following its recommendations, just make the recommended changes and the to-do item will disappear of its own accord.There is no need to use this dialog.

Caution

Because of the layout of the dialog box in the current release of ArgoUML, it is all to possible to click the `Reason given below` button before entering a reason.

• ![icon] `Send Email To Expert`... This pops up a dialog allowing the user to send Email to an expert asking for advice. The dialog is specific to your default Email program, but will be a dialog for a new message to `jrobbins@ics.uci.edu` and the headline of the selected to-do item as subject.

Warning

This will send an Email to Jason Robbins' old Email address at UCI, and may not receive a response. You are better advised to join the ArgoUML users mailing list, and submit your request for help there. An empty email to `users-subscribe@argouml.tigris.org` will achieve this (an automatic

reply will take you through the process). You can then submit your request for help by mail to `users@argouml.tigris.org`

- `Snooze Critic` This suspends the activity of the critic that generated the current to-do item. The to-do item (and all others generated by the critic) will disappear from the to-do pane.

  The critic will make up after a period of time. Initially this period is 10 minutes, but it doubles on each successive application of the Snooze button. The critic can be woken up explicitly through the `Critique > Browse Critics…` menu (see Section 12.9.4).

  ### Tip

  Some common critics can fire the whole time as you build a big diagram. Some users find it useful to snooze these critics until the diagram has been completed.

Some of the more common critics now have a "wizard" available to help in fixing the problem. This comprises a series of pages in the `To Do Item` tab that step you through the changes. The wizard is driven through the first three buttons at the bottom of the tab.

- `Back`. This will take you back to the previous step in the wizard. Grayed out if this is the first step.

- `Next`. This will take you back to the next step in the wizard. Grayed out if this is the last step.

- `Finish`. This will commit the changes you have made through the wizard. Grayed out until you get to the last step.

  ### Note

  Not all to-do items yet have wizards in the current release of ArgoUML. If there is no wizard all three buttons will remain grayed out.

There is one remaining button at the bottom of the `To Do Item` tab, labeled `Help`. This will fire up a browser to a URL with further help.

  ### Warning

  In the current release of ArgoUML the URLs are generally invalid. The few that are valid have no information.

10.4. Properties Tab

Through this tab, the properties of artifacts selected in the navigation pane or editing pane may be set. The properties of an artifact may be displayed in one of two ways:

1. Selection of the artifact in the navigation or editing panes, followed by selection of the properties tab in the details pane; or

2. Context sensitive pop-up menu `Properties` (using button 2 click) on an artifact in the editing or navigation pane, which will select the item *and*select the `Properties` tab in the details pane.

Figure 10.5 shows a typical properties tab for an artifact in ArgoUML (in this case a class).

Figure 10.5. A typical Properties tab on the details pane
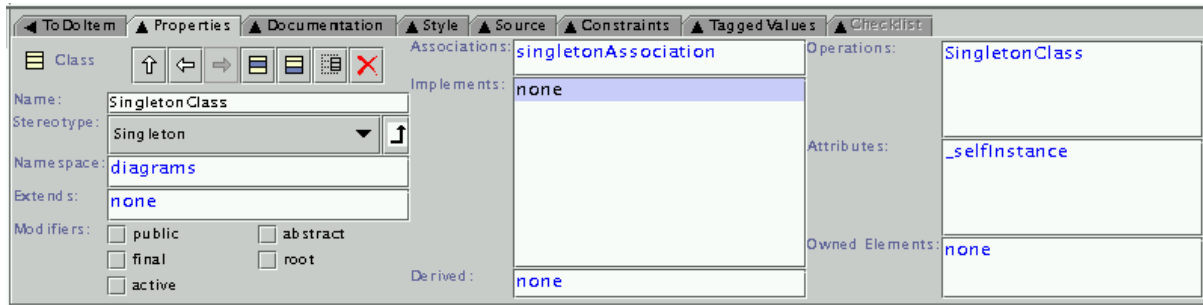
At the top left is the icon and name of the type of artifact (i.e. the UML metaclass, not the actual name of this particular artifact). In this example the property tab is for a class.

To the right of this is a toolbar of icons relevant to this property tab. The first three are always navigation (Go up, Go back and Go forward). The last is always Delete to delete the selected artifact from the model.

The remainder of the tab comprises fields, laid out in two or three columns. Each field has a label to its left. The fields may be text boxes, text areas, drop down selectors, radio boxes and check boxes. In most (but not all cases) the values can be changed. In the case of text boxes this is sometimes by just typing the required value.

However for most text boxes and all text areas, data entry is usually via a context sensitive pop-up menu (using button 2 click), which offers options to open an entry (i.e. transfer to its property sheet), add a new entry, delete an entry or move entries up and down (in text areas with multiple entries).

The first field is almost always a text field Name, where the name of the specific artifact can be entered. The remaining fields vary depending on the artifact selected.

The detailed property sheets for all ArgoUML artifacts are discussed in separate chapters for each of the diagram types (use case diagram (Chapter 15, class diagram (Chapter 16, sequence diagram (Chapter 17, state diagram (Chapter 18, collaboration diagram (Chapter 19, activity diagram (Chapter 20, deployment diagram (Chapter 21). Property sheets for artifacts that are common to all diagram type have their own chapter (Chapter 14).

Caution

ArgoUML will always try to squeeze all fields on to the property sheet. If the size of the property tab is too small, it may become unusable. The solution is to either enlarge the property tab by moving the dividers to left and top, or to split it off altogether as a separate window (see Section 7.4.2).

10.5. Documentation Tab

Within the UML 1.3 standard, all artifacts are children of the Element metaclass. The Element metaclass defines a tagged value documentation for comment, description or explanation of the element to which it is attached. Since this tagged value applies to every artifact, it is given its own tab in the details pane, rather than being part of the Tagged Values tab.

Figure 10.6 shows a typical documentation tab for an artifact in ArgoUML.

Figure 10.6. A typical Documentation tab on the details pane
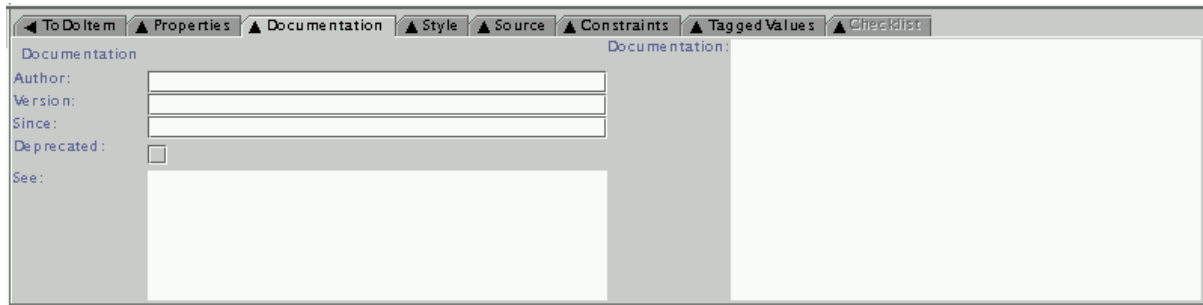
The fields on this tab are the same for all artifacts.

- `Author:` A text box for the author of the documentation.

- `Version:` A text box for the version of the documentation.

- `Since:` A text box to show how long the documentation has been valid.

- `Deprecated:` A check box to indicate whether this artifact is deprecated (i.e. planned for removal in future versions of the design model).

- `See:` Pointers to documentation outside the system.

- `Documentation:` Literal text of any documentation.

Tip

ArgoUML is not primarily a documentation system. For artifacts that require heavy documentation, notably use cases, the use of the `See:` field to point to external documents is more practical.

10.6. Style Tab

This tab provides some limited control over the graphical representation of artifacts in the editing pane.

Caution

Artifacts that do not have any specific direct graphical representation on the screen (beyond their textual description) do not have style tabs of their own. They show the style tab of their nearest parent that does have a style sheet, and any changes will affect that parent. For example the style sheet of an operation on a class will be the style sheet of the class.

Style sheets vary a little from artifact to artifact, but Figure 10.7 shows a typical style tab for an artifact in ArgoUML (in this case an interface).

Figure 10.7. A typical Style tab on the details pane
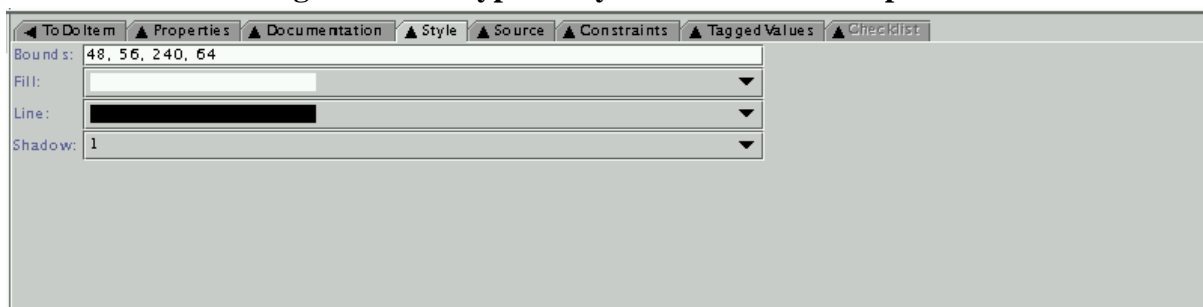
There may be further fields in some cases, but the four fields shown are common to many artifacts.

- `Bounds:` This defines the corners of the bounding box for a 2D artifact. It comprises four numbers separated by periods. These four numbers are respectively: i) the X coordinate of the upper left corner of the box; ii) the Y coordinate of the upper left corner of the box; iii) the width of the box; and iv) the height of the box. All units are pixels on the editing pane.

  ### Caution

  This field has no effect on 1D artifacts that link other artifacts (associations, generalizations etc), since their position is constrained by their connectedness.

  ### Warning

  In the current version of ArgoUML there is strange behavior when typing in this field. Be wary of the cursor jumping around unexpectedly.

- `Fill:` This drop-down selector specifies the fill color for 2D artifacts. It is not present for line artifacts.

- `Line:` This drop-down selector specifies the line color for artifacts.

- `Shadow:` This drop-down selector specifies the width of the shadow (if any) for 2D artifacts. It is not present for line artifacts.

10.7. Source tab

This tab is a single box, containing the source code that will be generated for this artifact. ArgoUML generates a template of the code required (most specifically for classes). The intention is that you will add artifact specific code here, and it will be incorporated.

Warning

This is part of ArgoUML that is very much under development. At this stage any code you add will be discarded (and not saved either).

10.8. Constraints Tab

Constraints are one of the extension mechanisms provided for UML. ArgoUML is equipped with a powerful constraint editor based on the *Object Constraint Language (OCL)Object Constraint Language (OCL)* defined in the UML 1.3 standard.

The standard pre-defines a small number of constraints (for example the `xor` constraint over a set of associations indicating that only one may be manifest for any particular instance.

The standard also envisages a number of circumstances where general purpose constraints may be useful:

- To specify invariants on classes and types in the class model;

- To specify type invariants for stereotypes;

- To describe pre- and post-conditions on operations and methods;

- To describe guards;

- As a navigation language; and

- To specify constraints on operations.

Figure 10.8 shows a typical constraint tab for a artifact in ArgoUML (in this case a class).

Figure 10.8. A typical Constraints tab on the details pane
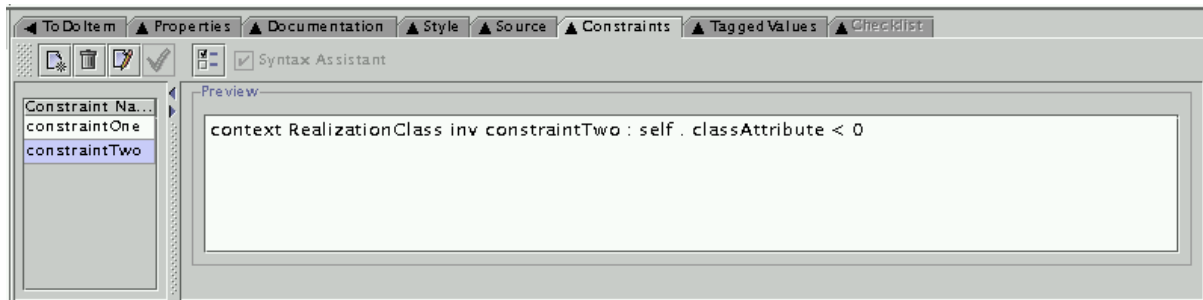
Along the top of the tab are a series of icons.

- New Constraint. This creates a new constraint and launches the constraint editor in the Con-straints tab for that new constraint (see Section 10.8.1). The new constraint is created with a context declaration for the currently selected artifact.

- Delete Constraint. The constraint currently selected in the Constraint Name box (see be-low) is deleted.

- Edit Constraint. This launches the constraint editor in the Constraints tab (see Section 10.8.1). The editor is invoked on the constraint currently selected in the Constraint Name box.

- Check OCL Syntax. This button is intended to check the syntax of the selected constraint.

  ## Caution

  In the current release of ArgoUML this button is always grayed out on the main Constraints tab. It may only be invoked from within the constraint editor (see Section 10.8.1).

- Configure Constraint Editor. This a dialog to configure options in the constraint editor (see Figure 10.9).

Figure 10.9. Dialog box for configuring constraints
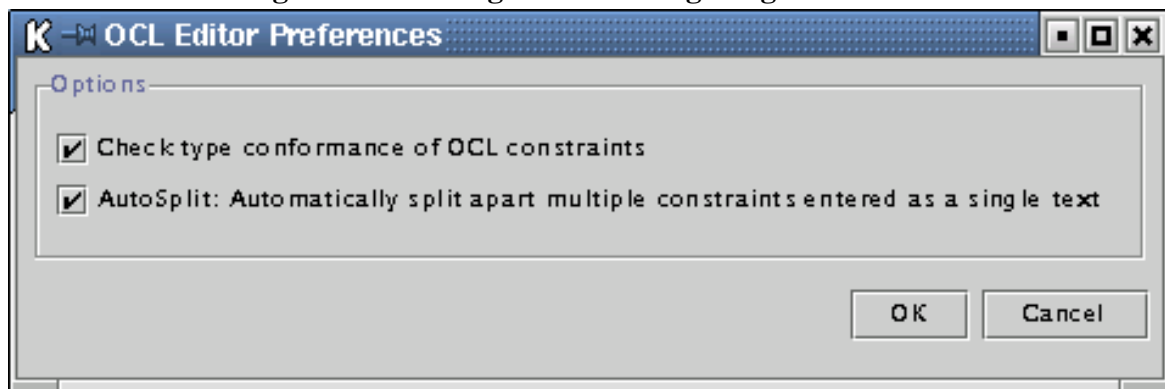
The dialog box has a check box for two options, both enabled by default.

- `Check type conformance of OCL constraints`. OCL is strictly typed. At the early stages of design it may be helpful to disable type checking, rather than follow through all the detailed specification needed to get type consistency.

- `AutoSplit: Automatically split apart multiple constraints entered as a single text`. If you enter multiple constraints in the editor, this will cause them to be split into separate constraints.

At the bottom are two buttons, labeled `OK` (to accept the option changes) and `Cancel` (to discard the changes).

To the right of the toolbar is a check box labeled `Syntax Assistant` (unchecked by default), which will enable the syntax assistant in the constraint editor (see Section 10.8.1).

Caution

In the current release of ArgoUML this check box is always grayed out on the main `Constraints` tab. It may only be changed from within the constraint editor.

The main body of the constraints tab comprises two boxes, a smaller to the left and a larger one to the right. The two are separated by two small arrow buttons which control the size of the boxes.

- `Shrink Left`. Button 1 click on this icon shrinks the box on the left. Its effect may be reversed by use of the `Shrink Right` button (see below).

- `Shrink Right`. Button 1 click on this icon shrinks the box on the right. Its effect may be reversed by use of the `Shrink Left` button (see above).

Finer control can be achieved by using button 1 motion to drag the dividing bar to left and right.

The box on the left is titled `Constraint Name` and lists all the constraints (if any) so far defined for the selected artifact. A constraint may be selected by button 1 click.

The box on the right is labeled `Preview` and contains the text of the constraint. Where a constraint is too large for the box, a scroll bar is provided to the right.

10.8.1. The Constraint Editor

This is invoked through the use of the `New Constraint` or `Edit Constraint` buttons on the main `Constraints` tab. The constraint editor takes up the whole tab (see Figure 10.10).

Figure 10.10. Dialog box for configuring constraints
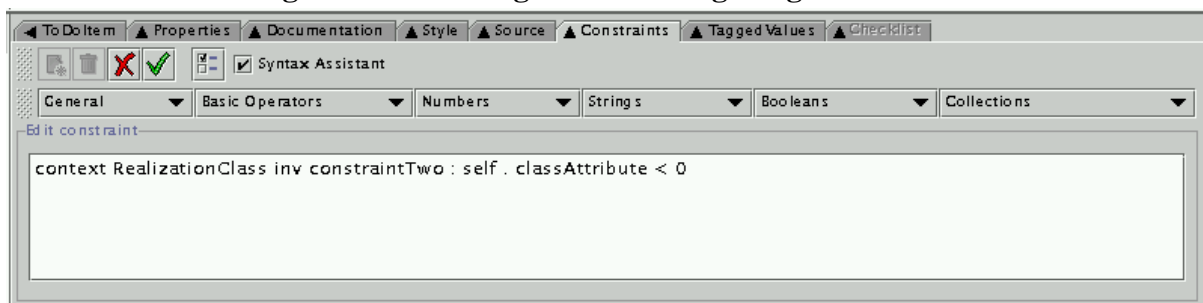
Along the top of the tab are a series of icons.

- ![icon] New Constraint. This is intended to create a new constraint for editing.

Caution

In the current release of ArgoUML this button is always grayed out in the constraints editor. It can only be invoked from the main Constraints tab (see Section 10.8).

- ![icon] Delete Constraint. The constraint is intended to delete the constraint being edited.

Caution

In the current release of ArgoUML this button is always grayed out in the constraints editor. It can only be invoked from the main Constraints tab (see Section 10.8).

- ![icon] Cancel Edit Constraint. This exits the constraint editor without saving any changes and returns to the main Constraints tab.

- ![icon] Check OCL Syntax. This button invokes a full syntax check of the OCL written in the editor. If the syntax is valid, the constraint is saved, and control returns to the main Constraints tab. If the syntax is not valid, a dialog box explains the problem.

Tip

Whether type checking is included is configurable using the Configure Constraint Editor button (see Section 10.8 for more details).

- ![icon] Configure Constraint Editor. This a dialog to configure options in the constraint editor. It is also available in the main Constraints tab and is discussed in detail there (see Section 10.8).

To the right of the toolbar is a check box labeled Syntax Assistant (unchecked by default), which will enable the syntax assistant in the constraint editor.

If the syntax assistant is enabled, six drop down menus are provided in a row immediately after the toolbar. These provide standard templates for OCL that, when selected, will be inserted into the constraint being edited.

- General. General OCL constructors. Entries: inv (inserts an invariant); pre (inserts a pre-condition); post (inserts a post-condition); self (inserts a self-reference); @pre (inserts a reference to a value at the start of an operation); and result (inserts a reference to a previous result).

- Basic Operators. Relational operators and parentheses. Entries: =; <>; <; >; <=; >=; and ().

- Numbers. Arithmetic operators and functions. Entries: +; -; *; /; mod; div; and abs.

- Strings. String functions. Entries: concat; size; toLower; toUpper; and substring.

- Booleans. Logical functions. Entries: or; and; xor; not; implies; and if then else.

- Collections. Operators and functions on collections—bags, sets and sequences. The large number of functions are organized into sub-groups.

  - General. Functions that apply to all types of collection. Entries: Collection {} (insert a new collection); Set {} (insert a a new set); Bag {} (insert a new bag); Sequence {} (insert a new sequence); size; count; isEmpty; notEmpty; includes; includesAll; iterate; exists; forAll; collect; select; reject; union; intersection; including; excluding; and sum.

  - Sets. Operators and functions that apply only to sets. Entries: - (set difference); and symmetricDifference.

- General. Functions that apply to sequences. Entries: `first`; `last`; `at`; `append`; `prepend`; and `subSequence`.

The remainder of the tab comprises a writable text area containing the text to be edited. The mouse buttons have their standard behavior within an editable text area (see Section 7.3).

In addition, cut, copy and paste operations may be invoked through the keyboard shortcuts `Ctrl-X`, `Ctrl-C` and `Ctrl-V` respectively.

Warning

There is a known problem with the current release of ArgoUML. The `Cut`, `Copy` and `Paste` functions on the `Edit` menu cannot be used within the constraint editor. If used they will work on the currently selected artifact in the editing pane.

Given the known problems with these menu buttons in the current release of ArgoUML (see Section 12.4.4, Section 12.4.5 and Section 12.4.6) this could disrupt the model. It should be avoided at all costs.

10.9. Tagged Values Tab

Tagged values are another extension mechanism provided by UML. The user can define name-value pairs to be associated with artifacts which define properties of that artifact. The names are known as *tagstags*. UML predefines a number of tags that are useful for many of its artifacts.

Note

The `documentation` tag is defined for the top UML metaclass, `Element` and is so available to all artifacts. In ArgoUML documentation values are provided through the `Documentation` tab, rather than by using the `Tagged Values` tab.

The `Tagged Values` tab in ArgoUML comprises a simple two column table, with an editable box on the left for the tag name and an editable box on the right for the associated value. There is always at least one empty row available for any new tag. The mouse buttons have their standard behavior within an editable text area (see Section 7.3).

In addition, cut, copy and paste operations may be invoked through the keyboard shortcuts `Ctrl-X`, `Ctrl-C` and `Ctrl-V` respectively.

Warning

There is a known problem with the current release of ArgoUML. The `Cut`, `Copy` and `Paste` functions on the `Edit` menu cannot be used within the constraint editor. If used they will work on the currently selected artifact in the editing pane.

Given the known problems with these menu buttons in the current release of ArgoUML (see Section 12.4.4, Section 12.4.5 and Section 12.4.6) this could disrupt the model. It should be avoided at all costs.

10.10. Checklist Tab

ArgoUML provides the concept that the user can maintain a checklist of things to do. The `Checklist` tab will eventually provide the mechanism to do this.

Caution

In the current release of ArgoUML this tab is not implemented, and is always grayed out.

Chapter 11. The To-Do Pane

Table of Contents

11.1. Introduction

Figure 11.1 shows the ArgoUML window with the to-do pane highlighted.

Figure 11.1. Overview of the to-do pane
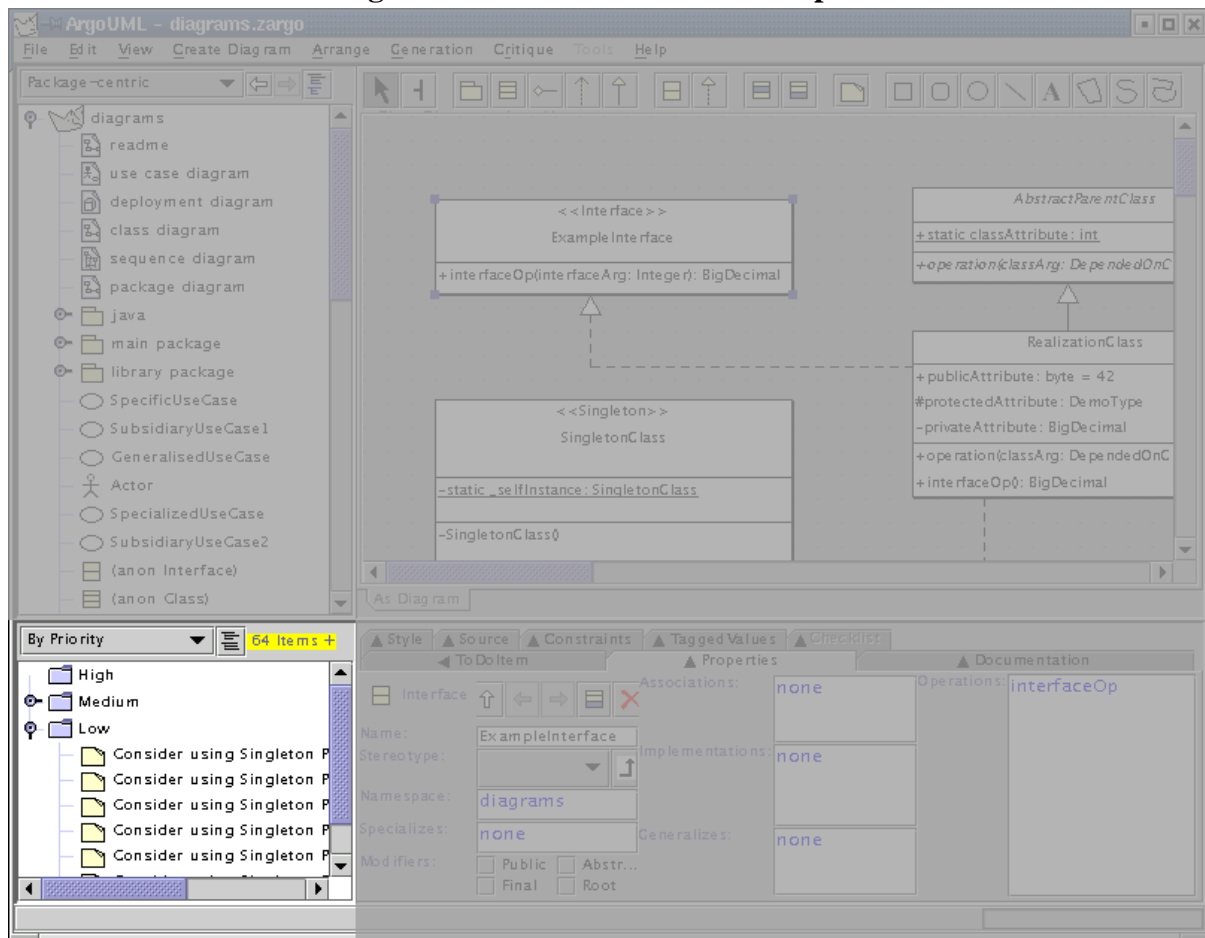
This pane provides access to the advice that comes from the critics processes running within ArgoUML.

A selector box at the top allows a choice of how the data is presented, a button allows the display of the hierarchy to be changed, and there is an indicator the number of to-do items identified.

95

More information on critics can be found in the discussion of the `Critique` menu (see Section 12.9).

11.2. Mouse Behavior in the To-Do Pane

Behaviour of the mouse in general, and the naming of the buttons is covered in the chapter on the overall user interface (see Chapter 7).

11.2.1. Button 1 Click

This action is generally used to select an item for subsequent operations.

Within the hierarchical display, elements which have sub-hierarchies may be indicated by when the hierarchy is hidden and when the hierarchy is open.

When these icons are displayed, the display of the hierarchy is toggled by button 1 click on these icons.

> **Tip**
>
> It is possible to turn off the display of these icons using the `Flat` button in the navigation pane.

Button 1 click over the headline of any to-do item will cause its details to be shown in the `To Do Item` tab of the details pane. That tab is automatically selected if it is not currently visible.

11.2.2. Button 1 Double Click

When applied to the folder icon alongside a hierarchy category, this will cause the display of that hierarchy to be toggled.

When applied to a headline, button 1 double click will show the diagram for the artifact to which the to-do item applies in the editing pane and select the artifact on the diagram using an appropriate clarifier (the artifact may be highlighted, underlined with a wavy line or surrounded by a colored box as appropriate).

11.2.3. Button 2 Actions

There are no button 2 functions in the to-do pane.

11.2.4. Button 2 Double Click

There are no button 2 functions in the to-do pane.

11.3. Presentation Selection

At the top of the pane is a drop-down selector controlling how the to-do items are presented. The to-do items may be presented in six different ways.

- `By Priority`. The to-do items are organized into three hierarchies by priority: `High`, `Medium` and `Low`. The priority associated with the to-do items generated by a particular critic may be altered through the `Critique > Browse Critics…` menu (see Section 12.9.4).

- `By Decision`. The to-do items are organized into 16 hierarchies by design issue: `Uncategorized`, `Class Selection`, `Naming`, `Storage`, `Planned Extensions`, `MState Machines`, `Design Patterns`, `Relationships`, `Instantiation`, `Modularity`, `Expected Usage`, `Meth-`

ods, `Code Generation`, `Stereotypes`, `Inheritance` and `Containment`. The details of the critics in each category are discussed in Section 12.9.2.

- `By Goal`. ArgoUML has a concept that critics may be grouped according to the user goals they affect. This presentation groups the to-do items into hierarchies by goal.

    ### Caution

    In the current release of ArgoUML there is only one goal, `Unspecified` and all to-do items will appear under this heading.

- `By Offender`. The to-do items are organized into a hierarchy according to the artifact that caused the problem.

    ### Caution

    In the current release of ArgoUML collaboration and sequence diagrams which generate to-do items sometimes appear with internal package reference names, rather than their given name.

- `By Poster`. The to-do items are organized into a hierarchy according to which critic generated the to-do item.

    ### Caution

    In the current release of ArgoUML the fully qualified class name of the critic is given instead of just its headline name.

- `By Knowledge Type`. ArgoUML has the concept that a critic reflects a deficiency in a category of knowledge. This presentation option groups the critics according to their knowledge category: `Designer's`, `Correctness`, `Completeness`, `Consistency`, `Syntax`, `Semantics`, `Optimization`, `Presentational`, `Organizational`, `Experiencial` and `Tool`.

11.4. Flat/Hierarchical Button

To the right of the presentation selections is a button which toggles whether hierarchies are shown with "twisties" ( ) to indicate the hierarchy.

- Hierarchical. The current presentation uses "twisties" to show the hierarchy. Button 1 click will on the icon will switch to a representation without them.

- Flat. The current presentation does not use "twisties" to show the hierarchy. Button 1 click will on the icon will switch to a representation which has them.

    ### Caution

    The current version of ArgoUML (as described above) really has the flat/hierarchical icons logically back to front.

11.5. Item Count

To the right of the flat/hierarchical button is a count of the number of to-do items currently found, highlighted in yellow. This is followed by + to indicate the last change in number was an increase or − to indicate the last change in number was a decrease.

Caution

This count will not show if the to-do pane is shrunk too small.

Chapter 12. The Menu bar

Table of Contents

12.1. Introduction

An important principle behind ArgoUML is that actions should be able to be invoked in whatever way the user finds convenient. As a result many (but not all) actions that can be carried out on the menu can be carried out in other ways as well under ArgoUML.

A number of the common menu entries are also available through keyboard shortcuts.

Warning

It should also be possible to navigate the menu from the keyboard, Motif style. Each level of each menu is identified by a letter (shown underlined in the menu or entry name). This sequence of letters while holding down the ALT key should select the entry. In the current version of ArgoUML this functionality is only available when running under Microsoft Windows.

12.2. Mouse Behavior in the Menu Bar

Behaviour of the mouse in general, and the naming of the buttons is covered in the chapter on the overall user interface (see Chapter 7).

12.2.1. Button 1 Click

Over a top level drop down menu this will expose the menu. If the button is help down and moved, the behaviour of button 1 motion (see below) will follow.

Over an exposed menu entry this will invoke the action of the menu.

Within menu dialog boxes this will generally select a text item, and invoke a button action.

12.2.2. Button 1 Double Click

This has no effect other than that of button 1 single click.

12.2.3. Button 1 Motion

Moving over a top level menu, or a sub-menu will cause that menu to be exposed.

If the button is released (thereby completing a click) over a menu entry that entry will be invoked.

12.2.4. Button 2 Actions

In general button 2 click behaves as button 1 click with the menu bar.

12.2.5. Button 2 Double Click

In general button 2 double click behaves as button 1 click with the menu bar.

12.2.6. Button 2 Motion

In general button 2 motion behaves as button 1 click with the menu bar.

12.3. The File Menu

These are actions concerned with input and output and the overall management of projects and the ArgoUML system.

12.3.1. New…

Shortcut Ctrl-N.

This initializes a new project within ArgoUML. The project is created with the name `untitledModel` and with two empty diagrams: a class diagram and a use case diagram.

Note

Although the entry is named New… there is no dialog—the project is just created. The entry should properly be just `New`.

Caution

`untitledModel` is not a valid model name (upper case is not permitted). See Section 14.2 for a discussion of this.

12.3.2. Open Project…

Shortcut Ctrl-O.

This opens an existing project. Selecting this menu option will open a file selection dialog (see Figure 12.1).

Figure 12.1. The file selection dialog for Open Project….
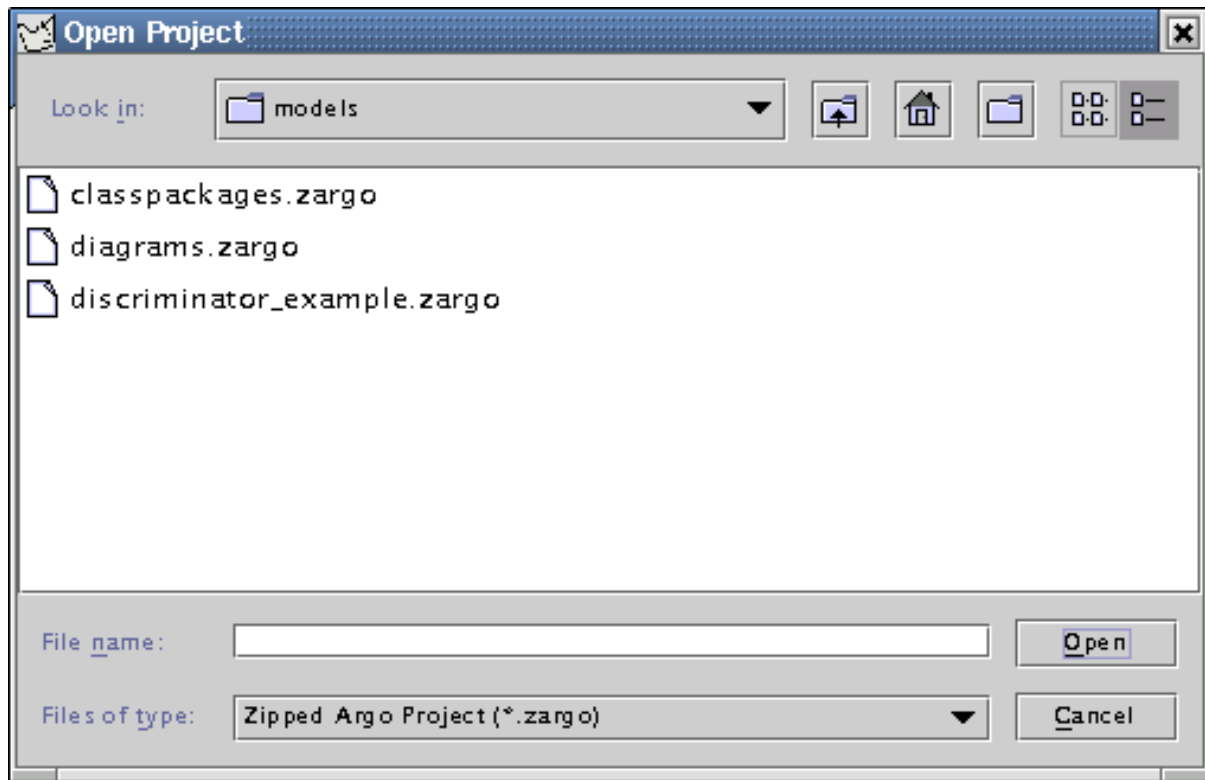
At the top of the dialog is a drop down selector labeled `Look in:`, specifying the currently selected directory preceded by the folder icon, ⬚. The drop down selector gives access to all directories above this directory.

Next to this are four icons: ⬚ to move up a directory; ⬚ to select the users home directory, ⬚ to specify that a simple file list is displayed; and ⬚ (which is always grayed out) that would show details of all files if enabled.

The main body of the dialog is a text area with a listing of all files in the currently selected directory which match the current filter (see below). If these will not all fit on the display, scroll bars are provided to move up and down.

In the lower portion of the dialog is a text box labeled `File name:` for the name of the file to be opened. The file name may be typed directly in here, or selected from the directory listing above using button 1 click.

Beneath this is a drop down selector labeled `Files of type:` to specify a filter on the files to be shown in the directory listing. Only files that match the filter are listed. The available filters are:

- all files;

- zipped Argo project files (the default);

- Argo project files; and

- XMI format files.

Argo project files are produced by earlier releases of ArgoUML, XMI files allow you to use diagrams from other tools (which may not always work!).

To the right of the file name and filter are two buttons, `Open` and `Cancel` to open the selected project or cancel the operation.

Note

Open will only succeed if a valid filename is provided.

12.3.3. Save Project

Shortcut Ctrl-S.

This saves the project using its current file name.

Note

A newly created project (see Section 12.3.1) will not have been saved to file. And so this entry is grayed out. Use `Save Project As…` to save the project to a file for the first time (see Section 12.3.4).

12.3.4. Save Project As…

This opens a dialog allowing you to save the project under a different file name (or to specify a file name for the first time if you are a new project).

Caution

If you are using an experimental version of ArgoUML you will receive a warning message that file formats are not guaranteed to be supported in the long term. You should be cautious about using unstable versions of the system for important projects.

The dialog box is identical to that for `Open Project` (see Figure 12.1). The `Files of type:` also specifies the file type used for saving, and you should choose a filename with the corresponding ending.

Tip

Zipped Argo Project files, ending in `.zargo` (the default) are the most efficient way of saving projects. They capture all the model and graphical information in a very compact format.

12.3.5. Import…

This sub-menu provides access to mechanisms to import a variety of material in to ArgoUML. In the current version of ArgoUML only one type of import is provided as standard, the importing of Java code for reverse engineering.

Note

Despite being named `Import…` there is no dialog box associated with the Import menu itself (there may be with the sub-menu entries). It should strictly speaking be labeled just `Import` on the menu.

12.3.5.1. Import Sources…

A very powerful feature of ArgoUML is that it can "Reverse Engineer" Java code to yield a class diagram. This sub-menu entry specifies Java code to be imported for reverse engineering.

The dialog box is very similar to that for `Open Project` (see Figure 12.1), but with two extra tabs placed alongside the directory listing, as shown in Figure 12.2).

Figure 12.2. The file selection dialog for Import Sources….
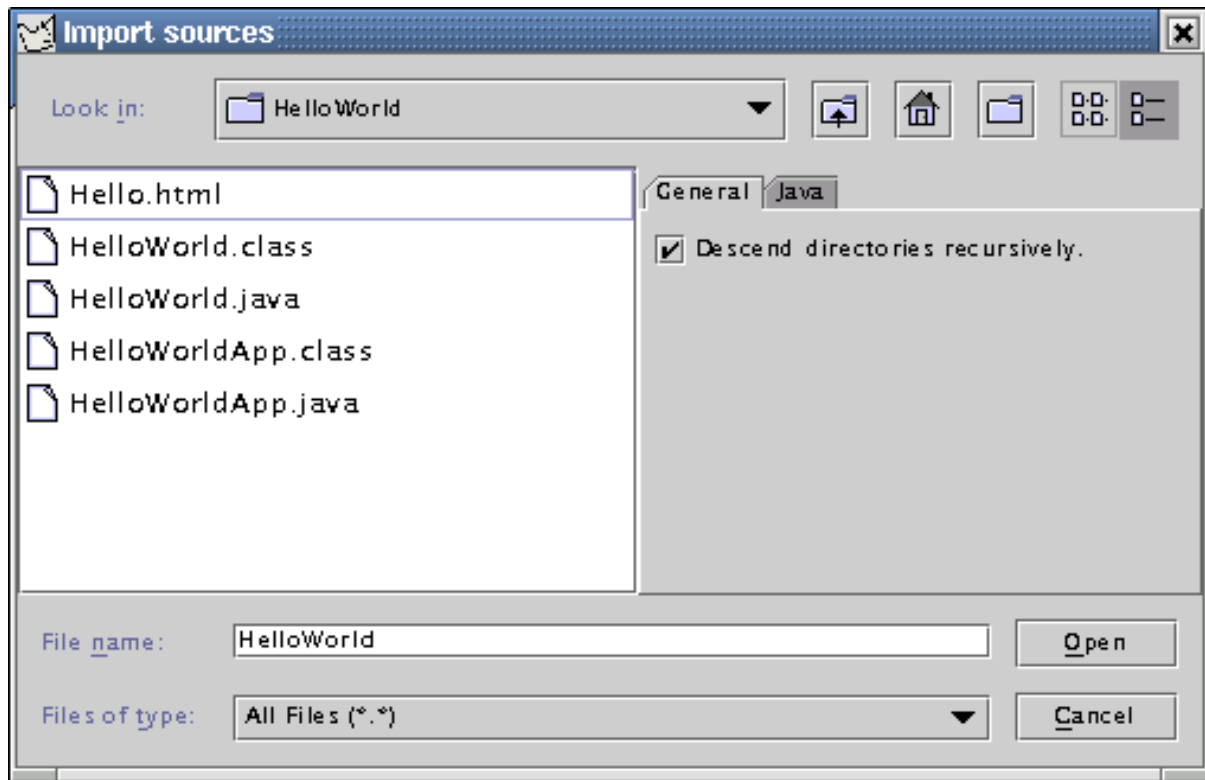
Those fields that are the same as Open Project behave in the same way (see Section 12.3.2).

Caution

The only file filter provided selects all files. However reverse engineering is currently only available, and Java source files only should be selected.

The first of the two tabs is labeled General and is selected by button 1 click on its tab. It provides a single check box Descend directories recursively. If enabled (the default), reverse engineering will track through sub-directories for any further Java files. If not it will restrict to the selected directory.

The second of the two tabs is labeled Java and is selected by button 1 click on its tab. It provides two pairs of radio boxes.

The first radio box allows selection between modeling attributes of Java classes as UML attributes (the default) or as UML associations to the class specified.

The second radio box allows selection between modeling arrays as v new datatypes in their own right (the default) or as their base datatype with multiplicity.

12.3.6. Load Model From DB

ArgoUML has the ability to save models to and from an SQL database. However the code for this is quite old, and at present this menu option is grayed out.

12.3.7. Save Model To DB

ArgoUML has the ability to save models to and from an SQL database. However the code for this is quite old, and at present this menu option is grayed out.

12.3.8. Print…

Shortcut Ctrl-P.

This brings up a dialog box allowing the ArgoUML model to be printed out.

Warning

In the current release of ArgoUML this function behaves strangely if you have no printer configured under Linux. A small dialog box appears, but it has no functionality.

12.3.9. Save Graphics…

This menu entry brings up a dialog box allowing the currently selected diagram (in the editing pane) to be saved in one of a number of graphic formats.

The dialog box is identical to that for Open Project (see Figure 12.1). The Files of type: also specifies the graphics type used for saving, and you should choose a filename with the corresponding ending.

The available graphics types are:

- GIF (the default);

- Postscript;

- Encapsulated Postscript; and

- Scalable Vector Graphics (SVG).

The filter also allows you to specify all files, in which case the file type is deduced from the filename ending.

Caution

You should be aware that there are patent issues surrounding GIF files and act accordingly. There are *currentlycurrently* no such concerns about SVG.

Warning

If you use the all files filter and specify a filename ending that ArgoUML does not support (i.e. anything other than .gif, .ps, .eps and .svg) ArgoUML will silently do nothing.

12.3.10. Save Configuration

ArgoUML has various user specific configurations that can be set, both through the Settings… entry on the Edit menu (see Section 12.4.10) and directly on the various panes. This menu entry saves the information in the file argo.user.properties in the users home directory.

Tip

This is a text file, which you can edit to configure ArgoUML.

12.3.11. Exit

Shortcut Alt-F4.

This closes down ArgoUML A warning message will pop-up if you have a project open with unsaved changes asking if you wish to save it. The three options are:

- Yes (save the project and exit ArgoUML);

- `No` (do not save the project, but still exit ArgoUML); and

- `Cancel` (do not save the project and do not exit ArgoUML).

12.4. The Edit Menu

This menu provides support for selecting artifacts on the editing pane; an undo/redo function; cut, copy and paste functions; removal of artifacts from diagrams and the model; and control of user settings.

12.4.1. The Select Sub-Menu

This sub-menu provides for selection of items on the editing menu. It has four entries.

- `Select All` (shortcut Ctrl-A). Selects all artifacts on the editing pane.

- `Select Previous`. ArgoUML has a concept of the ordering of artifacts on a diagram. This selects the item prior to the currently selected item in that ordering.

Caution

The behavior of this menu function can be very strange, selecting items that do not appear to be on any diagram. It is probably best avoided.

- `Select Next`. Similarly this selects the item following the currently selected item in that ordering.

Caution

Again the behavior of this menu function can be very strange, selecting items that do not appear to be on any diagram. It is probably best avoided.

- `Invert Selection`. This inverts the current selection (i.e. everything that was selected is de-selected and everything that was not selected is selected).

12.4.2. Undo

This is intended to undo editing actions.

Warning

In the current implementation of ArgoUML this function is not available and is permanently grayed out.

12.4.3. Redo

This is intended to redo editing actions—that is to reverse the effect of the Undo function.

Warning

In the current implementation of ArgoUML this function is not available and is permanently grayed out.

12.4.4. Cut

Shortcut Ctrl-X.

This should take the currently selected items, remove them from the diagram, and place them on the clipboard for subsequent pasting.

Warning

In the current implementation of ArgoUML there are some difficulties with clipboard based functions. This menu option is permanently grayed out.

12.4.5. Copy

Shortcut Ctrl-C.

This should take the currently selected items and make a copy of them on the clipboard for subsequent pasting.

Warning

If the original item is deleted from the model or removed from the diagram, subsequent pasting will produce a copy that has no namespace, and cannot be properly saved.

In addition pasting back to the diagram from which the copy was made will replace the original, rather than duplicating it.

In the light of this, the copy function is best avoided.

12.4.6. Paste

Shortcut Ctrl-V.

This should paste the item(s) last cut or copied to the clipboard on to the currently diagram in the drawing pane.

Warning

If the original item has since been deleted from the model or removed from the diagram, pasting will produce a copy that has no namespace, and cannot be properly saved.

In addition pasting back to the diagram from which the copy was made will replace the original, rather than duplicating it.

In the light of this, the paste function is best avoided.

12.4.7. Remove From Diagram

Shortcut Ctrl-R.

This removes the currently selected item(s) from the diagram, but not from the model.

Caution

In previous versions of ArgoUML there was a problem where, in some circumstances, the item(s) were deleted from the model completely, not just from the diagram.

This is now believed to be fixed, but you should consider saving your model prior to using this function.This is now believed to be fixed, but you should consider saving your model prior to using this function.

12.4.8. Delete From Model

Shortcut Delete.

This function deletes the selected item(s) from the model completely.

Caution

In the current version of ArgoUML the item(s) may not disappear from diagrams other than the current one. However the copies on other diagrams will have no associated properties. They should be deleted, or will disappear after the model is saved and reloaded.

12.4.9. Empty trash

This menu function has no discernible effect. It is believed to invoke a garbage collection in the Java system.

12.4.10. Settings…

This menu entry brings up a dialog box, which allows the user to set various options that control the behavior of ArgoUML (see Figure 12.3).

Figure 12.3. The dialog for Settings.
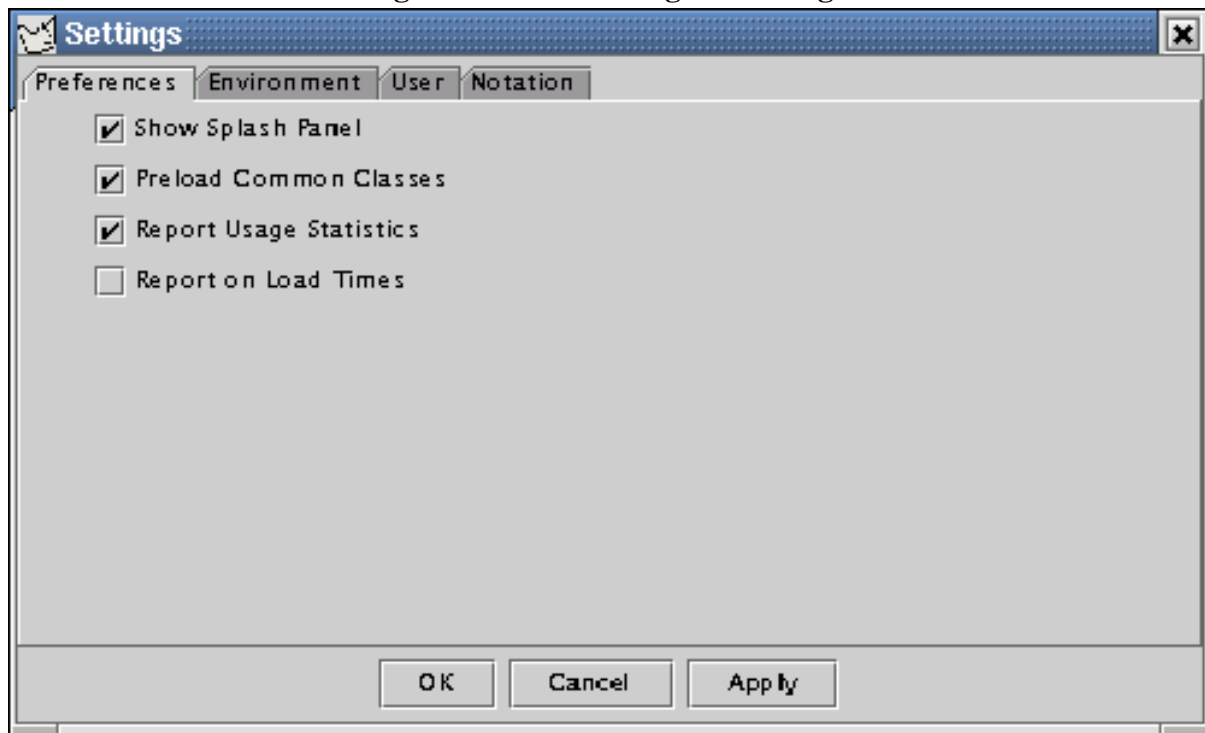
The options that can be set up on the various tabs are described in the following sections. For each tab there are three buttons at the bottom of the dialog box.

- OK. Selecting this button (button 1 click) applies the chosen settings and exits the dialog.

- Cancel. Selecting this button (button 1 click) exits the dialog without applying any settings changed since the last Apply (or since the dialog started if Apply has not been used).

- Apply. Selecting this button (button 1 click) applies the chosen settings and remains in the dialog.

These settings can be saved for use by subsequent ArgoUML sessions (see Section 12.3.10).

12.4.10.1. Preferences Tab

Selecting the `Preferences` tab (button 1 click on the tab) gives four options as check boxes.

- `Show Splash Panel` (set by default). If enabled ArgoUML will show a small panel with a picture while loading.

  ### Tip

  The splash panel can be seen by using the Help menu (see Section 12.11.1).

- `Preload Common Classes` (set by default). If enabled ArgoUML creates class objects of a number of classes internally during start up so that instantiation is quicker when they are needed.

- `Report Usage Statistics` (set by default). If enabled usage statistics are reported on the console.

  ### Caution

  In the current version of ArgoUML there is only a very small amount of information, which may be missed amongst the other console output.

- `Report on Load Times` (clear by default). If enabled load times at start up for the various phases of initialization are reported.

  ### Caution

  These times are only of use to developers—they have no meaning to the ordinary user.

12.4.10.2. Environment Tab

Selecting the `Environment` tab (button 1 click on the tab) lists seven environmental items. Note that none of these can be set—this tab is just a matter of record.

- `${argo.root}`. The directory holding the top level argo class definitions. By default the `classes` sub-directory of the ArgoUML build directory.

- `${argo.home}`. The ArgoUML build directory.

- `${argo.ext.dir}`. The directory holding ArgoUML extensions—by default the `ext` sub-directory of the ArgoUML build directory.

- `${java.home}`. The home directory of the Java Runtime Environment (JRE).

- `${user.home}`. The user's home directory.

- `${user.dir}`. The directory from which ArgoUML was started.

- `startup directory`. The directory in which ArgoUML starts file searches etc.

12.4.10.3. User Tab

This tab allows the user to record additional information of use to the system. There are two text boxes provided.

- `Full Name:`. Allows the user to record their full name.

- `Email Address:`. Allows the user to record their Email address.

This information is used when requesting to-do help by Email.

12.4.10.4. Notation Tab

This tab allows the user to specify certain notation settings. It comprises two check boxes.

- `Only allow strict UML notation text` (clear by default). This feature is for future use. One possibility is that a designer may create a diagram and indicate that the notation of the diagram is Java. Someone viewing the diagram with strict UML notation checked will *onlyonly* see UML notation in the classes no matter what notation the designer has used. See also Section 12.5.10).

Warning

This is not currently supported and is not a commitment to the feature of assigning a notation to a diagram either.

- `Use guillemots (« ») for stereotypes` (clear by default). By default ArgoUML uses pairs of "less than" and "greater than" ($<<$ $>>$) characters for stereotypes If this box is checked stereotypes on diagrams are shown between true guillemots (« »).

Caution

This feature is not fully implemented in the current version of ArgoUML.

12.5. The View Menu

This menu is used for actions that affect how the various panes are viewed.

12.5.1. Goto Diagram…

This menu entry brings up a dialog box, describing all the diagrams in the current project under ArgoUML (see Figure 12.4).

Figure 12.4. The dialog for Goto Diagram….
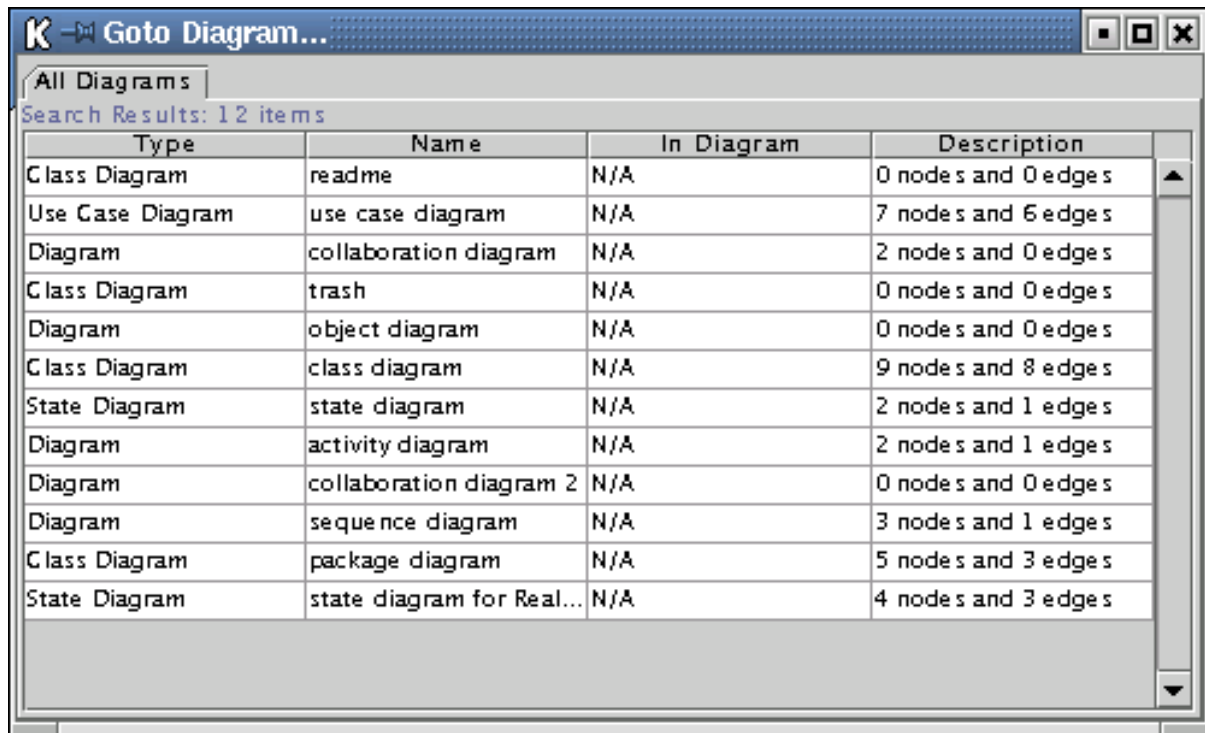
The dialog box is in the form of a tab containing a table with four columns and one row for each diagram in the current project. A scroll bar gives access if the list is too long for the box. Double button 1 click on any row will select that diagram in the editing pane. The four columns are as follows.

- Type. Lists the type of diagram.

  ### Caution

  In the current release of ArgoUML activity diagrams, collaboration diagrams, sequence diagrams and deployment diagrams are all listed as type Diagram.

- Name. Lists the name given to the diagram.

- In Diagram. This shows the diagram in which the digram is contained. Since diagrams cannot be contained within other diagrams, this always shows the entry N/A.

  ### Tip

  This column is present, because this menu item reuses the code for the more general Find menu.

- Description. Shows how many nodes and edges there are on the diagrams. The number of "2-D" artifacts and the number of connector artifacts respectively.

12.5.2. Find…

This menu entry brings up a dialog box for the ArgoUML search engine (see Figure 12.5).

Figure 12.5. The dialog for Find….
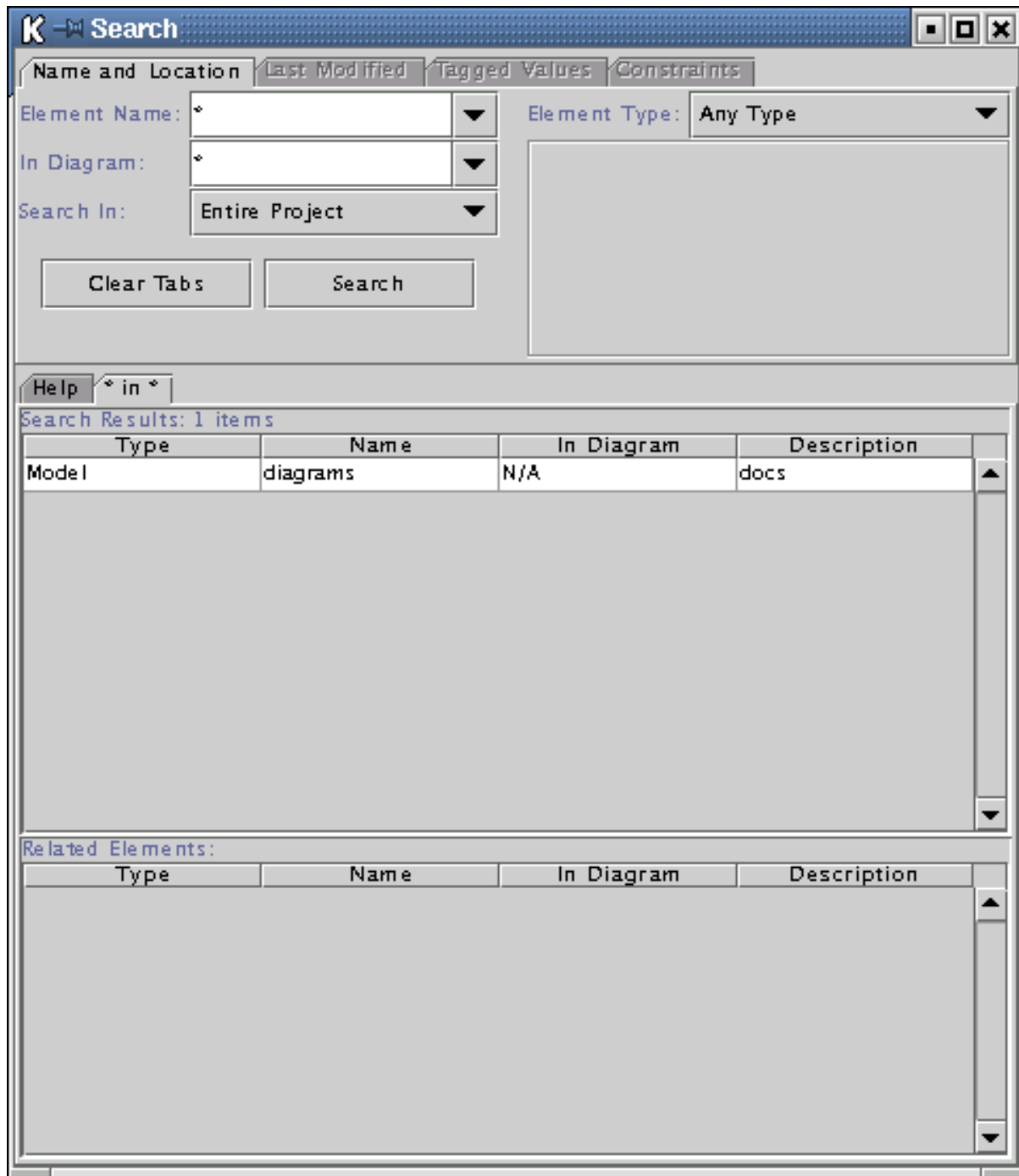
Warning

In the current version of ArgoUML this function is largely broken. The only item that can ever be found is the model itself, and it is not possible (or meaningful) to navigate to this item's diagram.

At the top level, the dialog box has four tabs labeled `Name and Location`, `Last Modified`, `Tagged Values` and `Constraints`. Of these all but the first are grayed out, so the first tab is always selected.

The tab is split into three areas vertically. The top third specifies the search to be made. A text box labeled `Element Name:` specifies the name of the element (or artifact) to search for. Wild cards (`*`, `?`) may be used here. A drop down gives access to search expressions previously used. A text box labeled `In Diagram:` specifies which diagrams are to be searched. Again wild cards may be used. Both these two text boxes have a default

entry of `*`, i.e. match anything.

To the right of these two text boxes, a selector labeled `Element Type:` allows you to specify the UML meta-class for which you are searching.

Caution

The names in of the element types use the NovoSoft UML convention, being the UML metaclass name preceded by `M`. Not all metaclasses are available.

A selector labeled `Entire Project` allows the search to be made over the entire project (the default) or as a sub-search over the results of a previous search.

Beneath these boxes are two buttons.

- `Clear Tabs`. This clears the display of tabs with the results from previous searches (see below). The results are still available however for sub-searching.

- `Search`. This causes the search specified in the text boxes and selectors above to be executed. The results are displayed in a tab taking up the lower two thirds of the page.

The lower two thirds of the dialog comprises an initial tab (labeled `Help`) giving summary help, and further tabs displaying the results of searches. These search tabs are labeled with a summary of the search *ele-elementment* in *diagramdiagram* and are divided horizontally in two halves.

The top half is labeled `Search Results:` followed by a count of the number of items found. It comprises a table with one row for each element (artifact) and four columns.

- `Type`. Lists the type of element (artifact).

- `Name`. Lists the name given to the element (artifact).

- `In Diagram`. Where the artifact is visible on a diagram, this lists the name of the diagram, otherwise it shows `N/A`.

Caution

The behavior when an item is on more than one diagram is unknown.

- `Description`. Contains a description of the element (artifact). At present this seems to be restricted to the single entry `docs`.

Button 1 click on any row will give more information on that artifact and show related artifacts in the bottom half (see below). Double click on any row describing an artifact on a diagram and that item and diagram will be selected.

The bottom half of the tab is a table labeled `Related Elements:` and is a table with the same columns as the top half. When an element (artifact) has been selected in the top half, this table shows the details of any related elements.

12.5.3. The Zoom Sub-Menu

This entry brings up a sub-entry, which scales the view of the current diagram to a given percentage of its notional size.

The sub-menu values that can be selected are `25%`, `50%`, `75%`, `100%`, `125%`, `150%`.

When created all diagrams are at a zoom level of `100%`.

Warning

The current version of ArgoUML has a known problem, where zoom may not work on some diagrams.

12.5.4. The Editor Tabs Sub-Menu

The editing pane is conceived as a multi-mode editor, allowing you to edit a UML diagram from a number of perspectives.

- As a diagram. This views the diagram in conventional graphical terms and is the default.

- As a table. The artifacts on the diagram are listed in a table with a column for each of their properties, which may be edited.

  Note

  Although this representation was available in ArgoUML 0.8, it is not supported in the current version.

- As metrics. This representation is yet to be defined and is not currently available in ArgoUML.

The various representations may be selected by button 1 click on the corresponding tab at the bottom of the editing pane. Since the current version of ArgoUML only supports a diagram representation, there is only one tab labeled As Diagram.

The Editor Tabs sub-menu has four entries.

- Next Editing Tab (shortcut F6). Selects the next tab on the editing pane.

  Caution

  Since there is only one editing tab in the current version of ArgoUML, As Diagram, this always selects that representation, and is thus a null operation.

- As Diagram (shortcut Alt+Shift-1). Selects the As Diagram tab on the editing pane.

- As Table (shortcut Alt+Shift-2). Selects the As Table tab on the editing pane.

  Caution

  Since table representation is not supported in the current version of ArgoUML this entry is always grayed out and not available.

- As Metrics (shortcut Alt+Shift-3). Selects the As Metrics tab on the editing pane.

  Caution

  Since metrics representation is not supported in the current version of ArgoUML this entry is always grayed out and not available.

12.5.5. The Details Tabs Sub-Menu

This provides a sub-menu allowing selection of the various tabs on the details pane for the currently selected artifact. The sub-menu has nine entries.

- Next Details Tab (shortcut F5). Selects the next available tab in the details pane.

  Note

  If a tab is not available, it is skipped (for example a diagram text box only has ToDoItem, Proper-

ties and `Source` tabs, with all other tabs grayed out).

- `ToDoItem` (shortcut Alt-1). Selects the `ToDoItem` tab for the currently selected artifact.

- `Properties` (shortcut Alt-2). Selects the `Properties` tab for the currently selected artifact.

Caution

The `Properties` tab will be selected even if it is not available for the current artifact, in which case it will be completely blank.

- `Javadocs` (shortcut Alt-3). Selects the `Javadocs` tab for the currently selected artifact.

Warning

The current version of ArgoUML has no `Javadocs` tab, so this entry is always grayed out.

- `Source` (shortcut Alt-4). Selects the `Source` tab for the currently selected artifact.

Caution

The `Source` tab will be selected even if it is not available for the current artifact, in which case it will just hold the grayed out text `Nothing selected`

- `Constraints` (shortcut Alt-5). Selects the `Constraints` tab for the currently selected artifact.

Warning

The `Constraints` tab will be selected even if it is not available for the current artifact, in which case it will present a standard constraints tab with no entries. The side effects of entering any data are unknown.

- `Tagged Values` (shortcut Alt-6). Selects the `Tagged Values` tab for the currently selected artifact.

Warning

The `Tagged Values` tab will be selected even if it is not available for the current artifact, in which case it will contain arbitrary data. The side effects of altering this data are unknown.

- `Checklist` (shortcut Alt-7). Selects the `Checklist` tab for the currently selected artifact.

Caution

The `Checklist` tab will be selected even if it is not available for the current artifact, in which case it will be completely blank.

- `History` (shortcut Alt-8). Selects the `History` tab for the currently selected artifact.

Warning

The current version of ArgoUML has no `History` tab, so this entry is always grayed out.

12.5.6. Adjust Grid

This cycles the grid representation on the screen through the following sequence:

- dots at 16 pixel spacing (the default);

- dots at 32 pixel spacing;

- no grid of any form;

- full grid at 16 pixel spacing; and

- full grid at 8 pixel spacing.

Warning

A known problem with the current version of ArgoUML means that altering the grid may cause problems with the display, particularly if no grid is selected.

12.5.7. Adjust Grid Snap

This cycles the spacing of grid snapping through the following sequence:

- snap at 8 pixel spacing (the default);

- snap at 16 pixel spacing;

- snap at 32 pixel spacing; and

- snap at 4 pixel spacing.

Note

There is no option to turn of snap to grid altogether

Caution

In the current version of ArgoUML this is not actually a true snap to grid. It just defines the increments by which artifacts are moved around. Thus if an item has an X coordinate of 4 and Grid Snap is set to 8, moving will take it to X coordinates of 12, 20, 28 etc, *notnot* 8, 16, 24 etc

If you wish to align on true grid snap boundaries, you can use the `Arrange > Align To Grid` menu (see Section 12.7.1).

12.5.8. Adjust Page Breaks

This toggles whether page breaks are shown on the diagram (as white dotted lines).

Caution

This is incorrectly named and implemented in the current version of ArgoUML, since there is no mechanism to alter the page breaks. It should be a tick box toggle labeled `Page Breaks Visible`

12.5.9. Buttons on Selection

If this menu option is enabled, certain artifacts on diagrams will display graphic handles when selected that can be dragged to create connections of various types (associations, generalizations) to other artifacts.

Caution

The menu entry is somewhat misleading labeled. It is better thought of as `Handles On Selection`.

12.5.10. The Notation Sub-Menu

This sub-menu presents a radio box with two options.

Note

This feature is not completely handled throughout ArgoUML, primarily because the multi-notation support is still rather new. See also Section 12.4.10.4).

In the future, this feature will define the user's default notation. At some point ArgoUML will allow defining a class in a diagram as being represented using a specific notation.

For example a class `Able` which is represented in Java, and class `Baker` which is represented in C++, with class `Able` using class `Baker` and a class `Charlie` which references `Able`, but does not have a specific notation defined for the diagram element.

In this situation, `Charlie` would be represented in the default notation, while `Able` and `Baker` would be represented as defined in the diagram element. Changing the default notation would cause `Charlie`'s notation to change but not `Able` or `Baker`.

- `Uml.1.3`. Uses UML notation as the default notation for classes.

- `Java`. Uses Java notation as the default notation for classes.

12.6. The Create Diagram Menu

This menu provides for creating the various types of UML diagrams supported by ArgoUML.

12.6.1. Class Diagram

This menu entry creates a blank class diagram, and selects that diagram in the editing pane. If a package is currently selected, the class diagram will be created within that package, it will be shown within the package on the navigation pane hierarchy (under Package-centric view) and artifacts created on the diagram will have the namespace of the package.

Tip

This does not prevent artifacts from other namespaces/packages appearing on the diagram. They can added from the navigation pane using `Add to Diagram` from the button 2 pop-up menu.

12.6.2. Use Case Diagram

This menu entry creates a blank use case diagram, and selects that diagram in the editing pane. If a package is currently selected, the use case diagram will be created within that package, it will be shown within the package on the navigation pane hierarchy (under Package-centric view) and artifacts created on the diagram will have the namespace of the package.

Tip

This does not prevent artifacts from other namespaces/packages appearing on the diagram. They can added from the navigation pane using `Add to Diagram` from the button 2 pop-up menu.

12.6.3. State Diagram

This menu entry creates a blank state diagram associated with the currently selected class, and selects that diagram in the editing pane. Artifacts created on the diagram will have the namespace of the owning class.

Caution

State diagrams *mustmust* be associated with a class. In the current version of ArgoUML the `State Diagram` menu entry is not grayed out if a class is not selected. However invoking the menu entry will have no effect.

12.6.4. Activity Diagram

This menu entry creates a blank activity diagram associated with the currently selected class, and selects that diagram in the editing pane. Artifacts created on the diagram will have the namespace of the owning class.

Caution

Activity diagrams *mustmust* be associated with a class. In the current version of ArgoUML the `Activity Diagram` menu entry is not grayed out if a class is not selected. However invoking the menu entry will have no effect.

12.6.5. Collaboration Diagram

This menu entry creates a blank collaboration diagram, and selects that. Within the package hierarchy, collaboration diagrams are always created within a `Collaboration`, within the package or model, rather than within the package or model directly itself. If a package is current selected, the collaboration diagram will be created within a collaboration within that package, it will be shown within the collaboration within the package on the navigation pane hierarchy (under Package-centric view) and artifacts created on the diagram will have the namespace of the collaboration within the package.

Tip

This does not prevent artifacts from other namespaces/packages appearing on the diagram. They can added from the navigation pane using `Add to Diagram` from the button 2 pop-up menu.

12.6.6. Deployment Diagram

This menu entry creates a blank deployment diagram, and selects that diagram in the editing pane. If a package is current selected, the deployment diagram will be created within that package, it will be shown within the package on the navigation pane hierarchy (under Package-centric view) and artifacts created on the diagram will have the namespace of the package.

Tip

This does not prevent artifacts from other namespaces/packages appearing on the diagram. They can added from the navigation pane using `Add to Diagram` from the button 2 pop-up menu.

12.6.7. Sequence Diagram

This menu entry creates a blank sequence diagram, and selects that diagram in the editing pane. If a package is current selected, the sequence diagram will be created within that package, it will be shown within the package on the navigation pane hierarchy (under Package-centric view) and artifacts created on the diagram will have the namespace of the package.

Tip

This does not prevent artifacts from other namespaces/packages appearing on the diagram. They can added from the navigation pane using `Add to Diagram` from the button 2 pop-up menu.

12.7. The Arrange Menu

This menu provides a range of functions to help in the alignment of artifacts on diagrams within the editing pane. In general the menu function invoked is applied to any artifact or artifacts currently selected in the editing pane.

12.7.1. The Align Sub-Menu

This sub-menu aligns the selected items. There are seven alignment options provided.

- `Align Tops`. Aligns the selected artifacts by their top edges.

- `Align Bottoms`. Aligns the selected artifacts by their bottom edges.

- `Align Lefts`. Aligns the selected artifacts by their left edges.

- `Align Rights`. Aligns the selected artifacts by their right edges.

- `Align Horizontal Centers`. Aligns the selected artifacts so their horizontal centers are in a vertical line.

- `Align Vertical Centers`. Aligns the selected artifacts so their vertical centers are in a horizontal line.

- `Align To Grid`. Aligns the selected artifacts so their top and right edges are on the grid snap boundary (see Section 12.5.7) edge.

  #### Caution

  The alignment is to the current *grid snapgrid snap* setting, which may be smaller, larger or the same as the displayed grid. Since items are aligned to the grid snap boundary any way when you place them, this menu entry has no effect unless you have either changed the grid snap to a larger value or used one of the other `Arrange` menu entries to push items off their initial positions.

12.7.2. The Distribute Sub-Menu

This sub-menu distributes the selected items. There are four distribution options provided.

- `Distribute Horizontal Spacing`. The leftmost and rightmost selected artifacts are not moved. The others are adjusted horizontally until the horizontal space (i.e. from the right edge of the left artifact to the left edge of the right artifact) is the same for all of the selected items

- `Distribute Horizontal Centers`. The leftmost and rightmost selected artifacts are not moved. The others are adjusted horizontally until the distance between the horizontal centers of all the selected items is the same.

- `Distribute Vertical Spacing`. The top and bottom selected artifacts are not moved. The others are adjusted vertically until the vertical space (i.e. from the bottom edge of the top artifact to the top edge of the bottom artifact) is the same for all of the selected items

- ▤ Distribute Vertical Centers. The top and bottom selected artifacts are not moved. The others are adjusted vertically until the distance between the vertical centers of all the selected items is the same.

12.7.3. The Reorder Sub-Menu

This sub-menu adjusts the ordering of overlapping items. There are four reorder options provided.

- ▣ To Back. The selected artifacts are moved to the back of any other artifacts they overlap.

- ▣ To Front. The selected artifacts are moved to the front of any other artifacts they overlap.

- ▣ Backward. The selected artifacts are moved one step back in the ordering hierarchy with respect to other artifacts they overlap.

- ▣ Forward. The selected artifacts are moved one step forward in the ordering hierarchy with respect to other artifacts they overlap.

12.7.4. The Nudge Sub-Menu

This sub-menu provides fine adjustment to the positioning of selected artifacts. There are four nudge options provided.

- ▣ Nudge Left. The selected artifacts are moved one pixel to the left.

- ▣ Nudge Right. The selected artifacts are moved one pixel to the right.

- ▣ Nudge Up. The selected artifacts are moved one pixel to the up.

- ▣ Nudge Down. The selected artifacts are moved one pixel to the down.

12.7.5. The Layout Sub-Menu

This provides control of layout of items on class diagrams during reverse engineering (see Section 12.3.5.1). It provides two options.

- Automatic. Automated layout.

- Incremental. Incremental layout.

    #### Caution

    For all except class diagrams, the two menu entries are grayed out and unavailable (since reverse engineering only creates a class diagram). For class diagrams only the Automatic option is available. So the utility of this menu entry is somewhat limited.

12.8. The Generation Menu

This menu provides support for code generation from UML diagrams. The functionality is built around the structural information of class diagrams.

Note

The current version of ArgoUML supports only code generation of Java. Future work will extend this to other languages.

Caution

This is still very much work in progress. The current version of ArgoUML will generate a structural template for your code, but is not able to handle behavioral specifications to generate code for the dynamic behavior of the model.

12.8.1. Generate Selected Classes

This menu entry brings up a dialog box for the ArgoUML code generator (see Figure 12.6).

Figure 12.6. The dialog for Generate Selected Classes.
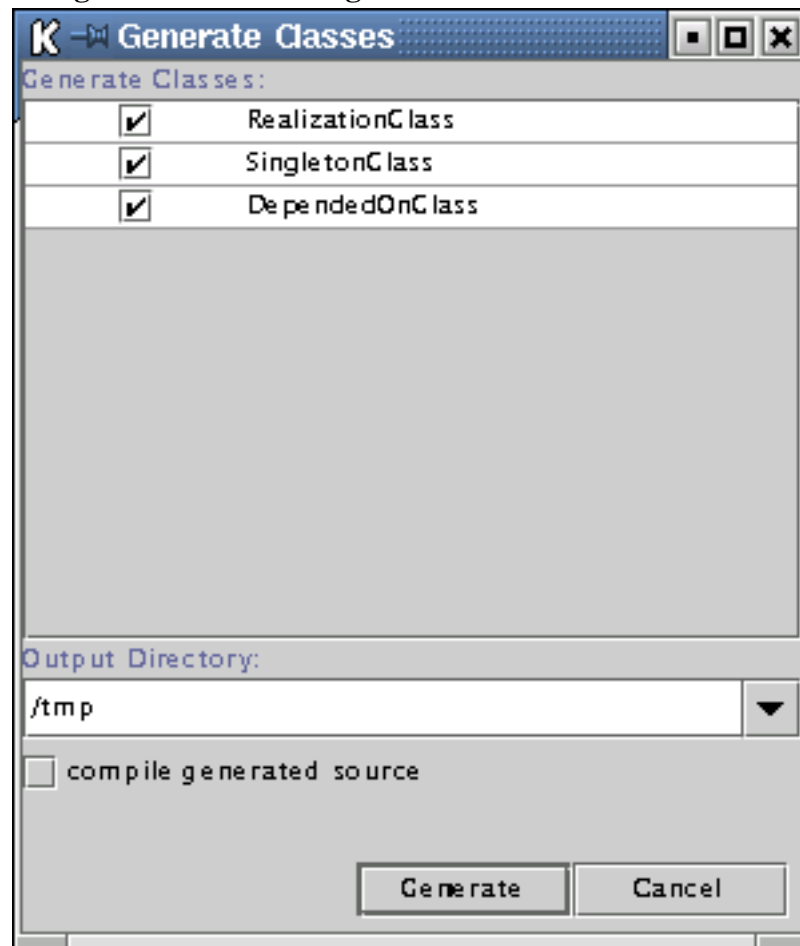
Below a label `Generate Classes:` the dialog box lists each of the selected classes by name with a check box to the left, which is initially checked. Unchecking any of these boxes will omit code generation for that class.

In the lower portion of the dialog box is a text box labeled `Output Directory:` to specify the directory in which code is generated. Within this directory, a top level directory will be created with the name of the model. Further sub-directories will be created to reflect the package/namespace hierarchy of the model. A drop down selector gives access to previously selected output directories.

Beneath this a check box labeled `compile generated source` (clear by default) controls whether the generated code should be compiled.

Finally, at the bottom of the dialog box are two buttons, labeled `Generate` and `Cancel`. Button 1 on the former will cause the code to be generated, button 1 on the latter will cancel code generation.

Note

Strictly speaking this menu entry should be labeled `Generate Selected Classes…`, since it invokes a dialog box.

12.8.2. Generate All Classes

Shortcut F7.

This function behaves as `Generate Selected Classes` (see Section 12.8.1) would with all classes in the model selected.

12.9. The Critique Menu

This menu controls one of ArgoUML's unique features—the use of critics to guide the designer. The theory behind this is well described in Jason Robbins' PhD dissertation http://argouml.tigris.org/docs/robbins_dissertation/ [http:/-/-argouml.tigris.org/-docs/-robbins_dissertation/-].

Note

A word about terminology. The *criticscritics* are background processes, which evaluate the current model according to various "good" design criteria. There is one critic for every design criterion.

The output of a critic is a *critiquecritique*—a statement about some aspect of the model that does not appear to follow good design practice.

Finally a critique will generally suggest how the bad design issue it has identified can be rectified, by raising a *to-do itemto-do item*.

Note

The critics run as asynchronous processes that run in parallel with the main ArgoUML tool. Changes typically take a second or two to propagate as the critics wake up.

12.9.1. Toggle Auto-Critique

This is a check box, controlling whether the critics are running. By default it is checked. If unchecked, then all critics are disabled, and any to-do items generated by critics (the only others being those the designer has added by hand) are hidden in the to-do pane.

12.9.2. Design Issues…

This menu entry brings up a dialog box controlling how critics associated with a particular design areas are to be handled (see Figure 12.7).

Figure 12.7. The dialog for Design Issues….
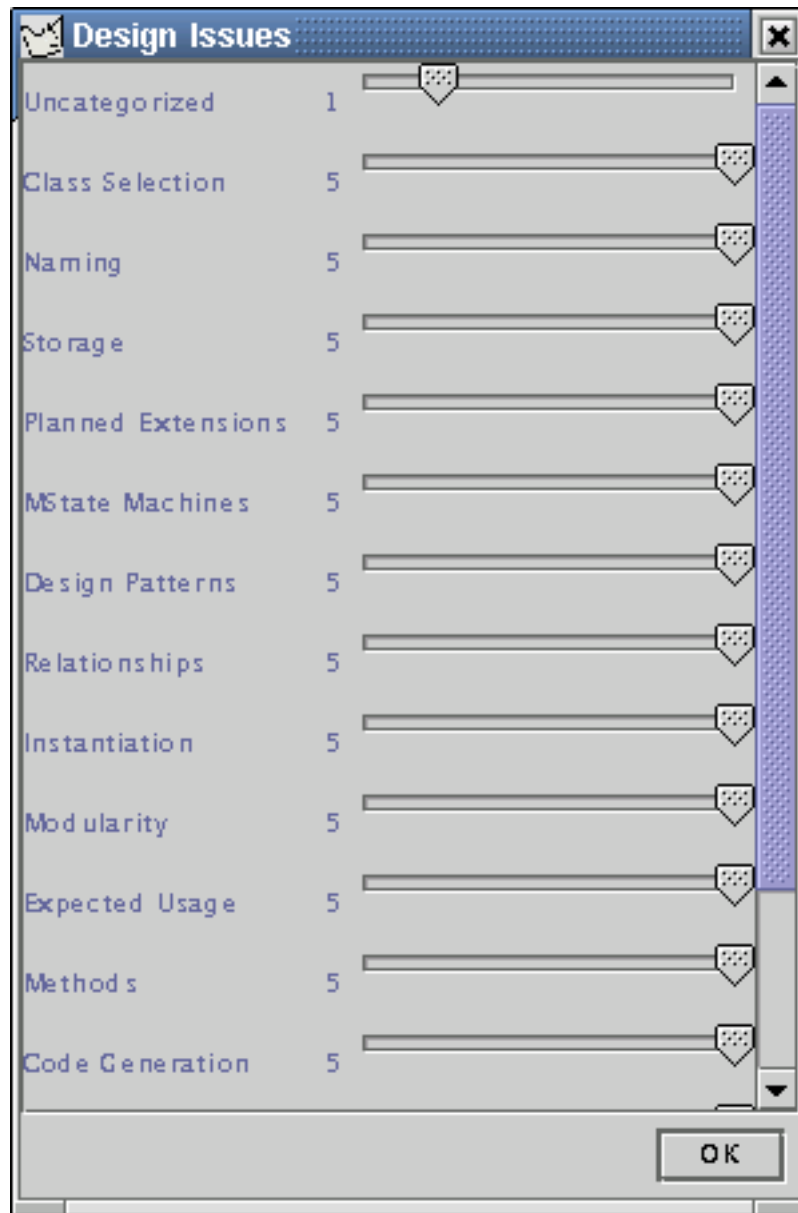
ArgoUML categorizes critics according the the design issue they address. At present there are 16 such categories. The critics in each category are discussed in detail in the chapter on critics (Chapter 13).

- `Uncategorized`. Unused (see Section 13.2.

- `Class Selection`. Three critics (see Section 13.3.

- `Naming`. Nineteen critics (see Section 13.4.

- `Storage`. Four critics (see Section 13.5.

- `Planned Extensions`. Three critics (see Section 13.6.

- `MState Machines`. Fourteen critics (see Section 13.7.

- `Design Patterns`. Sixteen critics (see Section 13.8.

- `Relationships`. Seven critics (see Section 13.9.

- `Instantiation`. Unused (see Section 13.10.

- `Modularity`. Two critics (see Section 13.11.

- `Expected Usage`.One critic (see Section 13.12.

- `Methods`. Four critics (see Section 13.13.

- `Code Generation`. One critic (see Section 13.14.

- `Stereotypes`. Unused (see Section 13.15.

- `Inheritance`. Ten critics (see Section 13.16.

- `Containment`. Six critics (see Section 13.17.

The sliders may be set for each category to control the critics that trigger for that category. Setting a slider to zero will disable all critics in that category, and remove all associated to-do items from the to-do pane.

Setting a slider to a higher value will enable all critics at or above that priority level within the design issue category (1 being the highest priority and 5 the lowest.

Note

The sliders are set by default to 5 for all design categories except `Uncategorized` which is set to 1.

Caution

In the current release of ArgoUML all critics trigger at priority 1, so any higher values of the slider make no difference.

12.9.3. Design Goals…

This menu entry brings up a dialog box controlling how design goals are to be handled (see Figure 12.8).

Figure 12.8. The dialog for Design Goals….
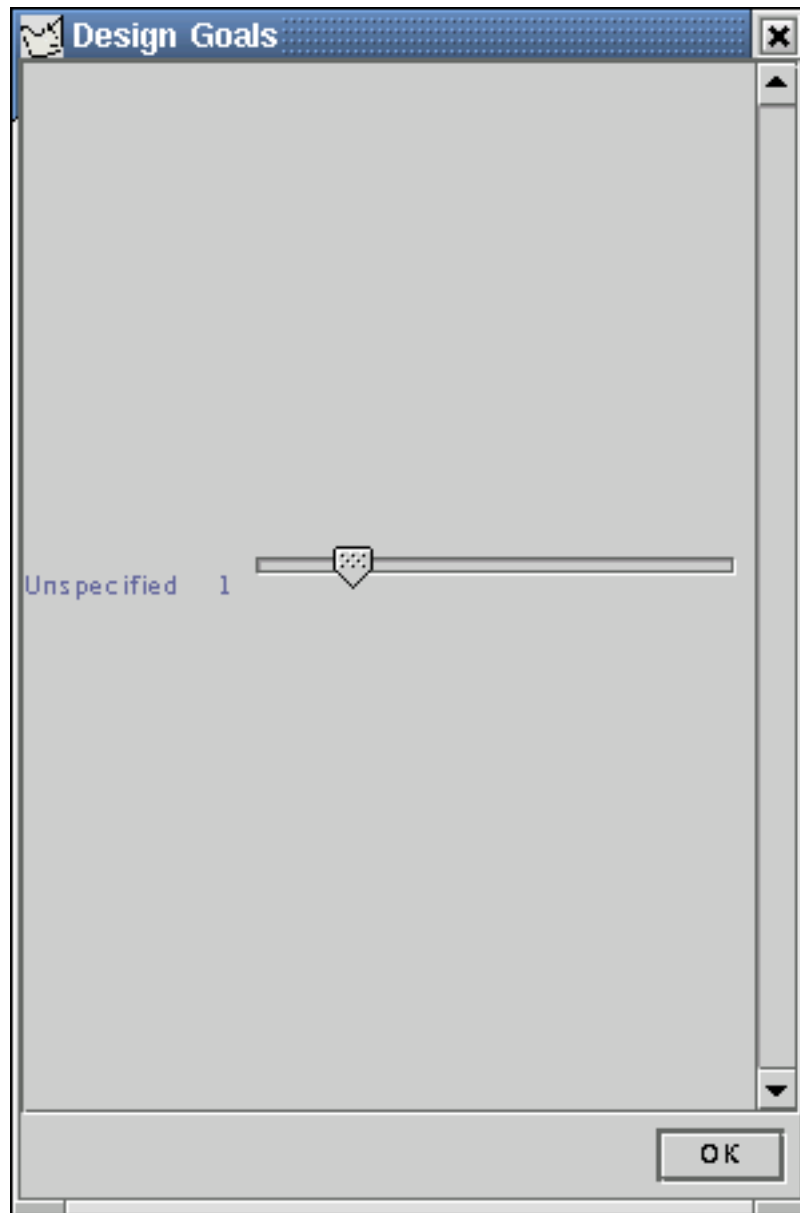
ArgoUML has the concept that the designer will have a number of design goals to be achieved (for example good structural representation, detailed behavioral representation etc). Critics are associated with one or more goals.

This dialog allows the user to specify the priority of each design goal.

The sliders may be set for each design goal to control the critics that trigger for that goal. Setting a slider to zero will disable all critics in that goal, and remove all associated to-do items from the to-do pane.

Setting a slider to a higher value will enable all critics at or above that priority level within the design issue category (1 being the highest priority and 5 the lowest.

Tip

It may be useful to think of this function as very similar to Design Issues… (see Section 12.9.2, but with grouping of critics according to the outcomes of OOA&D rather than grouping according to the structure of UML.

Caution

The current version of ArgoUML provides a single design goal, `Unspecified`, with its slider set by default to priority 1. However it contains no critics and so has no effect.

12.9.4. Browse Critics…

This menu entry brings up a dialog box controlling the individual critics (see Figure 12.9).

Figure 12.9. The dialog for Browse Critics….
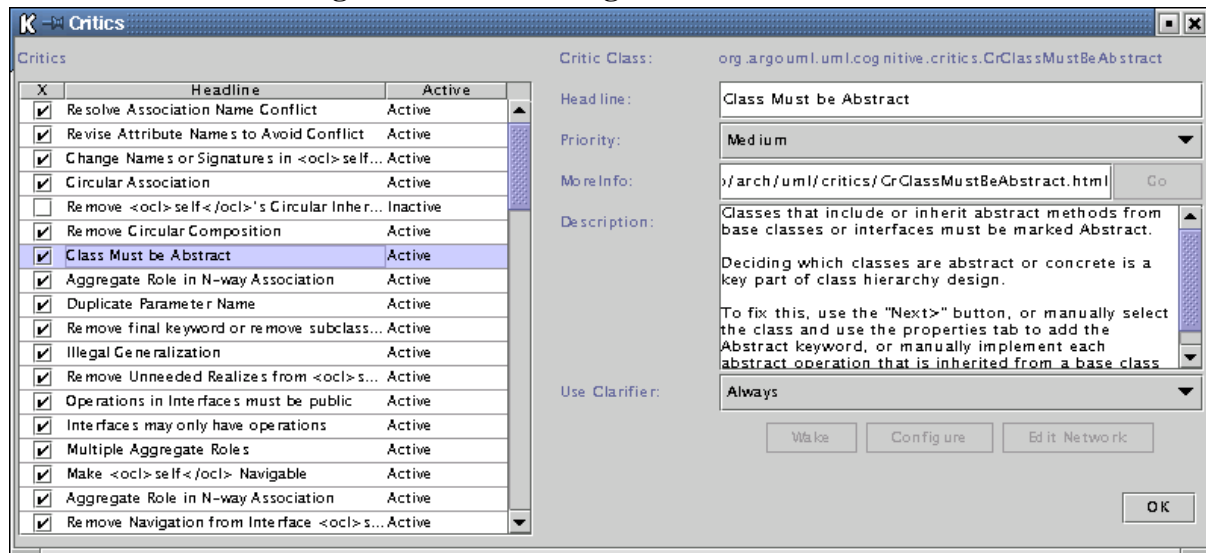
This dialog controls the behavior of individual critics. To the left is a list of all the critics. For each critic there are three columns, labeled X, `Headline` and `Active`. The first of these is a check box, which may be toggled with button 1 click. The second is the headline name of the critic, the third contains the word `Active` or `In-active`. A critic is inactive if the box in the first column is unchecked *oror* the box is checked, but the critic has been snoozed from the to-do pane (see Chapter 11

Any critic for which the box in the first column is unchecked is inactive and will not trigger. In addition any to-do items associated with that critic will be removed from the to-do pane.

Note

In the current version of ArgoUML all critics but one are active by default. The one exception is `Re-move <class>'s Circular Inheritance`.

The current version of ArgoUML has a total of 78 critics. They are described in detail by design issue category in the chapter on critics (see Chapter 13).

- `Resolve Association Name Conflict`. See Section 13.4.1

- `Revise Attribute Names to Avoid Conflict`. See Section 13.4.2

- `Change Names or Signatures in <artifact>`. See Section 13.4.3

- `Circular Association`. See Section 13.9.1

- `Remove <artifact>'s Circular Inheritance`. See Section 13.16.2

- `Remove Circular Composition`. See Section 13.17.1

- `Class Must be Abstract`. See Section 13.13.2

- Classifier not in Namespace of its Association. See Section 13.11.1

- Duplicate Parameter Name. See Section 13.17.2

- Duplicate end (role) names for <Association>. See Section 13.4.4

- Remove final keyword or remove subclasses. See Section 13.16.4

- Illegal Generalization . See Section 13.16.5

- Remove Unneeded Realizes from <class>. See Section 13.16.6

- Operations in Interfaces must be public. See Section 13.6.1

- Interfaces may only have operations. See Section 13.6.2

- Two aggregate ends (roles) in binary association. See Section 13.17.3

- Make <association> Navigable. See Section 13.9.2

- Aggregate end (role) in 3-way (or more) Association. See Section 13.17.4

- Remove Navigation from Interface <association>. See Section 13.9.3

- Role name conflicts with member. See Section 13.4.5

- Choose a Name (classes and interfaces). See Section 13.4.6

- Chose a Unique Name for <artifact> (classes and interfaces). See Section 13.4.7

- Choose a name (attributes). See Section 13.4.8

- Choose a name (operations). See Section 13.4.9

- Choose a Name (states). See Section 13.4.10

- Add Instance Variables to <class>. See Section 13.5.2

- Add Associations to <artifact>. See Section 13.9.4

- Add Operations to <artifact>. See Section 13.13.3

- Add Constructor to <class>. See Section 13.5.3

- Add Elements to Package <package>. See Section 13.11.2

- Wrap DataType. See Section 13.3.1

- Import Parameter Type into Class. See Section 13.17.6

- Define Concrete (Sub)Class. See Section 13.16.7

- Define Class to Implement <interface>. See Section 13.16.8

- Choose a Unique Name for <state> (states). See Section 13.4.11

- Revise Name to Avoid Confusion. See Section 13.4.12

- Remove Reference to Specific Subclass. See Section 13.6.3

- Choose a Legal Name. See Section 13.4.13

- Change <artifact> to a Non-Reserved Word. See Section 13.4.14

- Change Multiple Inheritance to interfaces. See Section 13.14.1

- `Reduce Associations on <artifact>`. See Section 13.9.6

- `Reduce Attributes on <class>`. See Section 13.5.4

- `Reduce Operations on <artifact>`. See Section 13.13.4

- `Reduce Transitions on <state>`. See Section 13.7.1

- `Reduce States in machine <diagram>`. See Section 13.7.2

- `Reduce Classes in diagram <diagram>`. See Section 13.3.2

- `Add Transitions to <state>`. See Section 13.7.3

- `Add Incoming Transitions to <state>`. See Section 13.7.4

- `Add Outgoing Transitions from <state>`. See Section 13.7.5

- `Remove Extra Initial States`. See Section 13.7.6

- `Place an Initial State`. See Section 13.7.7

- `Add Trigger or Guard to Transition`. See Section 13.7.8

- `Change Join Transitions`. See Section 13.7.9

- `Change Fork Transitions`. See Section 13.7.10

- `Change Branch Transitions`. See Section 13.7.11

- `Add Guard to Transition`. See Section 13.7.12

- `Choose a Better Operation Name`. See Section 13.4.15

- `Choose a Better Attribute Name`. See Section 13.4.16

- `Capitalise Class Name <class>`. See Section 13.4.17

- `Revise Package Name <package>`. See Section 13.4.18

- `Consider using Singleton Pattern for <class>`. See Section 13.8.1

- `Singleton Stereotype Violated in <class>`. See Section 13.8.2

- `Nodes normally have no enclosers`. See Section 13.8.3

- `NodeInstances normally have no enclosers`. See Section 13.8.4

- `Components normally are inside nodes`. See Section 13.8.5

- `ComponentInstances normally are inside nodes`. See Section 13.8.6

- `Classes normally are inside components`. See Section 13.8.7

- `Interfaces normally are inside components`. See Section 13.8.8

- `Objects normally are inside components`. See Section 13.8.9

- `LinkEnds have not the same locations`. See Section 13.8.10

- `Set classifier` (deployment diagram). See Section 13.8.11

- `Missing return-actions`. See Section 13.8.12

- `Missing call(send)-action`. See Section 13.8.13

- `No Stimuli on these links`. See Section 13.8.14

- `Set classifier` (sequence diagram). See Section 13.8.15

- `Wrong position of these stimuli`. See Section 13.8.16

- `Clean Up Diagram`. See Section 13.3.3

- `Make Edge More Visible`. See Section 13.7.14

To the right of the list are a series of fields giving detailed control over individual critics. Selecting a critic in the list on the left will populate the fields for that critic.

Warning

In the current release of ArgoUML there is a known problem with the user interface. When moving down through the list (i.e. selecting a critic below the previously selected one) the fields are populated for the previously selected critic. To get the fields for the desired critic, another critic further down must be selected, making the desired critic the "previous critic" whose details will then populate the fields on the right.

Selecting items further up the list (i.e. selecting a critic *aboveabove* the previously selected on) does not have this problem.

A consequence of the different behavior when selecting up or down is that the last critic on the list (`Make Edge More Visible`) can never populate the fields on the right.

The fields on the right are titled `Critic Class:` and then the full name of the class in ArgoUML that implements the critic.

The first field below this title is a text box labeled `Headline:` giving the complete headline of the critic (which may be truncated in the list on the left).

Note

In the headline you may see the text `<ocl>self</ocl>`, which will be replaced by the name of the artifact in question when the critic is triggered.

The next field is a drop-down selector, labeled `Priority:`. The three options available are `High`, `Medium` and `Low` and specify the priority category of any to-do item generated by this critic.

The next field is labeled `MoreInfo:` and contains a URL pointing to further information with a button to the right labeled `Go` to take you to that URL.

Caution

In the current release of ArgoUML there is no further information available, and the `Go` button is always grayed out and disabled.

The next field is labeled `Description:` and is a text area with a detailed explanation of what this critic means. If the text is too large for the area a scroll bar is provided to the right.

Note

In this text area you may see the text `<ocl>self</ocl>`, which will be replaced by the name of the artifact in question when the critic is triggered.

The last field is a drop-down selector labeled `Use Clarifier`, with three options, `Always`, `If Only One` and `Never`.

Clarifiers are the icons and wavy red underlines drawn on the actual diagrams to indicate the a critic has triggered. The original intention was to make the mapping from critics to clarifiers to somewhat customizable.

For example one user might make a `Missing Name` critic show a red underline, another user might turn off the clarifier, or have it draw a wavy green underline or a blue questionmark. Critics with their clarifier's disabled would still produce feedback that is listed in the to-do pane.

Caution

In the current release of ArgoUML this selector has no function, beyond printing a message on the console. It is for future development.

Underneath the fields are three buttons in a horizontal row.

- `Wake`. It is possible to snooze a critic from the to-do pane (see Chapter 11), which makes the critic inactive for a period. If the critic has been snoozed, this button is enabled and will wake the critic back up again. Otherwise it is grayed out.

  ### Tip

  You can tell a snoozed critic, because in the list on the left it will have the box in the first column checked, but will be shown as inactive in the third column.

- `Configure`. This button is for configuring the critic.

  ### Caution

  In the current version of ArgoUML this function is not implemented, and this button is always grayed out. It is for future development.

- `Edit Network`. Right now critics are implemented in java code. That means end-users cannot add new critics.

  The idea of a critic network is that they would be a state machine like diagram with several steps. Each step would have condition that together defined the "rule" that the critic is providing. If the rule fires, then remaining steps would define the steps of the wizard to help the user fix the problem.

  The ideas behind this are discussed in Chapter 4 of Jason Robbins PhD disseration (http://argouml.tigris.org/docs/robbins_dissertation/diss4.html [http:/-/-argouml.tigris.org/-docs/-robbins_dissertation/-diss4.html]. In particular look at Figure 1-6 in this chapter and the related discussion.

  A suggested implementation is that the conditions could be written in OCL against the UML meta-model. A library of predefined conditins and steps would allow end-users to build new critics by combining those in novel ways.

  ### Caution

  In the current version of ArgoUML this function is not implemented, and this button is always grayed out. It is for future development.

Finally the bottom right of the dialog contains a button labeled `OK`. Button 1 click here dismisses the dialog.

12.10. The Tools Menu

This menu provides a generic menu attachment point for any plug-ins provided with ArgoUML. The standard system has no plug-ins, so this menu entry is grayed out by default.

12.11. The Help Menu

This menu provides help on the use of ArgoUML. At present it has a single entry.

12.11.1. About Argo/UML

This menu entry brings up the help window for ArgoUML (see Figure 12.10).

Figure 12.10. The help window for ArgoUML
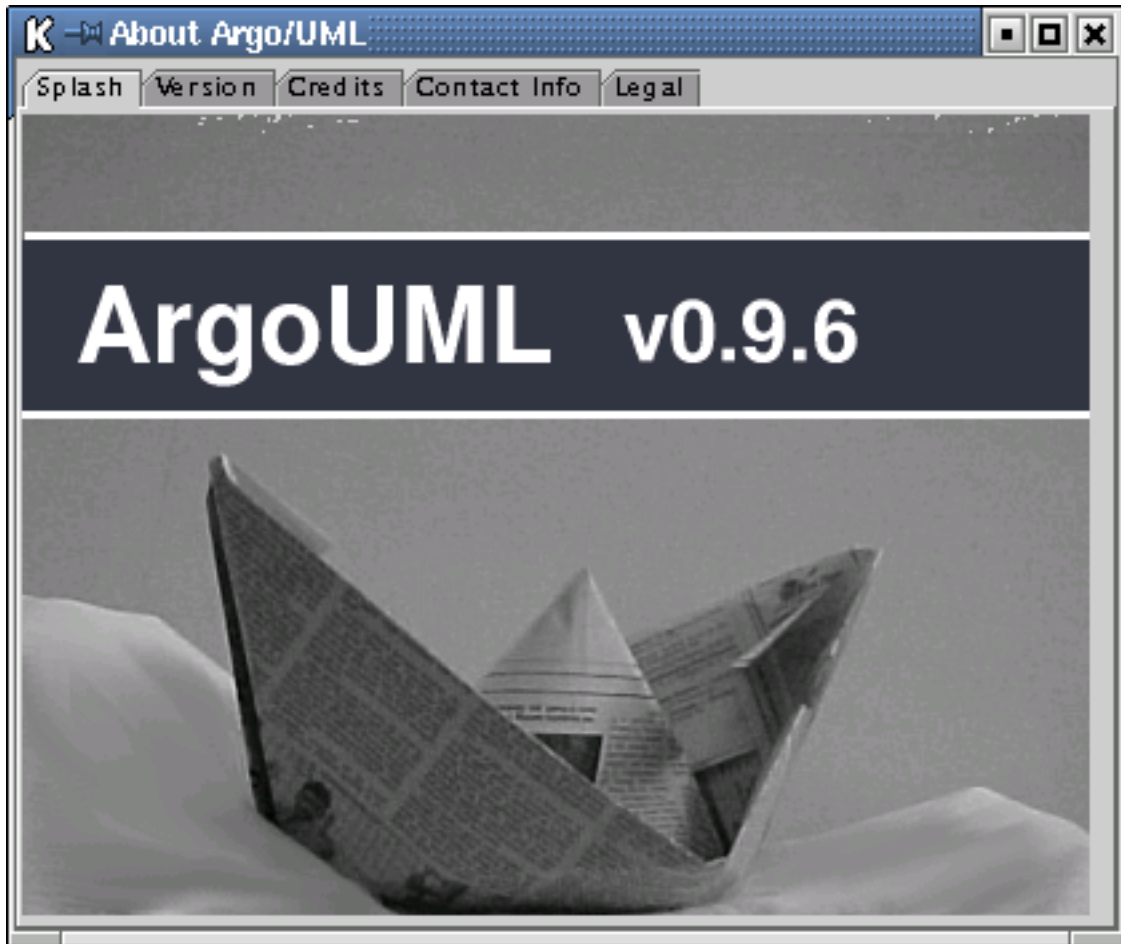
The window has five tabs, which are selected by button 1 click. By default the first tab (Splash) is shown.

- Splash. This displays the picture shown when ArgoUML starts up.

- Version. This provides version information on the various packages that make up ArgoUML.

- Credits. This details all those who have created ArgoUML, including contact details for the various module owners.

- Contact Info. This gives the major contact points for the ArgoUML project—the web site, Jason Robbins, the developers mailing list.

- Legal. A statement of the FreeBSD license which covers all the ArgoUML software.

Caution

The various documentation of the project are not all covered by FreeBSD (which is really meant for software). In particular this manual is covered by the OpenPub license (see Appendix F).

Chapter 13. The Critics

Table of Contents

13.1. Introduction

The key feature that distinguishes ArgoUML from other UML CASE tools is its use of concepts from cognitive psychology. The theory behind this is well described in Jason Robbins' PhD dissertation http://argouml.tigris.org/docs/robbins_dissertation/ [http://-/-argouml.tigris.org/-docs/-robbins_dissertation/-].

Critics are one of the main ways in which these ideas are implemented. Running in the background they offer advice to the designer which may be accepted or ignored. A key point is that they do not *imposeimpose* a deci-

sion on the designer.

Note

The critics are asynchronous processes that run in parallel with the main ArgoUML tool. Changes typically take a second or two to propagate as the critics wake up.

13.1.1. Terminology

The *criticscritics* are background processes, which evaluate the current model according to various "good" design criteria. There is one critic for every design criterion.

The output of a critic is a *critiquecritique*—a statement about some aspect of the model that does not appear to follow good design practice.

Finally a critique will generally suggest how the bad design issue it has identified can be rectified, by raising a *to-do itemto-do item*.

13.1.2. Design Issues

ArgoUML categorizes critics according the the design issue they address (some critics may be in more than one category). At present there are 16 such categories.

Within this manual the descriptions of critics are grouped in sections by design issue.

13.2. Uncategorized

These are critics that do not fit into any other category.

The current version of ArgoUML has no critics in this category.

13.3. Class Selection

These are critics concerning how classes are chosen and used.

The current version of ArgoUML has three critics in this category.

13.3.1. Wrap DataType

DataTypes are not full classes within UML 1.3. They can only have enumeration literals as values, and only support `query` operations (that is operations that do not change the DataType's state).

DataTypes cannot be associated with classes, unless the DataType is part of a composite (black diamond) aggregation. Such an association relects the tight binding of a collection of DataType instances to a class instance. In effect such a DataType is an attribute of the class with mutliplicity.

Good OOA&D depends on careful choices about which entities to represent as full objects and which to represent as attributes of objects.

There are two options to fix this problem.

- Replace the DataType with a full class.

- or change the association aggregation to composite relationship at the DataType end.

13.3.2. Reduce Classes in diagram <diagram>

Suggestion to improve readability by having fewer classes on a diagram.

13.3.3. Clean Up Diagram

Suggestion that the diagram could be improved by moving artifacts that are overlapping.

13.4. Naming

These are critics concerning the naming of artifacts. The current version of ArgoUML has 18 critics in this category.

13.4.1. Resolve Association Name Conflict

Suggestion that two association names in the same namespace have the same name. This is not permitted in UML.

13.4.2. Revise Attribute Names to Avoid Conflict

Suggestion that two attribute names of a class have the same name. This is not permitted in UML.

Note

This may be caused by inheritance of an attribute through a generalization relationship.

13.4.3. Change Names or Signatures in <artifact>

Two operations in <artifact> have the same signature. This means their name is the same, and the list of parameters has the same type.

Where there are conflicting signatures, correct code cannot be generated for mainstream OO languages. It also leads to very unclear semantics of the design.

In comparing signatures, this critic considers:

1. the name;

2. the list of in, out and in-out parameter types *in orderin order*; and

Only if these all match in both type and order, will the signatures be considered as the same.

This follows the line of Java/C++ in ignoring the return parameters for the signature. This *maymay* be unsatisfactory for some functional OO languages.

Note

Some purists would argue that the comparison should really differentiate between in, out and in-out parameters. However no practical programming language can do this when resolving an overloaded method invocation, so this critics lumps them all together.

13.4.4. Duplicate End (Role) Names for <Association:>

The specified association has two (or more) ends (roles) with the same name. One of the well-formedness rules in UML 1.3 for associations, is that all end (role) names must be unique.

This ensures that there can be unambiguous reference to the ends of the association.

To fix this, manually select the association and change the names of one or more of the offending ends (roles) using the button 2 pop-up menu or the property sheet.

13.4.5. Role name conflicts with member

A suggestions that good design avoids role names for associations that clash with attributes or operations of the source class. Roles may be realized in the code as attributes or operations, causing code generation problems.

13.4.6. Choose a Name (Classes and Interfaces)

The class or interface concerned has been given no name (it will appear in the model as `anon`). Suggestion that good design requires that all interfaces and classes are named.

13.4.7. Chose a Unique Name for <artifact> (Classes and Interfaces)

Suggestion that the class or interface specified has the same name as another (in the namespace), which is bad design and will prevent valid code generation.

13.4.8. Choose a Name (Attributes)

The attribute concerned has been given no name (it will appear in the model as `anon`). Suggestion that good design requires that all attributes are named.

13.4.9. Choose a Name (Operations)

The operation concerned has been given no name (it will appear in the model as `anon`). Suggestion that good design requires that all operations are named.

13.4.10. Choose a Name (States)

The state concerned has been given no name (it will appear in the model as `anon`). Suggestion that good design requires that all states are named.

13.4.11. Choose a Unique Name for <artifact> (States)

Suggestion that the state specified has the same name as another (in the current state diagram), which is bad design and will prevent valid code generation.

13.4.12. Revise Name to Avoid Confusion

Two names in the same namespace have very similar names (differing only by one character). Suggestion this could potentially lead to confusion.

Caution

This critic can be particularly annoying, since at times it is useful and good design to have a series of artifacts `var1`, `var2` etc.

It is important to remember that critics offer guidance, and are not always correct. ArgoUML lets you dismiss the resulting to-do items through the to-do pane (see Chapter 11).

13.4.13. Choose a Legal Name

All artifact names in ArgoUML must use only letters, digits and underscore characters. This critic suggests an entity has not met this requirement.

13.4.14. Change <artifact> to a Non-Reserved Word

Suggestion that this artifact's name is the same as a reserved word in UML (or within one character of one), which is not permitted.

13.4.15. Choose a Better Operation Name

Suggestion that an operation has not followed the naming convention that operation names begin with lower case letters.

Caution

Following the Java and C++ convention most designers give their constructors the same name as the class, which begins with an upper case character. In the current release of ArgoUML, this will trigger this critic, unless the constructor is stereotyped «create».

It is important to remember that critics offer guidance, and are not always correct. ArgoUML lets you dismiss the resulting to-do items through the to-do pane (see Chapter 11).

13.4.16. Choose a Better Attribute Name

Suggestion that an attribute has not followed the naming convention that attribute names begin with lower case letters.

13.4.17. Capitalize Class Name <class>

Suggestion that a class has not followed the naming convention that classes begin with upper case letters.

Note

Although not triggering this critic, the same convention should apply to interfaces.

13.4.18. Revise Package Name <artifact>

Suggestion that a package has not followed the naming convention of using lower case letters with periods used to indicated sub-packages.

13.5. Storage

Critics concerning attributes of classes.

The current version of ArgoUML has four critics in this category.

13.5.1. Revise Attribute Names to Avoid Conflict

This critic is discussed under an earlier design issues category (see Section 13.4.2).

13.5.2. Add Instance Variables to <class>

Suggestion that no instance variables have been specified for the given class. Such classes may be created to specify static attributes and methods, but by convention should be given the stereotype «utility».

Warning

In the current version of ArgoUML this critic triggers for all classes whether or not they have attributes, so should be treated with caution.

It is important to remember that critics offer guidance, and are not always correct. ArgoUML lets you dismiss the resulting to-do items through the to-do pane (see Chapter 11).

13.5.3. Add Constructor to <class>

You have not yet defined a constructor for class *classclass*. Constructors initialize new instances such that their attributes have valid values. This class probably needs a constructor because not all of its attributes have initial values.

Defining good constructors is key to establishing class invariants, and class invariants are a powerful aid in writing solid code.

To fix this, add a constructor manually by clicking on *classclass* in the navigator pane and adding an operation using the context sensitive pop-up menu in the property tab, or select *classclass* where it appears on a class diagram and use the Add Operation tool.

In the UML 1.3 standard, a constructor is an operation with the stereotype «create». Although not strictly standard, ArgoUML will also accept «Create» as a stereotype for constructors.

By convention in Java and C++ a constructor has the same name as the class, is not static, and returns no value. ArgoUML will also accept any operation that follows these conventions as a constructor even if it is not stereotyped «create».

Caution

Operators are created in ArgoUML with a default return parameter (named return). You will need to remove this parameter to meet the Java/C++ convention.

13.5.4. Reduce Attributes on <class>

Suggestion that the class has too many attributes for a good design, and is at risk of becoming a design bottleneck.

13.6. Planned Extensions

Critics concerning interfaces and subclasses.

Note

It is not clear why this category has the name "Planned Extensions".

The current version of ArgoUML has three critics in this category.

13.6.1. Operations in Interfaces must be public

Suggestion that there is no point in having non-public operations in Interfaces, since they must be visible to be realized by a class.

13.6.2. Interfaces may only have operations

Suggestion that an interfaces has attributes defined. The UML standard defines interfaces to have operations.

Caution

The current version of ArgoUML no longer allows you to add attributes to intefaces, so this should never occur in ArgoUML model. It might trigger if a project has been loaded with XMI created by another tool.

13.6.3. Remove Reference to Specific Subclass

Suggestion that in a good design, a class should not reference its subclasses directly through attributes, operations or associations.

13.7. MState Machines

Critics concerning state machines.

Note

The correct title for this category should be `State Machines`.

The current version of ArgoUML has 14 critics in this category.

13.7.1. Reduce Transitions on <state>

Suggestion given state is involved in so many transitions it may be a maintenance bottleneck.

13.7.2. Reduce States in machine <machine>

Suggestion that the given state machine has so many states as to be confusing and should be simplified (perhaps by breaking into several machines, or using a hierarchy).

13.7.3. Add Transitions to <state>

Suggestion that the given state requires both incoming and outgoing transitions.

13.7.4. Add Incoming Transitions to <artifact>

Suggestion that the given state requires incoming transitions.

13.7.5. Add Outgoing Transitions from <artifact>

Suggestion that the given state requires outgoing transitions.

13.7.6. Remove Extra Initial States

Suggestion that there is more than one initial state in the state machine or composite state, which is not permitted in UML.

13.7.7. Place an Initial State

Suggestion that there is no initial state in the state machine or composite state.

13.7.8. Add Trigger or Guard to Transition

Suggestion that a transition is missing either a trigger or guard, one at least of which is required for it to be taken.

13.7.9. Change Join Transitions

Suggestion that the join pseudostate has an invalid number of transitions. Normally there should be one outgoing and two or more incoming.

13.7.10. Change Fork Transitions

Suggestion that the fork pseudostate has an invalid number of transitions. Normally there should be one incoming and two or more outgoing.

13.7.11. Change Branch Transitions

Suggestion that the branch pseudostate has an invalid number of transitions. Normally there should be one incoming and two or more outgoing.

13.7.12. Add Guard to Transition

Suggestion that the transition requires a guard.

Caution

It is not clear that this is a valid critic. It is perfectly acceptable to have a transition without a guard—the transition is always taken when the trigger is invoked.

13.7.13. Clean Up Diagram

This critic is discussed under an earlier design issues category (see Section 13.3.3).

13.7.14. Make Edge More Visible

Suggestion that an edge artifact such as an association or abstraction is so short it may be missed. Move the connected artifacts apart to make the edge more visible.

13.8. Design Patterns

Critics concerning design pattern usage in ArgoUML.

These relate to the use of patterns as described by the so called "Gang of Four". ArgoUML also uses this category for critics associated with deployment and sequence diagrams. The current version of ArgoUML has 16 critics in this category.

13.8.1. Consider using Singleton Pattern for <class>

classclass has no non-static attributes nor any associations that are navigable away from instances of this class. This means that every instance of this class will be identical to every other instance, since there will be nothing about the instances that can differentiate them.

Under these circumstances you should consider making explicit that you have exactly one instance of this class, by using the singleton Pattern. Using the singleton pattern can save time and memory space. Within ArgoUML this can be done by using the «singleton» stereotype on this class.

If it is not your intent to have a single instance, you should define instance variables (i.e. non-static attributes) and/or outgoing associations that will represent differences bewteen instances.

Having specified *classclass* as a singleton, you need to define the class so there can only be a single instance. This will complete the information representation part of your design. To achieve this you need to do the following.

1. You must define a static attribute (a class variable) holding the instance. This must therefore have *classclass* as its type.

2. You must have only private constructors so that new instances cannot be made by other code. The creation of the single instance could be through a suitable helper operation, which invokes this private constructor just once.

3. You must have at least one constructor to override the default constructor, so that the default constructor is not used to create multiple instances.

For the definition of a constructor under the UML 1.3 standard, and extensions to that definition accepted by ArgoUML see Section 13.5.3.

13.8.2. Singleton Stereotype Violated in <class>

This class is marked with the «singleton» stereotype, but it does not satisfy the constraints imposed on singletons (ArgoUML will also accept «Singleton» stereotype as defining a singleton). A singleton class can have at most one instance. This means that the class must meet the design criteria for a singleton (see Section 13.8.1).

Whenever you mark a class with a stereotype, the class should satisfy all constraints of the stereotype. This is an important part of making a self-consistent and understangle design. Using the singleton pattern can save time and memory space.

If you no longer want this class to be a singleton, remove the «singleton» stereotype by clicking on the class and selecting the blank selection on the stereotype drop-down within the properties tab.

To apply the singleton pattern you should follow the directions in Section 13.8.1.

13.8.3. Nodes normally have no enclosers

A suggestion that nodes should not be drawn inside other artifacts on the deployment diagram, since they repre-

sent an autonomous physical object.

13.8.4. NodeInstances normally have no enclosers

A suggestion that node instances should not be drawn inside other artifacts on the deployment diagram, since they represent an autonomous physical object.

13.8.5. Components normally are inside nodes

A suggestion that components represent the logical entities within physical nodes, and so should be drawn within a node, where nodes are shown on the deployment diagram.

13.8.6. ComponentInstances normally are inside nodes

A suggestion that component instances represent the logical entities within physical nodes, and so should be drawn within a node instance, where node instances are shown on the deployment diagram.

13.8.7. Classes normally are inside components

A suggestion that classes, as artifacts making up components, should be drawn within components on the deployment diagram.

13.8.8. Interfaces normally are inside components

A suggestion that interfaces, as artifacts making up components, should be drawn within components on the deployment diagram.

13.8.9. Objects normally are inside components

A suggestion that objects, as instances of artifacts making up components, should be drawn within components or component instances on the deployment diagram.

13.8.10. LinkEnds have not the same locations

A suggestion that a link (e.g. association) connecting objects on a deployment diagram has one end in a component and the other in a component instance (since objects can be in either). This makes no sense.

13.8.11. Set classifier (Deployment Diagram)

Suggestion that there is an instance (object) without an associated classifier (class, datatype) on a deployment diagram.

13.8.12. Missing return-actions

Suggestion that a sequence diagram has a send or call action without a corresponding return action.

13.8.13. Missing call(send)-action

Suggestion that a sequence diagram has a return action, but no preceding call or send action.

13.8.14. No Stimuli on these links

Suggestion that a sequence diagram has a link connecting objects without an associated stimulus (without which the link is meaningless).

Warning

Triggering this critic indicates a serious problem, since ArgoUML provides no mechanism for creating a link without a stimulus. It probably indicates that the diagram was created by loading a corrupt project, with an XMI file describing a link without a stimulus, possibly created by a tool other than ArgoUML.

13.8.15. Set Classifier (Sequence Diagram)

Suggestion that there is an object without an associated classifier (class, datatype) on a sequence diagram.

13.8.16. Wrong position of these stimuli

Suggestion that the initiation of send/call-return message exchanges in a sequence diagram does not properly initiate from left to right.

13.9. Relationships

Critics concerning associations in ArgoUML.

The current version of ArgoUML has seven critics in this category.

13.9.1. Circular Association

Suggestion that an association class has roles that refer back directly to itself, which is not permitted.

Warning

This critic is meaningless in the current version of ArgoUML which does not support association classes. In addition the current code would never trigger, even if an association class were created.

13.9.2. Make <association> Navigable

Suggestion that the association referred to is not navigable in either direction. This is permitted in the UML standard, but has no obvious meaning in any practical design.

13.9.3. Remove Navigation from Interface via <association>

Associations involving an interface can be not be navigable in the direction from the interface. This is because interfaces contain only operation declarations and cannot hold pointers to other objects.

This part of the design should be changed before you can generate code from this design. If you do generate code before fixing this problem, the code will not match the design.

To fix this, select the association and use the `Properties` tab to select in turn each association end that is *notnot* connected to the interface. Uncheck `Navigable` for each of these ends.

The association should then appear with a stick arrowhead pointed towards the interface

Note

In an ideal world, ArgoUML would not permit associations to be created that were navigable from interfaces.

Warning

The current version of ArgoUML has a known defect where it creates an association between a class and interface that is either navigable in both directions, or navigable only from the interface to the class. The fix is to remove the navigation from the class end of the association and if necessary add navigation to the interface end (navigation refers to navigability *towardstowards* the end) if this critic is triggered.

13.9.4. Add Associations to <artifact>

Suggestion that the specified artifact (actor, use case or class) has no associations connecting it to other artifacts. This is required for the artifact to be useful in a design.

13.9.5. Remove Reference to Specific Subclass

This critic is discussed under an earlier design issues category (see Section 13.6.3).

13.9.6. Reduce Associations on <artifact>

Suggestion that the given artifact (actor, use case, class or interface) has so many associations it may be a maintenance bottleneck.

13.9.7. Make Edge More Visible

This critic is discussed under an earlier design issues category (see Section 13.7.14).

13.10. Instantiation

Critics concerning instantiation of classifiers in ArgoUML.

The current version of ArgoUML has no critics in this category.

13.11. Modularity

Critics concerning modular development in ArgoUML.

The current version of ArgoUML has two critics in this category.

13.11.1. Classifier not in Namespace of its Association

One of the well-formedness rules in UML 1.3 for associations, is that all the classifiers attached to the ends of the association should belong to the same namespace as the association.

If this were not the case, there would be no naming, by which each end could refer to all the others.

This critic is triggered when an association does not meet this criterion. The solution is to delete the association, and recreate it on a diagram, whose namespace includes those of all the attached classifiers.

Caution

In the current implementation of ArgoUML this critic does not handle hierarchical namespaces. As a consequence it will trigger for associations where the immediate namespaces of the attached classifiers is different, even though they are part of the same namespace hierarchy.

13.11.2. Add Elements to Package <package>

Suggestion that the specified package has no content. Good design suggests packages are created to put things in.

Note

This will always trigger when you first create a package, since you cannot create one that is not empty!

13.12. Expected Usage

Critics concerning generally accepted good practice in ArgoUML.

The current version of ArgoUML has one critic in this category.

13.12.1. Clean Up Diagram

This critic is discussed under an earlier design issues category (see Section 13.3.3).

13.13. Methods

Critics concerning operations in ArgoUML.

The current version of ArgoUML has four critics in this category.

13.13.1. Change Names or Signatures in <artifact>

This critic is discussed under an earlier design issues category (see Section 13.4.3).

13.13.2. Class Must be Abstract

Suggestion that a class that inherits or defines abstract operations must be marked abstract.

13.13.3. Add Operations to <class>

Suggestion that the specified class has no operations defined. This is required for the class to be useful in a design.

13.13.4. Reduce Operations on <artifact>

Suggestion that the artifact (class or interface) has too many operations for a good design, and is at risk of becoming a design bottleneck.

13.14. Code Generation

Critics concerning code generation in ArgoUML.

The current version of ArgoUML has one critic in this category.

13.14.1. Change Multiple Inheritance to interfaces

Suggestion that a class has multiple generalizations, which is permitted by UML, but cannot be generated into Java code, because Java does not support multiple inheritance.

13.15. Stereotypes

Critics concerning stereotypes in ArgoUML.

The current version of ArgoUML has no critics in this category.

13.16. Inheritance

Critics concerning generalization and specialization in ArgoUML.

The current version of ArgoUML has ten critics in this category.

13.16.1. Revise Attribute Names to Avoid Conflict

This critic is discussed under an earlier design issues category (see Section 13.4.2).

13.16.2. Remove <class>'s Circular Inheritance

Suggestion that a class inherits from itself, through a chain of generalizations, which is not permitted.

Caution

This critic is marked inactive by default in the current release of ArgoUML (the only one so marked). It will not trigger unless made active.

13.16.3. Class Must be Abstract

This critic is discussed under an earlier design issues category (see Section 13.13.2).

13.16.4. Remove final keyword or remove subclasses

Suggestion that a class that is final has specializations, which is not permitted in UML.

13.16.5. Illegal Generalization

Suggestion that there is a generalization between artifacts of different UML metaclasses, which is not permitted.

Caution

It is not clear that such a generalization can be created within ArgoUML. It probably indicates that the diagram was created by loading a corrupt project, with an XMI file describing such a generalization, possibly created by a tool other than ArgoUML.

13.16.6. Remove Unneeded Realizes from <class>

Suggestion that the specified class has a realization relationship both directly and indirectly to the same interface (by realization from two interfaces, one of which is a generalization of the other for example). Good design deprecates such duplication.

13.16.7. Define Concrete (Sub)Class

Suggestion that a class is abstract with no concrete subclasses, and so can never be realized.

13.16.8. Define Class to Implement <interface>

Suggestion that the interface referred to has no influence on the running system, since it is never implemented by a class.

13.16.9. Change Multiple Inheritance to interfaces

This critic is discussed under an earlier design issues category (see Section 13.14.1).

13.16.10. Make Edge More Visible

This critic is discussed under an earlier design issues category (see Section 13.7.14).

13.17. Containment

Critics concerning containment in ArgoUML, that is where one artifact forms a component part of another.

The current version of ArgoUML has six critics in this category.

13.17.1. Remove Circular Composition

Suggestion that there is a series of composition relationships (associations with black diamonds) that form a cycle, which is not permitted.

13.17.2. Duplicate Parameter Name

Suggestion that a parameter list to an operation or event has two or more parameters with the same name, which is not permitted.

13.17.3. Two Aggregate Ends (Roles) in Binary Association

Only one end (role) of a binary association can be aggregate or composite. This a well-formedness rule of the UML 1.3 standard.

Aggregation and composition are used to indicate whole-part relationships, and by definition, the "part" end cannot be aggregate.

To fix this, identify the "part" end of the association, and use the critic wizard (the `Next>` button, or manually set its aggregation to `none` using the button 2 pop-up menu or the property sheet.

Composition (more correctly called composite aggregation) is used where there is a whole-part relationship that is one-to-one or one-to-many, and the lifetime of the part is inextricably tied to the lifetime of the whole. Instances of the whole will have responsibility for creating and destroying instances of the associated part. This also means that a class can only be a part in one composite aggregation.

An example of a composite aggregation might be a database of cars and their wheels. This is a one-to-four relationship, and the database entry for a wheel is associated with its car. When the car ceases to exist in the database, so do its wheels.

Aggregation (more correctly called shared aggregation) is used where there is a whole-part relationship, that does not meet the criteria for a composite aggregation. An example might be a database of university courses and the students that attend them. There is a whole-part relationship between courses and students. However there is no lifetime relationship between students and course (a student continues to exist even after a course is finished) and the relationship is many-to-many.

13.17.4. Aggregate End (Role) in 3-way (or More) Association

Three-way (or more) associations can not have aggregate ends (roles). This a well-formedness rule of the UML 1.3 standard.

Aggregation and composition are used to indicate whole-part relationships, and by definition can only apply to binary associations between artifacts.

To fix this, manually select the association, and set the aggregation of each of its ends (roles) to `none` using the button 2 pop-up menu or the property sheet.

13.17.5. Wrap DataType

This critic is discussed under an earlier design issues category (see Section 13.3.1).

13.17.6. Import Parameter Type into Class

Suggestion that the type of a parameter used within an operation of a class is not visible.

Warning

It is not clear how this can ever be caused within ArgoUML. In any case this critic can never trigger in the current implementation of ArgoUML.

Chapter 14. ArgoUML Top Level Artifact Reference

Table of Contents

14.1. Introduction

This chapter describes each artifact that can be created within ArgoUML. The chapter covers top-level "general" artifacts. The following chapters (see Chapter 15 through Chapter 21) cover each of the ArgoUML diagrams.

There is a close relationship between this material and the properties tab of the details pane (see Section 10.4). That section covers properties in general, in this chapter they are linked to specific artifacts.

14.2. The Model

The model is the top level artifact within ArgoUML. In the UML meta-model it is a sub-class of package. In many respects within ArgoUML it behaves similarly to a package (see Section 16.2).

Note

ArgoUML is restricted to one model within the tool.

Caution

In principle you can add a model to a class diagram using the context sensitive (button 2) pop up menu `Add To Diagram` from the navigation pane. However this is very demanding of memory and may cause a problem with ArgoUML running out of memory.

Where standard data types, classes and packages are loaded (the default, see Chapter 22), these are sub-packages of the model.

Note

In the current version of ArgoUML these sub-packages are not initially visible from the model property pane as owned packages. However if you use the data type navigator from a suitable pane, you can navigate up to the model, and thereafter that data type's sub-packages will appear.

14.2.1. Model Details Tabs

The details tabs that are active for the model are as follows.

Note

The `Style` tab is grayed out, since the model typically has no representation on a diagram (see the caution above).

ToDoItem	Standard tab.
Properties	See Section 14.2.2 and Section 14.2.3 below.
Documentation	Standard tab.
Source	Standard tab. This contains a template package declaration appropriate to the entire model. In the future this would be part of the code generation activity.
Constraints	Standard tab. There are no standard constraints defined for `Model` within the UML meta-model.
Tagged Values	Standard tab. In the UML meta-model, `Model` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the class is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived models still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

14.2.2. Model Property Toolbar

Go up	Navigate up through the package structure of the model.
⇦Go back	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨Go forward	Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

| | Add class | This creates a new class (see Section 16.5) within the model (which appears on no diagram), navigating immediately to the properties tab for that class. |

Tip

Whilst it can make sense to create classes this way, it is usually a lot clearer to create them within diagrams where you want them.

| | Add interface | This creates a new interface (see Section 16.14) within the model (which appears on no diagram), navigating immediately to the properties tab for that interface. |

Tip

Whilst it can make sense to create interfaces this way, it is usually a lot clearer to create them within diagrams where you want them.

| | Add datatype | This creates a new data type (see Section 14.3) within the model (which appears on no diagram), navigating immediately to the properties tab for that datatype. |

| | Add stereotype | This creates a new stereotype (see Section 14.4) within the model, navigating immediately to the properties tab for that stereotype. |

| | Add actor | This creates a new actor (see Section 15.2) within the model (which appears on no diagram), navigating immediately to the properties tab for that actor. |

Tip

Whilst it can make sense to create actors this way, it is usually a lot clearer to create them within diagrams where you want them.

| | Add use case | This creates a new use case (see Section 15.3) within the model (which appears on no diagram), navigating immediately to the properties tab for that use case. |

Tip

Whilst it can make sense to create use cases this way, it is usually a lot clearer to create them within diagrams where you want them.

| | Add subpackage | This creates a new sub-package (see Section 16.2) within the model (which appears on no diagram), navigating immediately to the properties tab for that package. |

Tip

Whilst it can make sense to create sub-packages of the model this way, it is usually a lot clearer to create them within diagrams where you want them.

| | Add generalization | This creates a new generalization (see Section 15.6 and Section 16.13) within the model (which appears on no diagram), navigating immediately to the properties tab for that generalization. |

Tip

Whilst it can make sense to create generalizations this way, it is usually a lot clearer to create them within diagrams where you want them.

⬆	Add realization	This should create a new realization (i.e. an abstraction with stereotype `realize`—see Section 16.15) within the model (which appears on no diagram).

Warning

This appears to be broken in the current release of ArgoUML. Even if it did work, it would be of limited value, since the abstraction property sheet has no mechanism for adding supplier and client artifacts.

✗	Delete	This has no effect—it is meaningless to delete the model!

14.2.3. Property Fields For The Model

`Name` Text box. The name of the model. The name of a model, like all packages, is by convention all lower case.

Note

The default name supplied to a new model by ArgoUML, `untitledModel`, is thus erroneous and guarantees that ArgoUML always starts up with at least one problem being reported by the design critics.

`Stereotype` Drop down selector. Model is provided by default with the UML standard stereotypes for model (`systemModel` and `metamodel`) and package (`facade`, `framework`, `stub`), as well as the stereotypes for models and packages in the UML standard profiles for *software development processsoftware development process* (`use-case model`, `analysis model`, `design model`, `implementation model`, `use case package` and `analysis service package`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`.

Note

A number of standard stereotypes are currently missing from ArgoUML: `topLevel` (from package), `use case system`, `analysis system`, `analysis package` (all three from the *software development processsoftware development process* profile).

Caution

ArgoUML provides the stereotype `realize` for models and packages. This appears to be an error, since this stereotype properly belongs to the `Association` metaclass.

Stereotyping models is a useful thing, although it is of limited value in ArgoUML where you have only a single model.

`Navigate to stereotype` If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

`Namespace` Text box. Records the namespace for the model. This is the package hierarchy. However since the model is at the top of the hierarchy in ArgoUML, this box is always empty.

`Modifiers` Check box, with entries `Abstract`, `Final` and `Root`.

- `abstract` is used to declare that this model cannot be instantiated, but must always be specialized.

Tip

The meaning of `abstract` applied to a model is not that clear. It might mean that the model contains

interfaces or abstract classes without realizations. Since ArgoUML only permits one model, this is not a meaningful box to check.

- `final` indicates that this model can have no further subpackages, while `root` indicates it is the top level model.

Tip

Within ArgoUML `root` only meaningfully applies to the Model, since all packages sit within the model. In the absence of the `topLevel` stereotype, this could be used to emphasize that the Model is at the top level.

`Specialization` Text box. Lists any specialized model (i.e. for which this model is a generalization.

Caution

Since there is only one model in ArgoUML there is no sensible specialization of generalization that could be created.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the model (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

`Generalization` Text area. Lists any model that *generalizesgeneralizes* this model.

Caution

Since there is only one model in ArgoUML there is no sensible specialization of generalization that could be created.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the

diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the model (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

`Implements` Text area. Lists any models for which this model is the realization. Of little value within ArgoUML, since multiple models are not permitted.

It is possible to add realizations from this menu, although since there is no way to add a model to the other end of the abstraction, it is meaningless. Button 1 click allows you to navigate to them and button 2 click gives a pop up menu with options to `Open`, `Add`, `Delete`, `Move Up` and `Move Down` (see Generalizations for examples of how these operations are used).

`Owned Elements` Text area A hierarchical listing of the packages, classes, interfaces, datatypes, actors, use cases, associations, generalizations and stereotypes within the model. For each of these types of artifact, where there is at least one example in the model, a folder icon is shown.

Button 1 double click on the folder icon ( ) opens the folder to reveal the artifacts owned under that category.

Button 1 click on any of the revealed artifacts yields navigates to that artifact.

Button 1 double click on an open folder (with artifacts displayed) will close the folder and hide the artifacts.

Alternatively, Button 2 double click on any item gives a pop up menu.

- `Add`. This option creates a new artifact of the specified type within the model, and navigates to that item.

Note

The item is placed under its relevant folder, irrespective of where the pop up menu is revealed. So button 2 lick on `Actor` folder and selection of `Add/Package` will place the new package under the `Package` folder, not the `Actor` folder.

This is the only menu option actually available to the model (the others apply to packages within the model, see Section 16.2).

Tip

It is often far easier to add artifacts by drawing them on the relevant diagram in the editing pane.

- `Delete`. This is always grayed out for a model.

- `Go`. This is always grayed out for a model.

- `Open`. This is always grayed out for a model.

14.3. Datatype

Datatypes can be thought of as simple classes. They have no attributes, and any operations on them must have no side-effects. A useful analogy is primitive datatypes in a language like Java. The integer "3" stands on its own—it has no inner structure. There are operations (for example addition) on the integers, but when I perform 3 + 4 the result is a new number, "3" and "4" are unchanged by the exercise.

Within UML 1.3, `DataType` is a sub-class of the `Classifier` metaclass. It embraces the predefined primitive types (`byte`, `char`, `double`, `float`, `int`, `long` and `short`), the predefined enumeration, `boolean` and user defined *enumeration typesenumeration types*.

Note

`void` is implemented as a class rather than a datatype within ArgoUML

Within ArgoUML new datatypes may be created using the `New datatype` button on the property tabs of the model and packages (in which case the new datatype is restricted in scope to the package), as well as the properties tab for datatype..

You can navigate to existing property types via the `NavClass` button on any property sheet that has a Type drop-down (for example attribute or parameter).

Caution

UML 1.3 allows user defined datatypes to be placed on class diagrams. This is not permitted in ArgoUML.

14.3.1. Datatype Details Tabs

The details tabs that are active for datatypes are as follows.

`ToDoItem`	Standard tab.
`Properties`	See Section 14.3.2 and Section 14.3.3 below.
`Documentation`	Standard tab.
`Source`	Standard tab, but with no effect. In a future version of ArgoUML the `Attributes` and `Operations` tick boxes will control the display of literals and operations defined for the datatype.
`Source`	Standard tab. Unused. One would expect a class declaration for the new datatype to support code generation.
`Constraints`	Standard tab. There are no standard constraints defined for `Datatype` (or `Enumeration` within the UML metamodel.
`Tagged Values`	Standard tab. In the UML metamodel, `Datatype` has the following standard tagged values defined.

- `persistence` (from the superclass, `Classifier`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

Tip

Since user defined datatypes are enumerations, they have no state to preserve, and the value of this tagged value is irrelevant.

- `semantics` (from the superclass, `Classifier`). The value is a specification of the semantics of the datatype.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the class is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Tip

Whilst formally available, a derived datatype does not have an obvious value, and so datatypes should always be marked with `derived=false`.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

14.3.2. Datatype Property Toolbar

	`Go up`	Navigate up through the package structure.
⇦	`Go back`	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨	`Go forward`	Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

DT	`New datatype`	This creates a new datatype (see Section 16.5) within the same package as the current datatype.

Tip

Whilst it can make sense to create datatypes this way, it can be clearer to create them within the package or model where you want them.

▤	`Add enumeration literal`	This creates a new enumeration literal within the datatype, navigating immediately to the properties tab for that literal.

Caution

ArgoUML does not actually have a separate concept of a literal. The navigation will take you to the property sheet for an attribute of a class (see Section 16.6). When defining a literal, all that matters is its name. Other parts of the attribute property sheet should be ignored.

✘ Delete   This deletes the datatype from the model.

14.3.3. Property Fields For Datatype

Name   Text box. The name of the datatype. The primitive datatypes all have lower case names, but there is
            no formal convention.

Note

The default name supplied to a new package by ArgoUML is the empty string "". The datatype will ap-
pear with the name anon in the navigation pane.

Stereotype Drop down selector. Package is provided by default with the UML standard stereotypes for classifier
            (metaclass, powertype, process, thread and utility). ArgoUML also provides the addi-
            tional stereotypes, machine, organization and person as well as the abstraction stereotype
            realize.

Most important of all, ArgoUML provides the stereotype enumeration for use with user defined
enumeration data types.

Tip

enumeration should always be used as the stereotype of any created datatypes.

Caution

Use of other stereotypes (particularly realize is unwise. If you really need to stereotype user defined
types, you should consider creating your own stereotype derived from enumeration.

Navigate to stereotype Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see
                        Section 14.4).

Namespace   Text box. Records the namespace for the datatype. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the
top level namespace).

Modifiers   Check box, with entries public, abstract, final and root.

- public has identical meaning to the public declaration in Java. It indicates whether this
  datatype is restricted to use in its own package (the default) or can be used by anyone.

- abstract is used to declare that this datatype cannot be instantiated, but must always be special-
  ized.

Caution

ArgoUML provides no mechanism for specializing datatypes, so this check box is of little use.

- final indicates that this datatype can have no further sub-types, while root indicates it is a top
  level datatype.

Tip

In the absence of specialization of datatypes within ArgoUML these have little value. In effect all
datatypes are both root and final

Generalizations Text area. Lists any datatype that *generalizes* this datatype.

Caution

It is not clear that generalizing datatypes makes much sense.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the datatype (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

`SpecialText boxLists` any specialized datatype (i.e. for which this datatype is a generalization.

Caution

It is not clear that specializing datatypes makes much sense.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the datatype (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specializa-

tion selected is not at the top. It is moved one up the list.

- • `Move Down`. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

`Implements` Text box. Lists any abstract classes or interfaces realized by this datatype. By default shows the entry `none`.

Button 1 click takes you to any entry (but see the warning below), button 2 gives a pop up menu with five entries.

- • `Open`. Where an abstraction dependency exists, navigate to it (but see warning below).

- • `Add`. Add a new abstraction dependency.

Note

ArgoUML does not navigate to the new abstraction, and this field shows the name [empty], whatever name you choose to give the abstraction.

Tip

Whilst adding abstractions is possible, only the supplier end is created, and there is no way to supply the client end. The value of such abstractions is therefore minimal.

- • `Delete`. Where an abstraction has been added this removes it completely from the model.

- • `Move Up`. Only available where there are two or more abstractions, and the abstraction selected is not at the top. It is moved one up the list.

- • `Move Down`. Only available where there are two or more abstractions listed, and the abstraction selected is not at the bottom. It is moved one down the list.

Tip

In principle a datatype could realize some other artifact, but it would be unusual. Best avoided.

Warning

Button 1 click on the default entry navigates to the most recently selected to-do item in the to-do pane. This seems to be a bug in ArgoUML.

`Operations` Text area. Lists all the operations defined on this datatype. Button 1 click takes you to the selected operation, button 2 click brings up a pop up menu with five entries.

- • `Open`. Navigate to the selected operation.

- • `Add`. Add a new operation and navigate to it.

- • `Delete`. Removes the selected operation from the model.

- • `Move Up`. Only available where there are two or more operations, and the operation selected is not at the top. It is moved one up the list.

- • `Move Down`. Only available where there are two or more operations listed, and the operation selected is not at the bottom. It is moved one down the list.

See Section 16.7 for details of operations.

Caution

ArgoUML treats all operations as equivalent. Any operations created here will use the same mechanism as operations for classes. Remember that operations on datatypes must have no side effects (they are read-only). This means the `query` modifier *mustmust* be checked for all operations.

Literal Text area. Lists all the literals defined for this datatype. Button 1 click takes you to the selected literal, button 2 click brings up a pop up menu with five entries.

- `Open`. Navigate to the selected literal.

- `Add`. Add a new literal and navigate to it.

- `Delete`. Removes the selected literal from the model.

- `Move Up`. Only available where there are two or more literals, and the literal selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more literals listed, and the literal selected is not at the bottom. It is moved one down the list.

Caution

ArgoUML does not actually have a separate concept of a literal. The navigations listed above will all take you to the property sheet for an attribute of a class (see Section 16.6). When defining a literal, all that matters is its name. Other parts of the attribute property sheet should be ignored.

14.4. Stereotypes

Stereotypes are the main extension mechanism of UML, providing a way to derive specializations of the standard metaclasses. `Stereotype` is a sub-class of `GeneralizableElement` in the UML metamodel. Stereotypes are supplemented by *constraintsconstraints* and *tagged valuestagged values*.

New stereotypes are added from the model property tab (see Section 14.2, the package property tab (see Section 16.2) or the stereotype property tab itself (see below). Properties of existing stereotypes can be reached by selecting the property tab for any artifact with that stereotype and using the navstereo button (⌐) within the property tab.

Note

Due to a bug in ArgoUML newly created stereotypes do not appear in stereotype drop down menus. Where the stereotypes apply to a class, it is sufficient to select a new class and then select the original class back. However this does not work for other artifacts. The solution is to save and reload the model.

14.4.1. Stereotype Details Tabs

The details tabs that are active for stereotypes are as follows.

Note

The `Style` tab is grayed out, since a stereotype has no representation on a diagram.

ToDoItem   Standard tab.

Properties   See Section 14.4.2 and Section 14.4.3 below.

`Documentation` Standard tab.

`Source`   Standard tab. This contains the representation of the stereotype on diagrams (its name be-
                  tween « and »).

Warning

You can edit this entry, but it has no effect and when you return to the entry it will be restored to its
original value.

`Constraints`   Standard tab. There are no standard constraints defined for `Package` within the UML
                  metamodel.

Caution

Constraints for a stereotype are rather different to those for elements in the UML core architecture, in
that they apply to all artifacts to which the stereotype is applied, *notnot* just the stereotype itself.

`Tagged Values` Standard tab. In the UML metamodel, `Stereotype` has the following standard tagged val-
                  ues defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the class is
  redundant—it can be formally derived from other elements, or `false` meaning it can-
  not.

Note

This indicates any element with this stereotype has the `derived` tag set accordingly.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element
`documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

Caution

Tagged values for a stereotype are rather different to those for elements in the UML core architecture,
in that they apply to all artifacts to which the stereotype is applied, *notnot* just the stereotype itself.

14.4.2. Stereotype Property Toolbar

`Go up`   Navigate up through the package structure of the model.

`Go back`   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

`Go forward`   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

`Add stereotype`   This creates a new stereotype (see Section 14.4) within the model

161

(which appears on no diagram), navigating immediately to the properties tab for that stereotype.

✕ `Delete`   This deletes the stereotype from the model.

14.4.3. Property Fields For Stereotype

`Name`   Text box. The name of the stereotype. There is no convention for naming stereotypes, beyond starting them with a lower case letter. Even the standard UML stereotypes vary between all lower case (e.g. `metamodel`), bumpy caps (e.g. `systemModel`) and space separated (e.g. `object model`).

Note

ArgoUML does not enforce any naming convention for stereotypes

`Base Class` Drop down selector. Any stereotype must be derived from one of the metaclasses in the UML meta-model `Abstraction`, `Association`, `AssociationEnd`, `BehavioralFeature`, `CallEvent`, `Class`, `Classifier`, `Collaboration`, `Comment`, `Component`, `Constraint`, `DataType`, `Dependency`, `Flow`, `Generalization`, `Model`, `ModelElement`, `ObjectFlowState`, `Package`, `Permission`, `Subsystem` and `Usage`) or the artifact classes that derive from them. The stereotype will then be available to artifacts that derive from that same metaclass or that artifact.

`Namespace` Text box. Records the namespace for the stereotype. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

`Modifiers` Check box, with entries `Abstract`, `Final` and `Root`.

- `abstract` is used to declare that artifacts that use this stereotype cannot be instantiated, but must always be specialized.

- `final` indicates that artifacts that use this stereotype can have no further sub-types, while `root` indicates it is a top level artifact.

Caution

Remember that these modifiers apply to the artifacts using the stereotype, not just the stereotype.

Warning

ArgoUML does neither imposes, nor checks that artifacts using a stereotype adopt the stereotype's modifiers.

`Generalizations` Text area. Lists any stereotype that *generalizes generalizes* this stereotype.

Caution

It is not clear that generalizing stereotypes makes much sense.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the

diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the stereotype (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

`Special Text box on`lists any specialized stereotype (i.e. for which this stereotype is a generalization.

Caution

It is not clear that specializing stereotypes makes much sense.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the stereotype (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

14.5. ArgoUML Diagrams

ArgoUML supports seven types of diagrams: i) use case; ii) class; iii) sequence; iv) state; v) collaboration; vi) activity; and vii) deployment. They are created using the `Create Diagram` drop down menu (see Section 12.6).

Note

State and activity diagrams are associated with a particular class, and can only be created when a class has been selected. In the current version of ArgoUML they are not grayed out even when a class is not selected. However under these circumstances they will have no effect.

14.5.1. Diagram Details Tabs

The details tabs that are active for diagrams are as follows.

ToDoItem   Standard tab.

Properties   See Section 14.5.2 and Section 14.5.3 below.

Source   Holds a reference to the ArgoUML implementation class for that diagram.

Note

The source tab content can be changed, but this has no effect.

14.5.2. Diagram Property Toolbar

⇦Go back   Navigate back to the previous diagram. Always grayed out, since there is no way to navigate to a diagram (and hence no way to navigate back).

⇨Go forward   Navigate forward to the next diagram. Always grayed out, since there is no way to navigate to a diagram (and hence no way to navigate forward).

14.5.3. Property Fields For Diagram

Name   The name of the diagram. There are no conventions for naming diagrams. By default, ArgoUML uses the (space separated) diagram name and a sequence number, thus use case diagram 1.

Chapter 15. ArgoUML Use Case Diagram Artifact Reference

Table of Contents

15.1. Introduction

This chapter describes each artifact that can be created within a use case diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

There is a close relationship between this material and the properties tab of the details pane (see Section 10.4). That section covers properties in general, in this chapter they are linked to specific artifacts.

Figure 15.1 shows a use case diagram with all possible artifacts displayed.

Figure 15.1. Possible artifacts on a use case diagram.
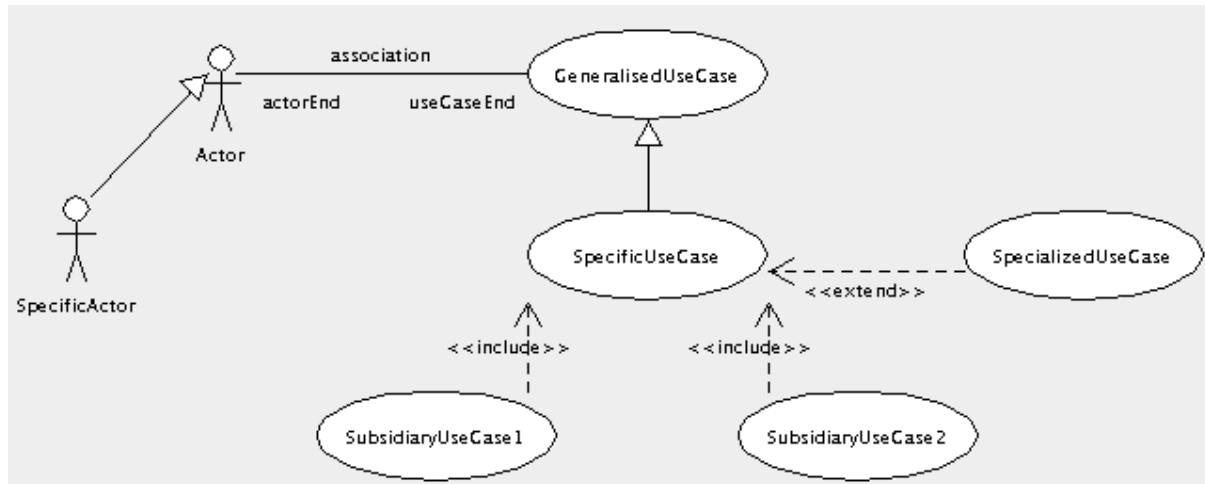
15.1.1. ArgoUML Limitations Concerning Use Case Diagrams

During the standardization process of UML, there was considerable change over the definition of `Includes` (originally known as `Uses`) and `Extends`. Initially they were specializations of `Generalization` within the UML metamodel (itself a specialization of `Relationship`.

In UML 1.3, `Includes` and `Extends` are defined as specializations directly from `Relationship`.

Warning

ArgoUML further confuses matters by defining `Includes` and `Extends` as stereotyped specializations of `Dependency` (itself a specialization of `Relationship`. This is an area where ArgoUML is out of line at present with the standard.

Further difficulties arise with `Extension Points`, the points within a use case, where extending use cases may be invoked. In the absence of a `LocationReference` within ArgoUML these are restricted to purely documentation features.

15.2. Actor

An actor represents any external entity (human or machine) that interacts with the system, providing input, receiving output, or both.

Within the UML metamodel, actor is a sub-class of `classifier`.

The actor icon is a stick man (see Figure 15.1).

15.2.1. Actor Details Tabs

The details tabs that are active for actors are as follows.

ToDoItem   Standard tab.

Properties   See Section 15.2.2 and Section 15.2.3 below.

Documentation Standard tab.

Style   Standard tab. Fill color is used for the stick man's head.

Source   Standard tab. It would not be usual to provide any code for an actor, since it is external to

the system.

Note

The source tab content can be changed, but this has no effect.

Constraints   Standard tab. There are no standard constraints defined for `Actor` within the UML meta-model.

Tagged Values Standard tab. In the UML metamodel, `Actor` has the following standard tagged values defined.

- `persistence` (from the superclass, `Classifier`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

Tip

Actors sit outside the system, and so their internal behavior is of little concern, and this tagged value is best ignored.

- `semantics` (from the superclass, `Classifier`). The value is a specification of the semantics of the actor.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the actor is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived actors have limited value, since they sit outside the system being designed. They may have their value in analysis to introduce useful names or concepts.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.2.2. Actor Property Toolbar

Go up   Navigate up through the package structure of the model.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Add actor   This creates a new actor within the model, (but not within the diagram), navigating immediately to the properties tab for that actor.

167

Tip

This method of creating new actors can be confusing. Much better to create them within diagrams where you want them.

✕ `Delete`   This deletes the selected actor from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete an actor from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

15.2.3. Property Fields For Actor

`Name`   Text box. The name of the actor. Since an actor is a classifier, it would be conventional to Capitalize the first letter (and initial letters of any component words), e.g. `RemoteSensor`.

Note

ArgoUML does not enforce any naming convention for actors

`Stereotype` Drop down selector. Actor is provided by default with the UML standard stereotypes (`metaclass`, `powertype`, `process`, `thread`, `utility`) for classifiers. ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`. Stereotypes are of limited value with actors, but these last three are probably of most use.

`Navigate stereotype` Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

`Namespace` Text box. Records the namespace for the actor. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

`Modifiers` Check box, with entries `Abstract`, `Final` and `Root`.

- `abstract` is used to declare that this actor cannot be instantiated, but must always be specialized.

Caution

Whilst actors can be specialized and generalized, it is not clear that an abstract actor has any meaning. Perhaps it might be used to indicate an actor that does not itself interact with a use case, but whose children do.

- `final` indicates that this actor can have no further children, while `root` indicates it is a top level actor with no parent.

`Generalizations` Text area. Lists any actor that *generalizesgeneralizes* this actor.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the

diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the actor (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

`Special`Text box.Lists any specialized actor (i.e. for which this actor is a generalization. The specialized actors can communicate with the same use case instances as this actor.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the actor (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

`Associations`Text area. Lists any associations connected to the actor. Button 1 click on an association will navigate to that association. Button 2 click gives a pop up menu with five options.

- `Open`. Where a navigation exists, navigate to it.

- `Add`. Add a new association. Note that the new association is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add associations by drawing them on the relevant diagram in the editing pane.

- Delete. Where an association is linked to this actor, this removes the link to the actor.

Tip

This *does notdoes not* remove the association from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove associations from the editing pane.

- Move Up. Only available where there are two or more associations listed, and the association selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more associations listed, and the association selected is not at the bottom. It is moved one down the list.

ImplemeTetxs area. Listing the classes that are implemented by this actor. Although it is possible to create an abstraction through a button 2 pop up menu, this makes no sense for an actor.

15.3. Use Case

A use case represents a complete meaningful "chunk" of activity by the system in relation to its external users (actors), human or machine. It represents the primary route through which requirements are captured for the system under construction

Within the UML metamodel, use case is a sub-class of classifier.

The use case icon is an oval (see Figure 15.1).

15.3.1. Use Case Details Tabs

The details tabs that are active for use cases are as follows.

ToDoItem   Standard tab.

Properties   See Section 15.3.2 and Section 15.3.3 below.

Documentation Standard tab.

Style   Standard tab. Fill color is used for the use case oval.

Source   Standard tab. It would not be usual to provide any code for a use case, since it is primarily a vehicle for capturing requirements about the system under construction, not creating the solution.

Constraints   Standard tab. There are no standard constraints defined for UseCase within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, UseCase has the following standard tagged values defined.

- persistence (from the superclass, Classifier). Values transitory, indicating state is destroyed when an instance is destroyed or persistent, marking state is preserved when an instance is destroyed.

Tip

In general the instantiation of use cases is not a major aspect of any design method (they are mostly concerned with requirements capture. For most OOA&D methodologies, this tag can safely be ignored.

- semantics (from the superclass, Classifier). The value is a specification of the semantics of the use case.

- derived (from the superclass, ModelElement). Values true, meaning the use case is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived use cases still have their value in analysis to introduce useful names or concepts.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.3.2. Use Case Property Toolbar

Go up   Navigate up through the package structure of the model.

⇦Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⬭Add use case   This creates a new use case within the model, (but not within the diagram), navigating immediately to the properties tab for that use case.

Tip

This method of creating new use cases can be confusing. Much better to create them within diagrams where you want them.

EP Add extension point   This creates a new use extension point within the namespace of the current use case, with the current use case as its associated use case, navigating immediately to the properties tab for that extension point.

✗Delete   This deletes the selected use case from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete a use case from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

171

15.3.3. Property Fields For Use Case

Name   Text box. The name of the use case. Since a use case is a classifier, it would be conventional to Capi-
        talize the first letter (and initial letters of any component words), e.g. RemoteSensor.

Note

ArgoUML does not enforce any naming convention for use cases

Stereotype   Drop down selector. Use case is provided by default with the UML standard stereotypes meta-
        (class, powertype, process, thread, utility) for classifiers. ArgoUML also provides the
        additional stereotypes, machine, organization and person. Stereotyping can be useful when
        creating use cases in the problem domain (requirements capture) and solution domain (analysis), but
        none of the pre-defined stereotypes are well suited to this.

Warning

If a stereotype is selected in ArgoUML it will not be displayed in the diagram.

Navigate Stereotype   If a stereotype has been selected, this will navigate you to the stereotype property panel (see
        Section 14.4).

Namespace   Text box. Records the namespace for the use case. This is the package hierarchy.

        Button 1 click on the entry will take you to the package defining this namespace (or the model for the
        top level namespace).

Modifiers   Check box, with entries Abstract, Final and Root.

        • abstract is used to declare that this use case cannot be instantiated, but must always be special-
          ized.

Caution

Whilst use cases can be specialized and generalized, it is not clear that an abstract use case has any
meaning. Perhaps it might be used to indicate an use case that does not itself interact with an actor, but
whose children do.

        • final indicates that this use case can have no further children, while root indicates it is a top
          level use case with no parent.

Generalizations   Text area. Lists use cases which are generalizations of this one. Will be set whenever a generalization
        is created on the diagram. Button 1 click on a generalization will navigate to that generalization. But-
        ton 2 click gives a pop up menu with five options.

        • Open. Where a generalization exists, navigate to it.

        • Add. Add a new generalization. Note that the new association is added to the model, but not the
          diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

Caution

It is possible to add multiple generalizations. However is is hard to see a sensible meaning for this. In

172

addition the size of the text area makes this very inconvenient in ArgoUML.

- Delete. Where a generalization exists linking to this use case, this removes the link to this use case.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more generalizations listed (see *cautioncaution* above), and the generalization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more generalizations listed (see *caucautiontion* above), and the generalization selected is not at the bottom. It is moved one down the list.

SpecialTizatbionrLists any specialized use case (i.e. for which this use case is a generalization.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- Open. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- Delete. The link from the generalization to the use case (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

ExtendsText box. Lists any class that is extended by this use case.

Where an extends relationship has been created (see below), button 1 click will navigate to that relationship. Button 2 gives a pop up menu with five entries.

- Open. Where an extends relationship has been created, navigate to it.

- Add. Add a new extend relationship and navigate to it. The current use case is set as the extension end of the extend relationship.

Caution

It is possible to add multiple extends relationships. However is is hard to see a sensible meaning for

this.

- • `Delete`. Removes the link to the selected extend relationship.

Note

This deletes the connection to the extend relationship, but does not delete the extend relationship from the model.

- • `Move Up`. Only available where there are two or more extend relationships listed (not recommended), and the extend relationship selected is not at the top. It is moved one up the list.

- • `Move Down`. Only available where there are two or more extend relationships listed (not recommended), and the extend relationship selected is not at the bottom. It is moved one down the list.

`Include` Text box. Lists any use case that this use case includes.

Where an include relationship has been created (see below), button 1 click will navigate to that relationship. Button 2 gives a pop up menu with five entries.

- • `Open`. Where an include relationship has been created, navigate to it.

- • `Add`. Add a new include relationship and navigate to it. The current use case will be made the base use case of the include relationship.

- • `Delete`. Removes the link to the selected include relationship.

Note

This deletes the connection to the include relationship, but does not delete the include relationship from the model.

- • `Move Up`. Only available where there are two or more include relationships listed, and the include relationship selected is not at the top. It is moved one up the list.

- • `Move Down`. Only available where there are two or more include relationships listed, and the include relationship selected is not at the bottom. It is moved one down the list.

`Extension Point` Text box. If this use case is, or can be extended, this field lists the extension points for the use case.

Note

Extension points are listed by their location point rather than their name.

Where an extension point has been created (see below), button 1 click will navigate to that relationship. Button 2 gives a pop up menu with five entries.

- • `Open`. Where an extension point has been created, navigate to it.

- • `Add`. Add a new extension point and navigate to it, making this use case the owning use case of the extension point.

- • `Delete`. Where an extension point has been created, this removes it.

Note

This deletes the connection to the extension point, but does not delete the extension point from the model.

- Move Up. Only available where there are two or more extension points listed, and the extension point selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more extension points listed, and the extension point selected is not at the bottom. It is moved one down the list.

Associations area. Lists any associations connected to the use case. Button 1 click on an association will navigate to that association. Button 2 click gives a pop up menu with five options.

- Open. Navigate to the association.

- Add. Add a new association. Note that the new association is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add associations by drawing them on the relevant diagram in the editing pane.

- Delete. Where an association is linked to a use case, this removes the link to the use case.

Tip

This *does notdoes not* remove the association from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove associations from the editing pane.

- Move Up. Only available where there are two or more associations listed, and the association selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more associations listed, and the association selected is not at the bottom. It is moved one down the list.

Operations area. Listing the operations that are available on this use case. As a sub-class of the UML classifier metaclass, a use case has many of the properties of a class (see Section 16.5).

In some OOA&D processes it might be useful to treat use cases as classes as well, and give them operations and attributes. This is of very limited in ArgoUML, since it is not possible to take a use case and add it to a class diagram, so all class relationships must be shown with the notation of a use case diagram.

Should you wish to add operations to a use case, button 1 click allows you to navigate to them and button 2 click gives a pop up menu with options to Open, Add, Delete, Move Up and Move Down (see Generalizations for examples of how these operations are used).

Attributes area. Listing the attributes that are available on this use case. As a sub-class of the UML classifier metaclass, a use case has many of the properties of a class (see Section 16.5).

In some OOA&D processes it might be useful to treat use cases as classes as well, and give them operations and attributes. This is of very limited in ArgoUML, since it is not possible to take a use case and add it to a class diagram, so all class relationships must be shown with the notation of a use case diagram.

Should you wish to add attributes to a use case, button 1 click allows you to navigate to them and button 2 click gives a pop up menu with options to Open, Add, Delete, Move Up and Move Down (see Generalizations for examples of how these operations are used).

15.4. Association

An association on a use case diagram represents is a relationship between an actor and a use case showing that actor's involvement in the use case. The invocation of the use case will involve some (significant) change per-

ceived by the actor.

Within the UML metamodel, `Association` is a sub-class of both `Relationship` and `Generaliz-ableElement`.

The association is represented as a line connecting actor and use case (see Figure 15.1). The name of the association appears above the line and any stereotype above that.

Note

ArgoUML provides no specific way of showing the direction of the association as described in the UML 1.3 standard. The naming should attempt to make this clear.

The association contains two ends, which may be navigated to via the association property sheet. See Section 15.5 for more information.

15.4.1. Association Details Tabs

The details tabs that are active for associations are as follows.

ToDoItem   Standard tab.

Properties   See Section 15.4.2 and Section 15.4.3 below.

Documentation Standard tab.

Style   Standard tab. Fill color is used for the use case oval.

Note

The values for the bounds of the Association appear to have no meaning. Changing them has no effect on the diagram. This is sensible behavior, given the association is tied to a particular actor and use case.

Source   Standard tab. You would not expect to generate any code for an association, and any code entered here is ignored (it will have disappeared when you come back to the association.

Constraints   Standard tab. In the UML metamodel, `Association` has the following standard constraints defined.

- `xor`. Applied to a set of associations, indicating that over that set, exactly one association is manifest for any particular association instance.

Tagged Values Standard tab. In the UML metamodel, `Association` has the following standard tagged values defined.

- `persistence`. Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the association is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived associations still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.4.2. Association Property Toolbar

Go up
Navigate up through the package structure of the model. For an association this will be the package containing the association.

⇦ Go back
Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨ Go forward
Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete
This deletes the selected association from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete an association from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

15.4.3. Property Fields For Association

Name   Text box. The name of the association. By convention association names start with a lower case letter, with "bumpy caps" used to indicate words within the name, thus: `salesHandling`.

Note

ArgoUML does not enforce any naming convention for associations.

Tip

Although the design critics will advise otherwise, it is perfectly normal not to name associations on a use case, since the relationship is obvious from the actor and use case name.

Stereotype   Drop down selector. Association is provided by default with the UML standard stereotype for Association (`implicit`) and the additional Association stereotypes for the Software Engineering Process and Business Modeling in the UML standard (`communicate` and `subscribe`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person` and four additional stereotypes suggested by some writers using patterns (`offstage actor`, `primary actor`, `stakeholder` and `supporting actor`).

Stereotyping can be useful when creating use cases in the problem domain (requirements capture) and solution domain (analysis), as well as for processes based on patterns.

The stereotype is shown between « and » above the name of the association on the diagram.

Warning

The provision of `offstage actor`, `primary actor`, `stakeholder` and `supporting ac-tor` by ArgoUML as stereotypes of `Association` appears to be erroneous. They should be stereotypes of `Actor` (or arguably `Classifier`).

`Navigat ↱ Stereotype`   If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

`Namespace`   Text box. Records the namespace for the association. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

`Modifiers`   Check box, with entries `Abstract`, `Final` and `Root`. No known use.

Tip

Whilst attributes are generalizable elements, marking them as with these modifiers is of limited value with binary associations (the only support that ArgoUML supports properly.

`Association Ends`   Text area. Lists the ends of this association. An association can have any number of ends, but for use cases, two is the only useful number (associative classes can led to a third end on class diagrams, but this is not supported by ArgoUML). For more on association ends see Section 15.5.

The names are listed, unless the association end has no name (the case when it is first created), in which case the artifact at that end, and its multiplicity, is shown.

Note

The only representation of association ends on a diagram is that their name appears at the relevant end of the corresponding association. There is no way to access them from the diagram.

Button 1 click on an association end will navigate to that end. Button 2 click gives a pop up menu with five options.

- `Open`. Where an association end exists, navigate to it.

- `Add`. Add a new association end and navigate to its property tab.

Tip

Whilst adding new ends is possible, the value of this is limited for Use Case diagrams where the standard two ends is all that is every needed.

- `Delete`. Where an association end this removes it completely from the model.

- `Move Up`. Only available where there are two or more association ends, and the association end selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more association ends listed, and the association end selected is not at the bottom. It is moved one down the list.

`Generalization`   Text area. Lists any association that *generalizes* this association.

Caution

It is possible to create generalizations using the pop-up menu from here, but there is no way within ArgoUML to give them a parent, so the entry has no use.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- Open. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- Delete. The link from the generalization to the association (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

SpecialTextaboxorLists any specialized association (i.e. for which this association is a generalization.

Caution

It is possible to create generalizations using the pop-up menu from here, but there is no way within ArgoUML to give them a parent, so the entry has no use.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- Open. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- Delete. The link from the generalization to the association (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

15.5. Association End

Two or more association ends are associated with each association (see Section 15.4), although for the purposes of use case diagrams, the number of ends can only usefully be two.

Within the UML metamodel, `AssociationEnd` is a sub-class of `ModelElement`.

The association end has no direct representation on any diagram, although its name is shown at the relevant end of the parent association (see Figure 15.1).

Tip

Although you can change attributes of association ends when creating a use case model, this is often not necessary. Many of the properties of an association end relate to its use in class diagrams, and are of limited relevance to use cases. The most useful attributes to consider altering are the name (used as the role name) and the multiplicity.

Note

ArgoUML does not currently support qualifiers, as described in the UML 1.3 standard.

15.5.1. Association End Details Tabs

The details tabs that are active for associations are as follows.

ToDoItem   Standard tab.

Properties   See Section 15.5.2 and Section 15.5.3 below.

Documentation Standard tab.

Style   Standard tab, but with no meaning, since the only representation is via the parent association.

Source   Standard tab. You would not expect to generate any code for an association end, but this tab contains a declaration for the association end as an instance of the actor or use case artifact to which it is connected.

Note

This is probably not the correct thing to do for good code generation, and will need to be changed in future versions of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for `AssociationEnd` within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, `AssociationEnd` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the association end is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Tip

Derived association ends still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation. However they their composition with association, means they can only

sensibly have a true `derived` tag if the association also does.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.5.2. Association End Property Toolbar

Go up   Navigate up to the association to which this end belongs.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go to other end   This navigates to the other end of the association.

15.5.3. Property Fields For Association End

Name   Text box. The name of the association end, which provides a *role namerole name* for this end of the association. This role name can be used for navigation, and in an implementation context (not really relevant to use case diagrams), provides a name by which the source end of an association can reference the target end.

Note

ArgoUML does not enforce any naming convention for associations.

StereotypeDrop down selector. Association end is provided by default with the UML standard stereotypes for AssociationEnd (`association`, `global`, `local`, `parameter`, `self`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`.

Stereotyping association ends can be very useful in class diagrams. However it is of limited use in use case diagrams, although the default stereotype of `association` can be used to emphasis this.

Warning

There is no diagram representation of the stereotype of an association end.

Navigate to stereotypeIf a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

Type   Drop down selector providing access to all standard UML types provided by ArgoUML and all new classes created within the current model.

This is the type of the entity attached to this end of the association.

Tip

By default ArgoUML will select the class of the artifact to which the linkend is connected. There is little point in altering this when working with use case diagrams.

Caution

When you create new classes elsewhere in ArgoUML they will not appear in this drop down until after the model has been saved and reloaded.

Navigat Class If a datatype, interface or class has been selected, this will navigate you to its property sheet, (see Section 14.3, Section 16.14 and Section 16.5) .

Multipl Dropdown menu. Records the multiplicity of this association end (with respect to the other end), i.e. how many instances of this end may be associated with an instance of the other end.

Tip

For use case diagrams you can often ignore this, at least in the early stages.

Associa Text box. Records the parent association for this association end. Button 1 click on this entry will take you to that association. Button 2 click will invoke a pop-up menu, with a single entry, `Open` that will also navigate you to that association.

Navigab Check box, with a single entry, `navigable`. Indicates that this end can be navigated to from the other end.

Tip

This means little to a use case diagram. It is useful for associations on class diagrams.

Note

The UML 1.3 standard provides a number of options for how navigation is displayed on an association end. ArgoUML uses option 3, which means that arrow heads are shown at the end of an association, when navigation is enabled at only one end, to indicate the direction in which navigation is possible. This means that the default, with both ends navigable has non arrows. This is an appropriate representation for use cases.

Ordering Radio box, with three entries `unordered`, `ordered` and `sorted`. Where there are multiple instances indicates the nature of any ordering.

Note

`sorted` represents an extension to UML 1.3.

Tip

This means little to a use case diagram. It is useful for associations on class diagrams.

Aggrega Radio box, with three entries `none`, `aggregation` and `composite`. Indicates whether the relationship with the far end represents some type of loose grouping (`aggregation`) or tight grouping (`composite`.

Note

You may not have aggregation at both ends of an association.

Tip

Whilst it does have some meaning, it is of limited use to show aggregation on a use case diagram. It is useful for associations on class diagrams.

Classifier Check box, with a single entry, `classifier`. Indicates that this end relates to a class rather than an instance of a class (the default).

Tip

Not relevant to a use case diagram. It is useful for associations on class diagrams.

Changeability Radio box, with three entries `changeable`, `frozen` and `add only`. Indicates whether instances of this end of the attributed may be: i) created and deleted by the source after the target instance is created; ii) not created or deleted by the source after the target instance is created; or iii) created but not deleted after the target instance is created.

Tip

This is entirely related to associations as used in the realization of a solution, and is not relevant to use case diagrams.

Visibility Radio box, with three entries `public`, `protected` and `private`. Indicates whether navigation to this end may be by: i) any classifier; ii) only the source classifier and its children; or iii) only by the source classifier.

Tip

This is entirely related to associations as used in the realization of a solution, and is not relevant to use case diagrams.

15.6. Generalization

Generalization is a relationship between two use cases or two actors. Where A is a generalization of B, it means A describes more general behavior and B a more specific version of that behavior.

Examples for a travel agent sales system might be the use case for making a booking as a generalization of the use case for making a flight booking and a salesman actor being a generalization of a supervisor actor (since supervisors can also act as salesmen, but not vice versa).

Generalization is analogous to class inheritance within OO programming.

Note

It is easy to confuse *extends*extends relationships between use cases with generalization. However extends is about augmenting a use case's behavior at a specific point. Generalization is about specializing the behavior throughout the use case.

Within the UML metamodel, `Generalization` is a sub-class of `Relationship`.

Generalization is represented as an arrow with white filled head from the specialized use case or actor to the generalized use case or actor (see Figure 15.1).

15.6.1. Generalization Details Tabs

The details tabs that are active for associations are as follows.

ToDoItem   Standard tab.

Properties   See Section 15.6.2 and Section 15.6.3 below.

Documentation Standard tab.

Style   Standard tab

Note

The values for the bounds of the generalization appear to have no meaning. Changing them has no effect on the diagram. This is sensible behavior, given the generalization is tied to a particular actor and use case.

Source   Standard tab. You would not expect to generate any code for an association end, but this tab contains the name of the generalization.

Note

This is probably not the correct thing to do for good code generation, and will need to be changed in future versions of ArgoUML.

Constraints   Standard tab. In the UML metamodel, `Generalization` has the following standard constraints defined.

- `complete`. Applied to a set of generalizations with the same discriminator (see Section 16.13) and same parent, indicating that an instance of the parent must be an instance of one of those children.

  Where there is only one discriminator, this implies the parent must be abstract.

Tip

The point of this constraint is to indicate that within the design, no more children are expected to be created for this parent. If more are created, the implication is that there will be significant rework required within the design.

- `disjoint`. Applied to a set of generalizations indicates and instance of the parent must be an instance of no more than one of the children within the set.

Note

This is the default semantics of generalization

- `incomplete`. The converse of `complete` indicating that within a set of generalizations of the same parent sharing the same discriminator, an instance of the parent need not be an instance of a child within the set (there is no guarantee such an instance exists)

  This implies the parent must be concrete.

Tip

The point of this constraint is to indicate that within the design, more children may be created for this parent. If more are created, the implication is that there should be no significant rework required within the design.

- `overlapping`. The inverse f `disjoint`. Applied to a set of generalizations indicates an instance of one child in the set may simultaneously be an instance of another child within the set (although this is not guaranteed).

Tagged Values Standard tab. In the UML metamodel, `Generalization` has the following standard

tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the generalization is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived generalizations still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.6.2. Generalization Property Toolbar

Go up — Navigate up through the package structure of the model. For a generalization this will be the package containing the generalization.

Go back — Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward — Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Delete — This deletes the selected generalization from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete a generalization from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

15.6.3. Property Fields For Generalization

Name — Text box. The name of the generalization.

Tip

It is quite common to leave generalizations unnamed in use case analysis.

Note

ArgoUML does not enforce any naming convention for associations.

Note

There is no representation of the stereotype of a generalization on the diagram.

Stereotype Drop down selector. Generalization is provided by default with the UML standard stereotype, `im-plementation`. ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`. The stereotype is shown between « and » above or across the generalization.

Tip

Stereotyping generalization does not have great value on a use case diagram. The standard stereotype is about implementation, and suited to the use of generalization on class diagrams.

Navigate Stereotype Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

Discriminator Text box. The name of a discriminator for the specialization. UML 1.3 allows grouping of specializations into a number of sets, on the basis of this value.

Tip

The empty string "" is a valid entry (and the default) for this field. The discriminator is only of practical use in cases of multiple inheritance. An example is discussed in the section covering generalization on class diagrams (see Section 16.13). There is little point in use this with a use case diagram.

Namespace Text box. Records the namespace for the generalization. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Parent Text box. Records the use case or actor that is the *parentparent* in this relationship, i.e. the more general end of the relationship. Button 1 click on this entry will take you to that use case or actor. Button 2 click will give a pop up menu, with a single entry, `Open` which will also navigate to that use case or actor.

Child Text box. Records the use case or actor that is the *childchild* in this relationship, i.e. the more specific end of the relationship. Button 1 click on this entry will take you to that use case or actor. Button 2 click will give a pop up menu, with a single entry, `Open` which will also navigate to that use case or actor.

Powertype Drop down selector providing access to all standard UML types provided by ArgoUML and all new classes created within the current model.

This is the type of the child entity of the generalization.

Tip

This can be ignored for use case analysis. The only sensible value to put in would be the child use case type (as a classifier, this appears in the drop down list.

Figure 15.2. Example use of a discriminator with generalization
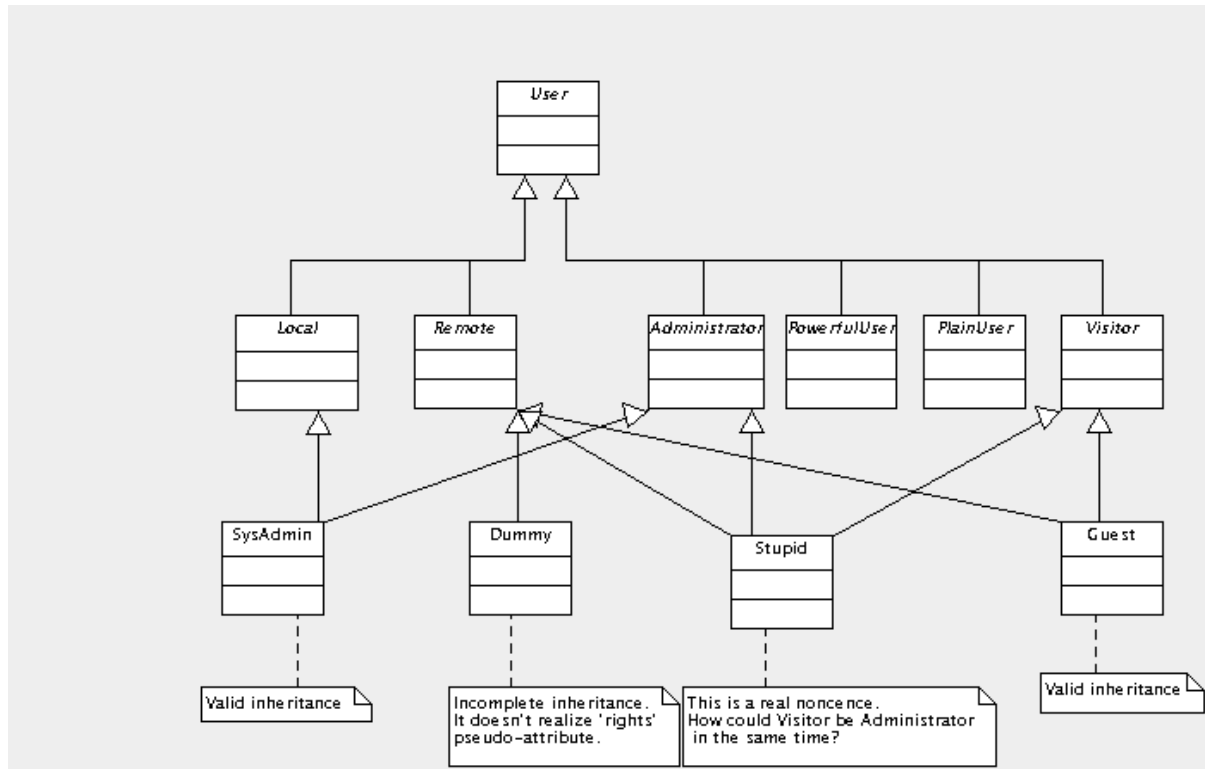
15.7. Extend

Extend is a relationship between two use cases. Where A extends B, it means A describes more specific behavior and B the general version of that behavior.

In many respects extend is like generalization. However the key difference is that the extended use case defines *extension pointsextension points* (see Section 15.8), which are the only places where its behaviour may be extended. The extending use case must define at which of these extension points it adds behavior.

This makes the use of extend more tightly controlled than general extension, and it is thus preferred wherever possible.

Examples for a travel agent sales system might be the use case for paying for a ticket, which has an extension point in the specification of the payment. Extending use cases may then extend at this point to pay by cash, credit card etc.

Within the UML metamodel, Extend is a sub-class of Relationship.

An extend relationship is represented as a dotted link with an open arrow head and a label «extend» (see Figure 15.1).

15.7.1. Extend Details Tabs

The details tabs that are active for extend relationships are as follows.

Note

There is no source tab, since there is no source code that could be generated for an extend relationship.

ToDoItem   Standard tab.

Properties   See Section 15.7.2 and Section 15.7.3 below.

Documentation Standard tab.

Style   Standard tab

Note

The values for the bounds of the extend relationships appear to have no meaning. Changing them has no effect on the diagram. This is sensible behavior, given the extend is tied to a particular pair of use cases.

Constraints   Standard tab. In the UML metamodel, Extend has no standard constraints defined.

Tagged Values Standard tab. In the UML metamodel, Extend has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the extend relationship is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived extend relationships could have their value in analysis to introduce useful names or concepts.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.7.2. Extend Property Toolbar

Go up   Navigate up through the package structure of the model. For a extend this will be the package containing the extend.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

EP Add extension point   This creates a new use case extension point within the namespace of the current extend relationship, with the current extend relationship as its first extending relationship, navigating immediately to the properties tab for that extension point.

Tip

Whilst it is perfectly valid to create extension points from an extend relationship, the created extension point will have no associated use case (it can subsequently be set up).

It would be more usual to instead create the extension point within a use case and subsequently link to it from an extend relationship (see Section 15.7.3 below).

✗ Delete   This deletes the selected extend relationshp from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete a extend from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

15.7.3. Property Fields For Extend

Name   Text box. The name of the extend relationship.

Tip

It is quite common to leave extends unnamed in use case analysis.

Note

ArgoUML does not enforce any naming convention for extend relationships.

Note

There is no representation of the stereotype of an extend relationship on the diagram.

Stereotype Drop down selector. ArgoUML provides the non-standard stereotypes, machine, organization and person. It also rather bizarly provides the realize stereotype (which is more properly just a stereotype of abstraction relationships.

Tip

Stereotyping does not have great value on an extend relationship.

Navigate to Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

Namespace Text box. Records the namespace for the extend. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Base   Drop down selector. Records the use case that is being extended by this extend relationship. Button 1 click on this entry will give a drop down menu of all available use cases (and an empty entry) which may be selected by button 1 click.

Caution

In the current version of ArgoUML if you change the base use case on an extend relationship that is already shown on a diagram, then it will NOT be redrawn. Currently the only way to effect a redraw is to remove the use cases at each end from the diagram and then add them back from the navigation pane.

Navigate to Use Case (Base) If a use case Base has been selected, this will navigate you to the use case property panel (see Section 15.3).

Extension Drop down selector. Records the use case that is doing the extending through this extend relationship. Button 1 click on this entry will give a drop down menu of all available use cases (and an empty en-

try) which may be selected by button 1 click.

Caution

In the current version of ArgoUML if you change the extension use case on an extend relationship that is already shown on a diagram, then it will NOT be redrawn. Currently the only way to effect a redraw is to remove the use cases at each end from the diagram and then add them back from the navigation pane.

Navigat ↑ Use Case (extension use case) has been selected, this will navigate you to the use case property panel (see Section 15.3).

Extensi Text Box. If this use case is, or can be extended, this field lists the extension points for the use case.

Note

Extension points are listed by their location point rather than their name.

Where an extension point has been created (see below), button 1 click will navigate to that relation-ship. Button 2 gives a pop up menu with six entries.

- Open. Navigate to the selected extension point. Grayed out if there are currently no extension points.

- Add. Add a new extension point before the current extension point in the list and navigate to it. The current extend relationship is added as the first in list of extending relationships of the new extension point.

- Link. Where there are existing extension points (other than those currently used by the extend re-lationship) this produces a drop down menu of those extension points. They are shown in two groups. First those that are extension points of the current extend relationships base use case (and which are therefore semantically valid within the UML standard). Secondly those of other use cases (which are not semnatically valid).

  button 1 click on an extension point will add it to the extension point list of the extend relationship before the selected point in the current extension point list.

Tip

It may still make sense to link to an extension point in the second part of the list in the middle of a model reorganisation. In the fullness of time, an ArgoUML critic will warn of this problem.

- Delete. Where an extension point has been createdAdd, this removes it.

Note

This deletes the connection to the extension point extension point, but does not delete the extension point from the model.

- Move Up. Only available where there are two or more extension points listed, and the extension point selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more extension points listed, and the exten-sion point selected is not at the bottom. It is moved one down the list.

15.8. Extension Point

An extension point describes a point in a use case where an extending use case may provide additional be-haviour.

Examples for a travel agent sales system might be the use case for paying for a ticket, which has an extension point in the specification of the payment. Extending use cases may then extend at this point to pay by cash, credit card etc.

Within the UML metamodel, `Extension Point` is a sub-class of `ModelElement`. It has no representation on the diagram.

15.8.1. Extension Point Details Tabs

The details tabs that are active for extension points are as follows.

Note

There is no source tab, since there is no source code that could be generated for an extension point.

ToDoItem	Standard tab.
Properties	See Section 15.8.2 and Section 15.8.3 below.
Documentation	Standard tab.
Style	Standard tab

Caution

The values here are those of the owning use case.

Constraints	Standard tab. In the UML metamodel, `ExtensionPoint` has no standard constraints defined.
Tagged Values	Standard tab. In the UML metamodel, `ExtensionPoint` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the extension point is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

It is not clear how derived extension points could have any value in analysis.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.8.2. Extension Point Property Toolbar

Go up	Navigate up through the package structure of the model. For a extension point this will be the package containing the extension point.
Go back	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this

button will take you.

⇨ Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the selected extension point from the model.

15.8.3. Property Fields For Extension Point

Name   Text box. The name of the extension point.

Tip

It is quite common to leave extension points unnamed in use case analysis, since they are always listed (within use cases and extend relationships) by their location.

Note

ArgoUML does not enforce any naming convention for extension points.

Stereotype   Drop down selector. ArgoUML provides the non-standard stereotypes, `machine`, `organization` and `person`. It also rather bizarly provides the `realize` stereotype (which is more properly just a stereotype of abstraction relationships.

Tip

Stereotyping does not have great value on an extension point.

Navigate stereotype   If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

Namespace   Text box. Records the namespace for the extension point. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Location   Text box. A description of the location of this extension point within the owning use case.

Tip

Extension points are always listed (within use cases and extend relationships) by their location. Typically this will be the number/name of the paragraph in the specification.

Base Use case   Drop down selector. Records the base use case within which this extension point is defined. Button 1 click on this entry will give a drop down menu of all available use cases (and an empty entry) which may be selected by button 1 click.

Navigate use case   If a base use case has been selected, this will navigate you to the use case property panel (see Section 15.3).

Extending Use Cases   Text box. Lists all use cases which extend the base use case through this extension point.

Where an extending use case exists, button 1 click will navigate to that relationship. Button 2 gives a pop up menu with five entries.

192

- `Open`. Navigate to the extend relationship that links to the selected use case. Grayed out if there are currently no extending use cases.

- `Add`. Add a new extend relationship (which can then add an extension use case) before the current use case in the list and navigate to it. The owning use case of the current extension point relationship is made the base use case of the new extend relationship.

- `Delete`. Where an extend relationship to an extending use case has been created`Add`, this removes the link to it.

Note

This deletes the connection to the extend relationship, but does not delete the extend relationship from the model.

- `Move Up`. Only available where there are two or more use cases listed, and the use case selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more use cases listed, and the use case selected is not at the bottom. It is moved one down the list.

15.9. Include

Include is a relationship between two use cases. Where A includes B, it means B described behaviour that is to be included in the description of the behaviour of A at some point (defined internally by A).

Examples for a travel agent sales system might be the use case for booking travel, which includes use cases for booking flights and taking payment.

Within the UML metamodel, `Include` is a sub-class of `Relationship`.

An include relationship is represented as a dotted link with an open arrow head and a label «`include`» (see Figure 15.1).

15.9.1. Include Details Tabs

The details tabs that are active for include relationships are as follows.

Note

There is no source tab, since there is no source code that could be generated for an include relationship.

`ToDoItem`   Standard tab.

`Properties`   See Section 15.9.2 and Section 15.9.3 below.

`Documentation` Standard tab.

`Style`   Standard tab

Note

The values for the bounds of the include relationships appear to have no meaning. Changing them has no effect on the diagram. This is sensible behavior, given the include is tied to a particular pair of use cases.

`Constraints`   Standard tab. In the UML metamodel, `Include` has no standard constraints defined.

Tagged Values Standard tab. In the UML metamodel, Include has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the include relationship is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived include relationships could have their value in analysis to introduce useful names or concepts.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.9.2. Include Property Toolbar

Go up   Navigate up through the package structure of the model. For a include this will be the package containing the include.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Delete   This deletes the selected include relationshp from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete a include from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

15.9.3. Property Fields For Include

Name   Text box. The name of the include relationship.

Tip

It is quite common to leave include relationships unnamed in use case analysis.

Note

ArgoUML does not enforce any naming convention for include relationships.

Note

There is no representation of the stereotype of an include relationship on the diagram.

Stereotype Drop down selector. ArgoUML provides the non-standard stereotypes, `machine`, `organization` and `person`. It also rather bizarly provides the `realize` stereotype (which is more properly just a stereotype of abstraction relationships.

Tip

Stereotyping does not have great value on an include relationship.

Navigate Stereotype Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

Namespace Text box. Records the namespace for the include. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Base Drop down selector. Records the use case that is doing the including in this include relationship. Button 1 click on this entry will give a drop down menu of all available use cases (and an empty entry) which may be selected by button 1 click.

Caution

In the current version of ArgoUML if you change the base use case on an include relationship that is already shown on a diagram, then it will NOT be redrawn. Currently the only way to effect a redraw is to remove the use cases at each end from the diagram and then add them back from the navigation pane.

Navigate Base Icon. If a base (use case) has been selected, this will navigate you to the use case property panel (see Section 15.3).

Addition Drop down selector. Records the use case that is being included by this include relationship. Button 1 click on this entry will give a drop down menu of all available use cases (and an empty entry) which may be selected by button 1 click.

Caution

In the current version of ArgoUML if you change the addition use case on an include relationship that is already shown on a diagram, then it will NOT be redrawn. Currently the only way to effect a redraw is to remove the use cases at each end from the diagram and then add them back from the navigation pane.

Navigate Addition Icon. If an addition (use case) has been selected, this will navigate you to the use case property panel (see Section 15.3).

15.10. Dependency

Dependency is a relationship between two artifacts showing that one depends on the other.

Caution

Dependency has little use in use case diagrams. It is provided, because earlier versions of ArgoUML used it (incorrectly) to implement include and extends relationships

Within the UML metamodel, `Dependency` is a sub-class of `Relationship`.

Dependency is represented as a dotted line with an open arrow head from the dependent to the depending artifact (see Figure 15.1).

15.10.1. Dependency Details Tabs

The details tabs that are active for dependencies are as follows.

Note

There is no source tab, since there is no source code that could be generated for a dependency relationship.

ToDoItem	Standard tab.
Properties	See Section 15.10.2 and Section 15.10.3 below.
Documentation	Standard tab.
Style	Standard tab

Note

The values for the bounds of the dependency appear to have no meaning. Changing them has no effect on the diagram. This is sensible behavior, given the dependency is tied to a particular use case.

Constraints	Standard tab. There are no standard constraints defined for Dependency within the UML metamodel.
Tagged Values	Standard tab. In the UML metamodel, Dependency has no tagged values of its own, but through superclasses has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the dependency relationship is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived dependencies still have their value in analysis to introduce useful names or concepts.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

15.10.2. Dependency Property Toolbar

Go up	Navigate up through the package structure of the model. For a dependency this will be the package containing the dependency.
Go back	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward	Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✖ Delete   This deletes the selected dependency from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete a dependency from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

15.10.3. Property Fields For Dependency

Name   Text box. The name of the dependency.

Tip

It is quite common to leave dependencies unnamed.

Note

ArgoUML does not enforce any naming convention for associations.

Note

There is no representation of the name of a dependency on the diagram.

Stereotype   Drop down selector. Dependency has no standard stereotypes of its own under UML 1.3. ArgoUML provides two stereotypes `extend` and `include`, which in the past allowed depend to be used to draw these relationships. ArgoUML also provides the additional stereotypes, `machine`, `organization`, `person` and rather bizarrely `realize` (which properly is a stereotype only of abstraction). The stereotype is shown between « and » above or across the generalization.

Navigate stereotype   Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 14.4).

Namespace   Text box. Records the namespace for the dependency. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Supplier   Text area. Lists the "depending" ends of the relationship, i.e. the end supplying what is needed by the other end.

- Open. Where a supplier exists, navigate to it.

- Move Up. Only available where there are two or more suppliers, and the supplier selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more suppliers listed, and the supplier selected is not at the bottom. It is moved one down the list.

Clients   Text area. Lists the "depending" ends of the relationship, i.e. the end supplying what is needed by the other end.

Button 1 click on a client will navigate to that use case. Button 2 click gives a pop up menu with three options.

- `Open`. Where a client exists, navigate to it.

- `Move Up`. Only available where there are two or more clients, and the client selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more clients listed, and the client selected is not at the bottom. It is moved one down the list.

Chapter 16. ArgoUML Class Diagram Artifact Reference

Table of Contents

16.1. Introduction

This chapter describes each artifact that can be created within a class diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

Class diagrams are used for only one of the UML static structure diagrams, the class diagram itself. Object diagrams are represented on the ArgoUML deployment diagram.

In addition, ArgoUML uses the class diagram to show model structure through the use of packages.

There is a close relationship between this material and the Properties Tab of the Details Pane (see Section 10.4). That section covers Properties in general, in this chapter they are linked to specific artifacts.

Figure 16.1 shows a class diagram with all possible artifacts displayed.

Figure 16.1. Possible artifacts on a class diagram.
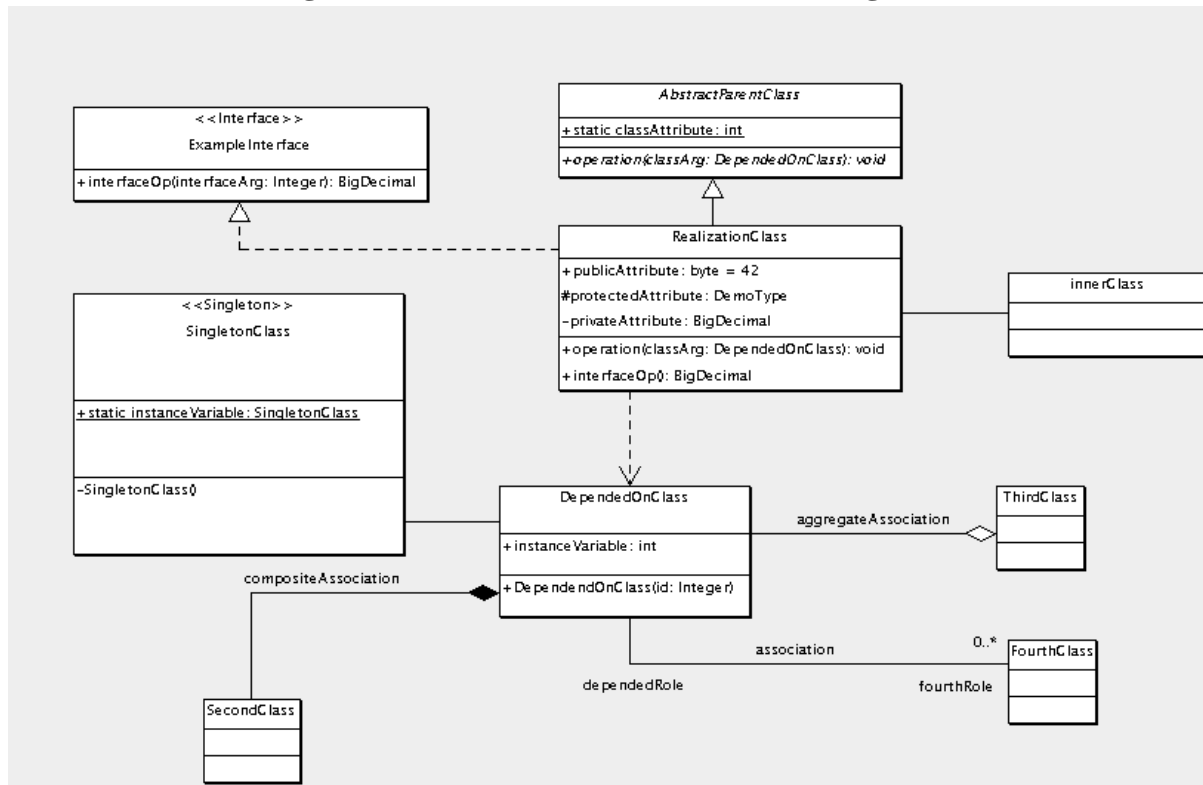
Figure 16.2 shows a package diagram with all possible artifacts displayed.

Figure 16.2. Possible artifacts on a package diagram.
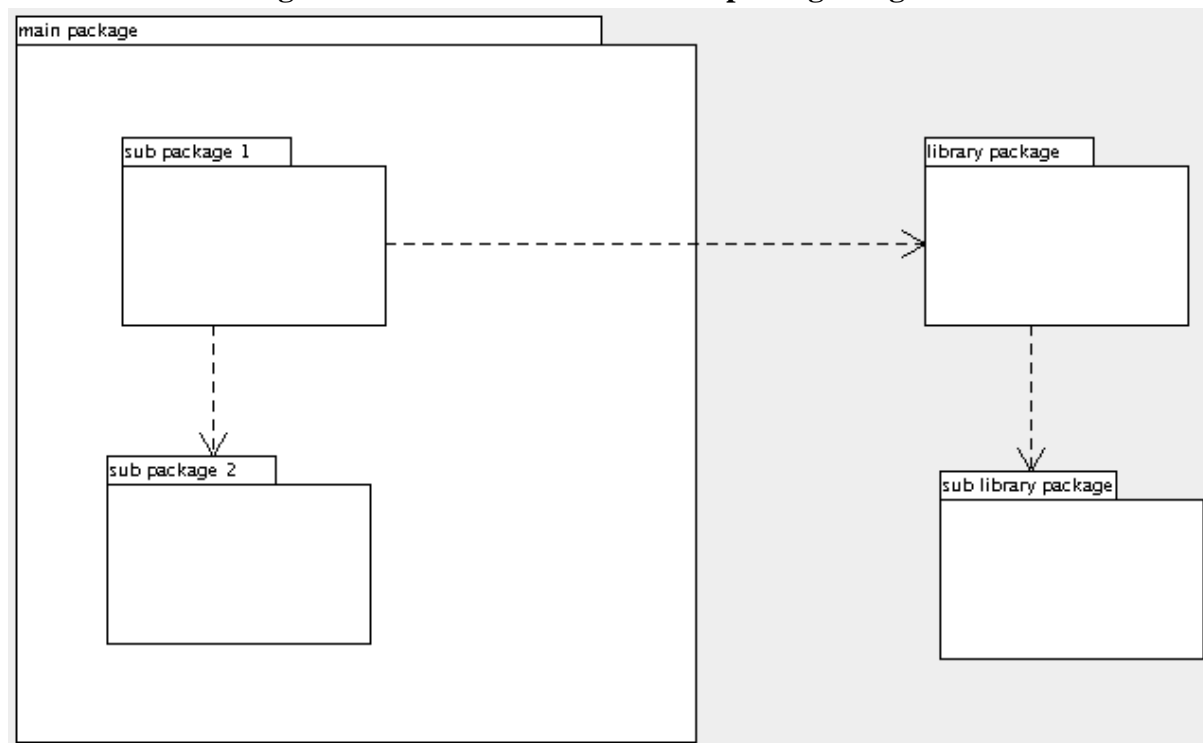
16.2. Package

The package is the main organizational artifact within ArgoUML. In the UML metamodel it is a sub-class of both `Namespace` and `GeneralizableElement`.

Note

ArgoUML also implements the UML `Model` artifact as a sub-class of package (see Section 16.2), but *notnot* the `Subsystem` artifact.

Caution

ArgoUML does not fully implement all aspects of UML model management. In particular the only relationship that can be shown between packages is `Dependency`, although UML 1.3 defines `Generalization` and the sub-class dependency `Permission` for use between packages.

16.2.1. Package Details Tabs

The details tabs that are active for packages are as follows.

ToDoItem   Standard tab.

Properties   See Section 16.2.2 and Section 16.2.3 below.

Documentation Standard tab.

Style   Standard tab. The `Bounds:` field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

Source   Standard tab. This contains a template package declaration appropriate to the entire package. In the future this would be part of the code generation activity.

Constraints   Standard tab. There are no standard constraints defined for `Package` within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, `Package` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the package is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived packages still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.2.2. Package Property Toolbar

	Go up	Navigate up through the package structure.
	Go back	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

	Go forward	Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

	Add class	This creates a new class (see Section 16.5) within the package (which appears on no diagram), navigating immediately to the properties tab for that class.

Tip

Whilst it can make sense to create classes this way, it can be clearer to create them within diagrams where you want them.

	Add interface	This creates a new interface (see Section 16.14) within the package (which appears on no diagram), navigating immediately to the properties tab for that interface.

Tip

Whilst it can make sense to create interfaces this way, it can be clearer to create them within diagrams where you want them.

	Add datatype	This creates a new datatype (see Section 16.3) within the package (which appears on no diagram), navigating immediately to the properties tab for that datatype.
	Add stereotype	This creates a new stereotype (see Section 16.4) within the package, navigating immediately to the properties tab for that stereotype.
	Add actor	This creates a new actor (see Section 15.2) within the package (which appears on no diagram), navigating immediately to the properties tab for that actor.

Tip

Whilst it can make sense to create actors this way, it is can be clearer to create them within diagrams where you want them.

	Add use case	This creates a new use case (see Section 15.3) within the package (which appears on no diagram), navigating immediately to the properties tab for that use case.

Tip

Whilst it can make sense to create use cases this way, it can be clearer to create them within diagrams where you want them.

	Add subpackage	This creates a new sub-package (see Section 16.2) within the package (which appears on no diagram), navigating immediately to the properties tab for that package.
	Add generalization	This creates a new generalization (see Section 15.6 and Section 16.13) within the package (which appears on no diagram), navigating immediately to the properties tab for that generalization.

Tip

Whilst it can make sense to create generalizations this way, it is usually a lot clearer to create them within diagrams where you want them.

	Add realization	This should create a new realization (i.e. an abstraction with stereotype `realize`—see Section 16.15) within the package (which appears on no diagram).

Warning

This appears to be broken in the current release of ArgoUML. Even if it did work, it would be of limited value, since the abstraction property sheet has no mechanism for adding supplier and client artifacts.

	Delete	Deletes the package from the model.

Warning

This is a deletion from the model *notnot* just the diagram. To delete a package from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

16.2.3. Property Fields For Package

Name   Text box. The name of the package. The name of a package, like all packages, is by convention all lower case, with periods seperating the components of the package hierarchy.

Note

The default name supplied to a new package by ArgoUML is the empty string "". The package will appear with the name `anon` in the navigation pane.

StereotypeDrop down selector. Package is provided by default with the UML standard stereotypes for package (`facade`, `framework`, `stub`), as well as the stereotypes for models and packages in the UML standard profiles for *software development processsoftware development process* (`use case package` and `analysis service package`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`.

Note

A number of standard stereotypes are currently missing from ArgoUML: `topLevel` (from package), `use case system`, `analysis system`, `analysis package` (all three from the *software development processsoftware development process* profile).

Caution

ArgoUML provides the stereotype `realize` for models and packages. This appears to be an error,

since this stereotype properly belongs to the `Abstraction` metaclass.

In addition the stereotypes `systemModel` and `metamodel` are provided, although they should only be used for models (a sub-class of `Package`).

`Navigat` Star If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

`Namespa` Text box. Records the namespace for the package. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

`Modifie` Check box, with entries `abstract`, `final` and `root`.

- `abstract` is used to declare that this package cannot be instantiated, but must always be specialized.

Tip

The meaning of `abstract` applied to a package if not that clear. It might mean that the package contains interfaces or abstract classes without realizations. This is probably better handled through stereotyping of the package (for example «`facade`»).

- `final` indicates that this package can have no further subpackages, while `root` indicates it is the top level package.

Tip

Within ArgoUML `root` only meaningfully applies to the Model, since all packages sit within the model. In the absence of the `topLevel` stereotype, this could be used to emphasize that the Model is at the top level.

`Special` Text box. Lists any specialized package (i.e. for which this package is a generalization.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the package (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more specializations listed, and the special-

ization selected is not at the bottom. It is moved one down the list.

General Text area. lists any package that *generalizesgeneralizes* this package.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- Open. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- Delete. The link from the generalization to the package (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

Implements Text area. Lists any packages for which this package is the realization.

Caution

ArgoUML uses this for any dependent packages, implying that the dependency relationship is one of implementation. However in the absence of a permission relationship, such dependency relationships are also used on diagrams to show import, access and similar relationships. Use of the realize stereotype for true realization relationships is recommended.

It is possible to add realizations from this menu, although since there is no way to add a package to the other end of the abstraction, it is meaningless. Button 1 click allows you to navigate to them and button 2 click gives a pop up menu with options to Open, Add, Delete, Move Up and Move Down (see Generalizations for examples of how these operations are used).

Owned Elements Text area A hierarchical listing of the packages, classes, interfaces, datatypes, actors, use cases, associations, generalizations and stereotypes within the package. For each of these types of artifact, where there is at least one example in the package, a folder icon is shown.

Button 1 double click on the folder icon () opens the folder to reveal the artifacts owned under that category.

Button 1 click on any of the revealed artifacts yields navigates to that artifact.

Button 1 double click on an open folder (with artifacts displayed) will close the folder and hide the artifacts.

Alternatively, Button 2 double click on any item gives a pop up menu.

- `Add`. This option creates a new artifact of the specified type within the package, and navigates to that item.

Note

The item is placed under its relevant folder, irrespective of where the pop up menu is revealed. So button 2 lick on `Actor` folder and selection of `Add/Package` will place the new package under the `Package` folder, not the `Actor` folder.

This is the only menu option actually available to the package (the others apply to packages within the package, see Section 16.2).

Tip

It is often far easier to add artifacts by drawing them on the relevant diagram in the editing pane.

- `Delete`. This is always grayed out for a package.

- `Go`. This is always grayed out for a package.

- `Open`. This is always grayed out for a package.

16.3. Datatype

Datatypes are not specific to packages or class diagrams, and are discussed within the chapter on top level artifacts (see Section 14.3).

16.4. Stereotype

Stereotypes are not specific to packages or class diagrams, and are discussed within the chapter on top level artifacts (see Section 14.4).

16.5. Class

The class is the dominant artifact on a class diagram. In the UML metamodel it is a sub-class of `Classifier` and `GeneralizableElement`.

A class is represented on a class diagram as a rectangle with three horizontal compartments. The top compartment displays the class name (and stereotype), the second compartment any attributes and the third any operations. The last two compartments may optionally be hidden.

16.5.1. Class Details Tabs

The details tabs that are active for classes are as follows.

`ToDoItem`	Standard tab.
`Properties`	See Section 16.5.2 and Section 16.5.3 below.
`Documentation`	Standard tab.
`Style`	Standard tab. The tick boxes, `Attributes` and `Operations` allow the attributes and operations compartments to be shown (the default) or hidden. The `Bounds:` field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

Source   Standard tab. This contains a template for the class declaration (with JavaDoc header) and declarations of associated classes.

Constraints   Standard tab. There are no standard constraints defined for `Class` within the UML meta-model.

Tagged Values   Standard tab. In the UML metamodel, `Class` has the following standard tagged values defined.

- persistence (from the superclass, `Classifier`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- semantics (from the superclass, `Classifier`). The value is a specification of the semantics of the class.

- derived (from the superclass, `ModelElement`). Values `true`, meaning the class is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived classes still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.5.2. Class Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Add attribute   This creates a new attribute (see Section 16.6) within the class, navigating immediately to the properties tab for that attribute.

Add operation   This creates a new operation (see Section 16.7) within the class, navigating immediately to the properties tab for that operation.

Add interface
This creates a new interface (see Section 16.14) within the package (which appears on no diagram), navigating immediately to the properties tab for that interface.

Tip

Whilst it can make sense to create interfaces this way, it can be clearer to create them within diagrams where you want them.

Add inner class
This creates a new inner class (which appears on no diagram) within the class. This belongs to the class and is restricted to the namespace of the class. It exactly models the Java concept of inner class. As an inner class it needs no attributes or operations, since it shares those of its owner.

Note

Inner class is not a separate concept in UML. This is a convenient shorthand for creating a class that is restricted to the namespace of its owning class.

Delete
This deletes the class from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a class from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

16.5.3. Property Fields For Class

Name
Text box. The name of the class. The name of a class has a leading capital letter, with words separated by "bumpy caps".

Note

The ArgoUML critics will complain about class names that do not have an initial capital.

Stereotype
Drop down selector. Class is provided by default with the UML standard stereotypes for `Class` im- (plementationClass and type) and for Classifier (metaclass, powerType, process, thread and utility), as well as the stereotypes for classes in the UML standard profiles for *software development processsoftware development process* (boundary, entity and control) and *business modelingbusiness modeling* (worker, case worker, internal worker and entity (again)). ArgoUML also provides the additional stereotypes, machine, organization and person, and rather strangely realize (a stereotype of Abstraction).

Caution

ArgoUML provides the stereotype realize for classes. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Tip

One stereotype that is not part of the UML standard, but is widely used is Singleton, used to distinguish classes which have a single static instance, and no public constructor. Although not part of ArgoUML by default, this stereotype is understood by the critics. You may find it useful to create this stereotype for yourself (see Section 14.4).

Navigate stereotype
Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace Text box. Records the namespace for the class. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Generalizations Text area. Lists any class that *generalizesgeneralizes* this class.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the class (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

Modifiers Check box, with entries `public`, `abstract`, `final`, `root` and `active`.

- `public` has identical meaning to the `public` declaration in Java. It indicated whether this class is restricted to use in its own package (the default) or can be used by anyone.

- `abstract` is used to declare that this class cannot be instantiated, but must always be subclassed. The name of an abstract class is displayed in italics on the diagram.

Caution

If a class has any abstract operations, then it should be declared abstract. ArgoUML will not enforce this.

- `final` indicates that this class cannot be further subclassed, while `root` indicates it can have no superclass. It is possible for a class to be both abstract and final, since its static operations may still be referenced.

- `active` indicates that this class exhibits dynamic behavior (and is thus associated with a state or activity diagram).

Associationends Text box. Lists any associations (see Section 16.10) connected to this class.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- `Open`. Navigate to the selected association.

- Add. Add a new association.

Tip

New associations can usually more easily be added from the class diagram.

- Delete. Removes the link to the selected association.

Note

This only removes the link to the association. To remove the association from the model you must navigate to it (or select it on the diagram or navigation pane) and then delete it.

- Move Up. Only available where there are two or more associations, and the association selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more associations listed, and the association selected is not at the bottom. It is moved one down the list.

Implements Text box. Lists any classes or interfaces (see Section 16.10) which are an abstraction of this class (i.e. which this class realizes).

Button 1 click takes you to the abstraction for the selected entry, button 2 gives a pop up menu with five entries.

- Open. Navigate to the selected abstraction.

- Add. Add a new abstraction.

Tip

New abstractions can usually more easily be added from the class diagram.

- Delete. Removes the link to the selected abstraction.

Note

This only removes the link to the abstraction. To remove the abstraction from the model you must navigate to it (or select it on the diagram or navigation pane) and then delete it.

- Move Up. Only available where there are two or more abstractions, and the abstraction selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more abstractions listed, and the abstraction selected is not at the bottom. It is moved one down the list.

Specializations Text box. Lists any specialized class (i.e. for which this class is a generalization.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- Open. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the class (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

`Derived`Text area. Lists any derived classes, i.e those for which this class is a generalization.

Button 1 click takes you to the generalization for the selected entry, button 2 gives a pop up menu with a single entry, which also takes you to the generalization.

Caution

You may potentially have a large number of entries here, and the pop up menu makes access very difficult. The work around is to go to the parent classes on the diagram or the navigation pane.

`Operations`Text area. Lists all the operations (see Section 16.7) defined on this class. Button 1 click takes you to the selected operation, button 2 click brings up a pop up menu with five entries.

- `Open`. Navigate to the selected operation.

- `Add`. Add a new operation and navigate to it.

- `Delete`. Removes the selected operation from the model.

- `Move Up`. Only available where there are two or more operations, and the operation selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more operations listed, and the operation selected is not at the bottom. It is moved one down the list.

`Attributes`Text area. Lists all the attributes (see Section 16.6) defined for this class. Button 1 click takes you to the selected attribute, button 2 click brings up a pop up menu with five entries.

- `Open`. Navigate to the selected attribute.

- `Add`. Add a new attribute and navigate to it.

- `Delete`. Removes the selected attribute from the model.

- `Move Up`. Only available where there are two or more attributes, and the attribute selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more attributes listed, and the attribute selected is not at the bottom. It is moved one down the list.

`Owned Elements`Text area. A listing of artifacts contained within the classes' namespace. This is where any inner class (see Section 16.5.2) will appear

Button 1 click on any of the artifacts navigates to that artifact. button 2 click brings up a pop up menu with five entries.

- Open. Navigate to the selected artifact.

- Add. Point to a sub-menu with seven options: Add_Menu_Actor; Add_Menu_Class; Add_Menu_Datatype; Add_Menu_Exception; Add_Menu_Interface; Add_Menu_Signal and Add_Menu_UseCase. Selecting any one will create an artifact of that type within the namespace of the class (but not on any diagram) and navigate to it.

Note

Add_Menu_Exception is identical to Add_Menu_Signal. Both add a signal.

Tip

Most namespace hierarchies should be managed through the package mechanism. Namespace hierarchies through classes are best restricted to inner classes. Conceivable datatypes, signals and interfaces could also appear here, but actors and use cases would seem of no value.

- Delete. Removes the selected artifact from the model.

- Move Up. Only available where there are two or more artifacts, and the artifact selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more artifacts listed, and the artifact selected is not at the bottom. It is moved one down the list.

16.6. Attribute

Attribute is a named slot within a class (or other Classifier describing a range of values that may be held by instances of the class. In the UML metamodel it is a sub-class of StructuralFeature which is itself a subclass of Feature.

An attribute is represented on a single line within the attribute compartment of the class. Its syntax is as follows:

> *visibilityvisibility* [static] [final] *attributeNameattributeName* : *typetype* [= *initialValueinitialValue*]

visibilityvisibility is +, # or – corresponding to public, protected or private visibility respectively.

static and final optionally appear if the attribute has those modifiers

In addition any attribute declared static will have its whole entry underlined on the diagram.

attributeNameattributeName is the actual name of the attribute being declared.

typetype is the type (UML datatype, class or interface) declared for the attribute.

initialValueinitialValue is any initial value to be given to the attribute when an instance of the class is created. This may be overridden by any constructor operation.

16.6.1. Attribute Details Tabs

The details tabs that are active for attributes are as follows.

ToDoItem   Standard tab.

Properties   See Section 16.6.2 and Section 16.6.3 below.

Documentation Standard tab.

Style   Standard tab, but relating to the owning class, rather than the attribute. The tick boxes, `At-tributes` and `Operations` allow the attributes and operations compartments of the class to be shown (the default) or hidden. The `Bounds:` field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

Source   Standard tab. This contains a declaration for the attribute.

Warning

The declaration for attributes with multiplicity (see Section 16.6.3) is not syntactically correct Java at present.

Constraints   Standard tab. There are no standard constraints defined for `Attribute` within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, `Attribute` has the following standard tagged values defined.

- `persistence`. Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the attribute is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived attributes still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.6.2. Attribute Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

New attribute — This creates a new attribute within the owning class of the current attribute, navigating immediately to the properties tab for that attribute.

Tip

This is a very convenient way to add a number of attributes, one after the other, to a class.

Delete — This deletes the attribute from the model

Warning

This is a deletion from the model *notnot* just the diagram. Since attributes have no meaning independent of their owning class, it makes no sense to delete them just from the model. If desired the whole attribute compartment can be hidden on the diagram using the *stylestyle* tab (see Section 16.6.2) or the button 2 pop up menu for the class on the diagram.

16.6.3. Property Fields For Attribute

Name — Text box. The name of the attribute. The name of a attribute has a leading lower case letter, with words separated by "bumpy caps".

Note

The ArgoUML critics will complain about attribute names that do not have an initial lower case letter.

Multiplicity — Drop down selector. The default (empty) value is that there is one instance of this attribute for each instance of the class, i.e. it is a scalar. The drop down provides a number of specifications for non-scalar attributes.

Note

The selection 1..1 is equivalent to the default (exactly one scalar instance). The selection 0..1 indicates an optional scalar attribute.

Caution

ArgoUML restricts you to a number of predefined ranges for multiplicity. There is no way to give a user defined range.

Stereotype — Drop down selector. There are no UML standard stereotypes for Attribute. ArgoUML provides the additional stereotypes, machine, organization and person, and rather strangely realize (a stereotype of Abstraction).

Caution

ArgoUML provides the stereotype realize for attributes. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Navigate to stereotype — If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Owner — Text box. Records the class which contains this attribute.

Button 1 click on the entry will navigate to the class. Button 2 click provides a pop up menu with a single entry, Open, which will also navigate to the class.

Type — Drop down selector. The type of this attribute. This can be any UML Classifier, although in practice only Class, DataType or Interface make any sense.

Note

A type must be declared (it can be `void`). By default ArgoUML supplies `int` as the type.

Navigat type This will navigate you to the property panel for the currently selected type. (see Section 16.5, Section 16.3 and Section 16.14).

Initial Text box with drop down. This allows you to set an initial value for the attribute if desired (this is optional). The drop down menu provides access to the common values 0, 1, 2 and `null`.

Caution

Any constructor operation may ignore this initial value.

Visibili Radio box, with entries `public`, `protected` and `private`.

- `public`. The attribute is available to any artifact that can see the owning class.

- `protected`. The attribute is available only to the owning class, or artifacts that are subclasses of the owning class.

- `private`. The attribute is available only to the owning class (and any inner classes).

Modifiedr Check box, with entries `static`, `final`, `transient` and `volatile`.

- `static`. There is only one instance of this attribute associated with the class (as opposed to one for each instance of the class). This is the `OwnerScope` attribute of a `Feature` metaclass within UML. Any attribute declared static is shown underlined on the class diagram.

- `final` The value of this attribute may not be changed after it has been instantiated.

Caution

This corresponds to the `Changeability` attribute of the `Attribute` metaclass within the UML standard. ArgoUML has no mechanism for indicating the `addOnly` value of this attribute.

- `transient`. This corresponds to the `permanence` tagged value (see Section 16.6.1), and should not really be on the property sheet.

Tip

UML Purists should not use this check box, but enter a permanence tagged value on the tagged value tab.

- `volatile`. This is an ArgoUML extension to the UML 1.3 standard to indicate that this attribute is realized in some volatile form (for example it will be a memory mapped control register).

Tip

The proper place for this extension is as a tagged value (or stereotype). UML purists may choose to adopt either of these mechanisms.

16.7. Operation

Operation is a service that can be requested from an object to effect behavior. In the UML metamodel it is a subclass of `BehavioralFeature` which is itself a sub-class of `Feature`.

An operation is represented on a single line within the operation compartment of the class. Its syntax is as follows:

>*visibilityvisibility* [static] [final] *operationNameoperationName* (*argumentDeclara-*
>[*tionargumentDeclaration* [, *argumentDeclarationargumentDeclaration*]*] : *re-*
>*returnTypeturnType*

visibilityvisibility is +, # or – corresponding to public, protected or private visibility respectively.

static and final optionally appear if the operation has those modifiers

In addition any operation declared static will have its whole entry underlined on the diagram.

operationNameoperationName is the name of the operation actually being declared.

There may be zero or more *argumentDeclarationsargumentDeclarations* separated by commas. An *argument-DeclarationargumentDeclaration* is a pair of the form:

>*namename* : *typetype*

returnTypereturnType is the type (UML datatype, class or interface) of the result returned.

Finally the whole entry is shown in italics if the operation is declared abstract.

16.7.1. Operation Details Tabs

The details tabs that are active for operations are as follows.

ToDoItem   Standard tab.

Properties   See Section 16.7.2 and Section 16.7.3 below.

Documentation Standard tab.

Style   Standard tab, but relating to the owning class, rather than the operation. The tick boxes, Attributes and Operations allow the attributes and operations compartments of the class to be shown (the default) or hidden. The Bounds: field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the Bounds: field. Take care when typing that you do get the values you want in place.

Source   Standard tab. This contains a declaration for the operation.

Warning

The final modifier is ignored and will not appear in the source declaration.

Constraints   Standard tab. There are no standard constraints defined for Operation within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, Operation has the following standard tagged values defined.

- semantics. The value is a specification of the semantics of the operation.

- derived (from the superclass, ModelElement). Values true, meaning the operation is redundant—it can be formally derived from other elements, or false meaning it

216

cannot.

Note

Derived operations still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.7.2. Operation Property Toolbar

`Go up`	Navigate up through the package structure.
`Go back`	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Go forward`	Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`New operation`	This creates a new operation within the owning class of the current operation, navigating immediately to the properties tab for that operation.

Tip

This is a very convenient way to add a number of operations, one after the other, to a class.

`Add parameter`	This creates a new parameter for the operation, navigating immediately to the properties tab for that parameter.
`Delete`	This deletes the operation from the model

Warning

This is a deletion from the model *notnot* just the diagram. Since operations have no meaning independent of their owning class, it makes no sense to delete them just from the model. If desired the whole operation compartment can be hidden on the diagram using the *stylestyle* tab (see Section 16.7.2) or the button 2 pop up menu for the class on the diagram.

16.7.3. Property Fields For Operation

Name   Text box. The name of the operation. The name of an operation has a leading lower case letter, with words separated by "bumpy caps".

Note

The ArgoUML critics will complain about operation names that do not have an initial lower case letter.

Tip

If you wish to follow the Java convention of constructors having the same name as the class, you will violate this rule. Remove the critic using the to-do tab (see Section 10.3).

Stereotype   Drop down selector. There are two UML standard stereotypes for `Operation` (from the parent metaclass, `BehavioralFeature`), `create` and `destroy`. ArgoUML provides the additional stereotypes, `machine`, `organization` and `person`, and rather strangely `realize` (a stereotype of `Abstraction`).

Tip

You should use `create` as the stereotype for constructors, and `destroy` for destructors (or finalize methods under Java).

Caution

ArgoUML provides the stereotype `realize` for operations. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate to stereotype   If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Owner   Text box. Records the class which contains this operation.

Button 1 click on the entry will navigate to the class. Button 2 click provides a pop up menu with a single entry, `Open`, which will also navigate to the class.

Visibility   Radio box, with entries `public`, `protected` and `private`.

- `public`. The operation is available to any artifact that can see the owning class.

- `protected`. The operation is available only to the owning class, or artifacts that are subclasses of the owning class.

- `private`. The operation is available only to the owning class (and any inner classes).

Modifiers   Check box, with entries `abstract`, `final`, `root`, `query` and `static`.

- `abstract`. This operation has no implementation with this class. The implementation must be provided by a subclass.

Important

Any class with an abstract operation must itself be declared abstract.

- `final`The implementation of this operation must not be overridden by any subclass.

- `root`. The declaration of this operation must not override a declaration of the operation from a superclass.

- `query`. This indicates that the operation must have no side effects (i.e. it must not change the state of the system). It can only return a value.

Caution

Operations for user defined datatypes must always check this modifier.

- `static`. There is only one instance of this operation associated with the class (as opposed to one for each instance of the class). This is the `OwnerScope` attribute of a `Feature` metaclass within UML. Any operation declared static is shown underlined on the class diagram.

Concurrency Radio box, with entries `sequential`, `synchronized` and `concurrent`.

- `sequential`. Only one call to an instance (of the class with the operation) may be outstanding at any one time. There is no protection, and no guarantee of behavior if the system violates this rule.

- `synchronized`. Multiple calls to one instance are allowed, but only one may execute at a time. The others are blocked. In the UML 1.3 standard this form of concurrency is known as `guarded`.

Caution

It is up to the system designer to ensure deadlock cannot occur. It is the responsibility of the operation to implement the blocking behavior (as opposed to the system).

- `concurrent`. Multiple calls to one instance may execute at the same time. The operation is responsible for ensuring correct behavior. This must be managed even if there are other sequential or synchronized (guarded) operations executing at the time.

Parameters Text area, with entries for all the parameters of the operation (see Section 16.8). A new operation is always created with one new parameter, `return` to define the return type of the operation.

Button 1 click on any of the parameters navigates to that parameter. button 2 click brings up a pop up menu with four entries.

- `Open`. Navigate to the selected parameter.

- `Add`. Create a new parameter and navigate to it.

- `Move Up`. Only available where there are two or more parameters, and the parameter selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more parameters listed, and the parameter selected is not at the bottom. It is moved one down the list.

Raised Signals Text area, with entries for all the signals (see Section 16.9) that can be raised by the operation.

Caution

ArgoUML at present has limited support for signals. In particular they are not linked to signal events that could drive state machines.

Button 1 click on any of the parameters navigates to that parameter. button 2 click brings up a pop up menu with four entries.

- `Open`. Navigate to the selected signal.

- `Add`. Create a new signal and navigate to it.

- `Move Up`. Only available where there are two or more signals, and the signal selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more signals listed, and the signal selected is not at the bottom. It is moved one down the list.

16.8. Parameter

A parameter is a variable that can be passed. In the UML metamodel it is a sub-class of ModelElement.

A parameter is represented on within the operation declaration in the operation compartment of a class.

> *namename* : *typetype*

namename is the name of the parameter.

typetype is the type (UML datatype, class or interface) of the parameter.

The exception is any parameter representing a return value, whose type only is shown at the end of the operation declaration.

16.8.1. Parameter Details Tabs

The details tabs that are active for parameters are as follows.

ToDoItem   Standard tab.

Properties   See Section 16.8.2 and Section 16.8.3 below.

Documentation Standard tab.

Style   Standard tab, but relating to the owning class, rather than the parameter. The tick boxes, Attributes and Operations allow the attributes and operations compartments of the class to be shown (the default) or hidden. The Bounds: field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the Bounds: field. Take care when typing that you do get the values you want in place.

Source   Standard tab. This contains a declaration for the parameter.

Constraints   Standard tab. There are no standard constraints defined for Parameter within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, Parameter has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the parameter is redundant—it can be formally derived from other elements, or false meaning it cannot.

Caution

A derived parameter is a meaningless concept.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element

documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.8.2. Parameter Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Add parameter   This creates a new parameter for the for the same operation as the current parameter, navigating immediately to the properties tab for that parameter.

Tip

This is a convenient way to add a series of parameters for the same operation.

Delete   This deletes the parameter from the model

Warning

This is a deletion from the model *notnot* just the diagram. Since parameters have no meaning independent of their owning operation, it makes no sense to delete them just from the model. If desired the whole operation compartment can be hidden on the diagram using the *stylestyle* tab (see Section 16.8.2) or the button 2 pop up menu for the class on the diagram.

16.8.3. Property Fields For Parameter

Name   Text box. The name of the parameter. By convention, the name of a parameter has a leading lower case letter, with words separated by "bumpy caps".

Note

The ArgoUML critics do not complain about parameter names that do not have an initial lower case letter.

Stereotype Drop down selector. There are no UML standard stereotypes for Parameter. ArgoUML provides the stereotypes, machine, organization and person, and rather strangely realize (a stereotype of Abstraction).

Caution

ArgoUML provides the stereotype `realize` for parameters. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Owner   Text box. Records the operation which contains this parameter.

Button 1 click on the entry will navigate to the operation. Button 2 click provides a pop up menu with a single entry, `Open`, which will also navigate to the operation.

Type   Drop down selector. The type of this parameter. This can be any UML `Classifier`, although in practice only `Class`, `DataType` or `Interface` make any sense.

Note

A type must be declared (it can be `void`, but this only makes sense for a return parameter). By default ArgoUML supplies `int` as the type the first time a parameter is created, and thereafter the type of the most recently created parameter.

Navigate Type This will navigate you to the property panel for the currently selected type. (see Section 16.5, Section 16.3 and Section 16.14).

Initial Value Text box with drop down. This allows you to set an initial value for the parameter if desired (this is optional). The drop down menu provides access to the common values 0, 1, 2 and `null`.

Caution

This only makes sense for `out` or `return` parameters.

Kind   Radio box, with entries `in/out`, `in`, `out` and `return`.

- `in/out`. The parameter is used both to pass values in and to pass results back out of the operation.

Note

This is the default for any new parameter.

- `in`. The parameter is used only to pass values in to the operation.

- `out`. The parameter is used only to pass values back from the operation.

- `return`. The parameter is a return result from the call.

Note

There is nothing to stop you declaring more than one return parameter (some programming languages support this concept), but all except the first appear in the list of parameters passed in on the diagram, which is confusing.

Tip

The name of the return parameter does not appear on the diagram, but it is convenient to give it an appropriate name (such as the default `return` to identify it in the list of parameters on the operation property tab.

16.9. Signal

A signal is a specification of an asynchronous stimulus communicated between instances. In the UML meta-model it is a sub-class of `Classifier`.

Within ArgoUML signals are not fully handled. Their value is when they are received as *signal eventssignal events* driving the asynchronous behavior of state machines and when associated with *send actionssend actions* in state machines and messages for collaboration diagrams.

Tip

In general there is limited value at present in defining signals within ArgoUML. It may prove more useful to define signals as classes, with a (user defined) stereotype of «`signal`» as suggested in the UML 1.3 standard. This allows any dependency relationships between signals to be shown.

16.9.1. Signal Details Tabs

The details tabs that are active for signals are as follows.

`ToDoItem`	Standard tab.
`Properties`	See Section 16.9.2 and Section 16.9.3 below.
`Documentation`	Standard tab.
`Style`	Standard tab, but relating to the owning class, rather than the signal. The tick boxes, `Attributes` and `Operations` allow the attributes and operations compartments of the class to be shown (the default) or hidden. The `Bounds:` field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

`Source`	Standard tab. There is nothing generated for a signal.
`Constraints`	Standard tab. There are no standard constraints defined for `Signal` within the UML meta-model.
`Tagged Values`	Standard tab. In the UML metamodel, `Signal` has the following standard tagged values defined.

- `persistence` (from the superclass, `Classifier`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- `semantics` (from the superclass, `Classifier`). The value is a specification of the semantics of the signal.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the signal is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived signals still have their value in analysis to introduce useful names or concepts, and in design to

avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.9.2. Signal Property Toolbar

`Go up`	Navigate up through the package structure.
`Go back`	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Go forward`	Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Add operation`	This creates a new operation (see Section 16.7) for the signal, navigating immediately to the properties tab for that operation.

Caution

Although signals are very similar to classes, it makes little sense to define operations for them.

`Add attribute`	This creates a new attribute (see Section 16.6) for the signal, navigating immediately to the properties tab for that attribute.

Note

The attributes are effectively the parameters of the signal.

`New signal`	This creates a new signal, navigating immediately to the properties tab for that signal.

Note

ArgoUML uses the icon for a class for this button, rather than the icon for a signal.

Caution

The signal is not associated with the same operation as the original signal, so there appears no way to use it in the model.

`Delete`	This deletes the signal from the model

224

Warning

This is a deletion from the model. ArgoUML at present has no way of referencing signals independently of their owning operation, and so once deleted they can not be accessed.

16.9.3. Property Fields For Signal

Name   Text box. The name of the signal. From their similarity to classes, by convention, the name of a signal has a leading upper case letter, with words separated by "bumpy caps".

Note

The ArgoUML critics do not complain about signal names that do not have an initial upper case letter.

Stereotype Drop down selector. Signal is provided by default with the UML standard stereotypes for its parent in the UML meta-model, `Classifier` (`metaclass`, `powerType`, `process`, `thread` and `utility`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`, and rather strangely `realize` (a stereotype of `Abstraction`).

Caution

ArgoUML provides the stereotype `realize` for parameters. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate to Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Extends Text box. Lists any class that generalizes this class (see Section 16.13).

Caution

ArgoUML is incorrect in using the designation "Extends" for this field. It should correctly be called "Generalizations".

By default this field is shown with the entry `none`. Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization and navigate to that new generalization.

Caution

ArgoUML creates the generalization with the signal as child, but no parent, and no menu for adding a parent. The name will always appear as `null`.

- `Delete`. Removes the link to the selected generalization.

Note

This only removes the link to the generalization. To remove the generalization from the model you must navigate to it (or select it on the diagram or navigation pane) and then delete it.

- `Move Up`. Only available where there are two or more generalizations, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the general-

ization selected is not at the bottom. It is moved one down the list.

Caution

ArgoUML allows you to create generalizations, but not to specify parents, so they are of no use at all.

Tip

To specify detailed properties of signals define them as classes with the stereotype «signal».

Implements Text box. Lists any class or interface that is realized by this signal. (see Section 16.15).

Caution

ArgoUML is incorrect in using the designation "Extends" for this field. It should correctly be called "Realizes".

By default this field is shown with the entry none. Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- Open. Navigate to the selected abstraction.

- Add. Add a new abstraction.

Caution

ArgoUML creates the abstraction with the signal as client, but no supplier, and no menu for adding a supplier. The name will always appear as [empty].

- Delete. Removes the link to the selected abstraction.

Note

This only removes the link to the abstraction. To remove the abstraction from the model you must navigate to it (or select it on the diagram or navigation pane) and then delete it.

- Move Up. Only available where there are two or more abstractions, and the abstraction selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more abstractions listed, and the abstraction selected is not at the bottom. It is moved one down the list.

Caution

ArgoUML allows you to create abstractions, but not to specify suppliers, so they are of no use at all. Since signals should not have operations or attributes, abstraction seems a meaningless concept.

Modifiers Check box, with entries public, abstract, final and root.

- public has identical meaning to the public declaration in Java. It indicates whether this signal is restricted to use in its own package (the default) or can be used by anyone.

Caution

With the limited support for signals within ArgoUML, signals are effectively tied to their owning operation anyway.

- abstract is used to declare that this signal cannot be instantiated, but must always be specialized.

226

Caution

ArgoUML provides no mechanism for defining specialization between signals.

- `final` indicates that this signal cannot be further specialized, while `root` indicates it can have no generalization.

Caution

ArgoUML provides no mechanism for defining specialization between signals, so all should be declared both `final` and `root`.

Namespace Text box. Records the namespace for the class. This is the package hierarchy of the class which owns the operation which owns this signal.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Derived Text area. Lists any derived signals, i.e those for which this signal is a generalization. There is a single entry shown by default, `none`.

Button 1 click has no effect, button 2 gives a pop up menu with a single entry which is always grayed out.

Note

This field has no use for a signal.

Operations Text area. Lists all the operations (see Section 16.7) defined on this signal. Button 1 click takes you to the selected operation, button 2 click brings up a pop up menu with five entries.

- `Open`. Navigate to the selected operation.

- `Add`. Add a new operation and navigate to it.

- `Delete`. Removes the selected operation from the model.

- `Move Up`. Only available where there are two or more operations, and the operation selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more operations listed, and the operation selected is not at the bottom. It is moved one down the list.

Caution

Signals should not have any operations.

Attributes Text area. Lists all the attributes (see Section 16.6) defined for this class. These are effectively the parameters to the signal. Button 1 click takes you to the selected attribute, button 2 click brings up a pop up menu with five entries.

- `Open`. Navigate to the selected attribute.

- `Add`. Add a new attribute and navigate to it.

- `Delete`. Removes the selected attribute from the model.

- `Move Up`. Only available where there are two or more attributes, and the attribute selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more attributes listed, and the attribute selected is not at the bottom. It is moved one down the list.

Associations area. Lists all the associations (see Section 16.10) defined on this signal. Button 1 click takes you to the selected association, button 2 click brings up a pop up menu with five entries.

- Open. Navigate to the selected association.

- Add. Add a new association and navigate to it.

- Delete. Removes the selected association from the model.

- Move Up. Only available where there are two or more associations, and the association selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more associations listed, and the association selected is not at the bottom. It is moved one down the list.

Caution

Signals should not have any associations—the concept is meaningless.

16.10. Association

An association on a class diagram represents a relationship between classes, or between classes and an interface.

Caution

ArgoUML is restricted to binary associations at present. See Section 16.10.1 for more on this.

Associations are described under use case diagrams (see Section 15.4). There are a few additional issues specific to their use on class diagrams.

Associations are permitted between interfaces and classes, but UML 1.3 specifies they must only be navigable toward the interface—in other words the interface cannot see the class. ArgoUML will draw such associations with the appropriate navigation.

As with use case diagrams, associations with the tag derived have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

As with use case diagrams, associations are often not named, since their meaning is obvious from the context.

As with use case diagrams, there is little value in creating more than two association ends, since ArgoUML does not support multi-way associations.

16.10.1. Three-way and Greater Associations

UML 1.3 provides for associations that are governed by a third *associative classassociative class*. ArgoUML provides no support for multi-way associations or associative classes at this stage.

16.11. Association End

Association ends are described under use case diagrams (see Section 15.5).

16.12. Dependency

Dependencies are described under use case diagrams (see Section 15.10).

16.13. Generalization

Generalization is described under use case diagrams (see Section 15.6).

Note

Within the context of classes, generalization and specialization are the UML terms describing class inheritance.

16.14. Interface

An interface is a set of operations characterizing the behavior of an element. It can be usefully thought of as an abstract class with no attributes. In the UML metamodel it is a sub-class of `Classifier` and through that `GeneralizableElement`.

A class is represented on a class diagram as a rectangle with two horizontal compartments. The top compartment displays the interface name (and above it «interface») and the second any operations. Unlike a class, the operations compartment may not be hidden.

16.14.1. Interface Details Tabs

The details tabs that are active for interfaces are as follows.

ToDoItem   Standard tab.

Properties   See Section 16.14.2 and Section 16.14.3 below.

Documentation Standard tab.

Style   Standard tab. The Bounds: field defines the bounding box for the package on the diagram.

Warning

There are some bugs remaining in entering data in the Bounds: field. Take care when typing that you do get the values you want in place.

Source   Standard tab. This contains a template for the interface declaration (with JavaDoc header) and declarations of associated interfaces.

Constraints   Standard tab. There are no standard constraints defined for Interface within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, Interface has the following standard tagged values defined.

- persistence (from the superclass, Classifier). Values transitory, indicating state is destroyed when an instance is destroyed or persistent, marking state is preserved when an instance is destroyed.

Warning

Since interfaces are by definition abstract, they can have no instance, and so this tagged value must refer to the properties of the realizing class.

- semantics (from the superclass, Classifier). The value is a specification of the semantics of the interface.

- derived (from the superclass, ModelElement). Values true, meaning the interface is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived interfaces still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.14.2. Interface Property Toolbar

Go up — Navigate up through the package structure.

Go back — Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward — Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Add operation — This creates a new operation (see Section 16.7) within the interface, navigating immediately to the properties tab for that operation.

Delete — This deletes the interface from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an interface from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

16.14.3. Property Fields For Interface

Name Text box. The name of the interface. The name of an interface has a leading capital letter, with words separated by "bumpy caps".

Note

Unlike classes, the ArgoUML critics will not complain about interface names that do not have an initial capital.

Stereotype Drop down selector. Interface is provided by default with the UML standard stereotypes for the parent meta-class, `Classifier` (`metaclass`, `powerType`, `process`, `thread` and `utility`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`, and rather strangely `realize` (a stereotype of `Abstraction`).

Caution

ArgoUML provides the stereotype `realize` for interfaces. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate to stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace Text box. Records the namespace for the interface. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Generalizations Text area Lists any interface that *generalizes* this interface.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the interface (*but not the generalization itself*) is removed.

Tip

This *does not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

Modifiers Check box, with entries `public`, `abstract`, `final` and `root`.

- `public` has identical meaning to the `public` declaration in Java. It indicated whether this interface is restricted to use in its own package (the default) or can be used by anyone.

- `abstract` is used to declare that this interface cannot be instantiated, but must always be specialized. The name of an abstract interface is displayed in italics on the diagram.

Caution

This is meaningless, since by definition an interface is an abstract entity. The UML 1.3 standard offers no clarification.

- `final` indicates that this interface cannot be further specialized, while `root` indicates it can have no generalizations.

`Associations` Text box. Lists any associations (see Section 16.10) connected to this interface.

Note

Associations between classes and interfaces *must must* be navigable *only only* from the class to the interface. ArgoUML will create associations between classes and interfaces with the correct navigability.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- `Open`. Navigate to the selected association.

- `Add`. Add a new association.

Tip

New associations can usually more easily be added from the interface diagram.

- `Delete`. Removes the link to the selected association.

Note

This only removes the link to the association. To remove the association from the model you must navigate to it (or select it on the diagram or navigation pane) and then delete it.

- `Move Up`. Only available where there are two or more associations, and the association selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more associations listed, and the association selected is not at the bottom. It is moved one down the list.

`Implemented by` Text box. Lists any classes or interfaces (see Section 16.10) which are an abstraction of this interface (i.e. which this interface realizes).

Note

This field should probably be correctly labeled `Implements`

Warning

Although the UML 1.3 standard does not appear to forbid this, ArgoUML does not support abstraction of interfaces.

This entry is always shown as `none`. Button 1 click has no effect, and button 2 gives a pop up menu with five entries all grayed out.

`Specializations` Text box. Lists any specialized interface (i.e. for which this interface is a generalization.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- Delete. The link from the generalization to the interface (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

Operations Text area. Lists all the operations (see Section 16.7) defined on this interface. Button 1 click takes you to the selected operation, button 2 click brings up a pop up menu with five entries.

- Open. Navigate to the selected operation.

- Add. Add a new operation and navigate to it.

- Delete. Removes the selected operation from the model.

- Move Up. Only available where there are two or more operations, and the operation selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more operations listed, and the operation selected is not at the bottom. It is moved one down the list.

Caution

All operations on an interface *mustmust* be public. The ArgoUML critics will complain if this is not the case.

16.15. Abstraction

An abstraction is a dependency relationship joining two artifacts within the model at different levels of abstraction. Within ArgoUML it is principally used through its specific stereotype realize to define realization dependencies, which link artifacts that *specifyspecify* behavior to the corresponding artifacts that *implementment* the behavior.

In the UML metamodel Abstraction is a sub-class of Dependency and through that Relationship.

An abstraction with stereotype realize is represented on a class diagram as a dotted line with a solid white head at the specifying end. Note that all other stereotypes of abstraction should be represented using an open arrow head, but this is not supported by ArgoUML

16.15.1. Abstraction Details Tabs

The details tabs that are active for abstractions are as follows.

ToDoItem   Standard tab.

Properties   See Section 16.15.2 and Section 16.15.3 below.

Documentation Standard tab.

Style   Standard tab.

Note

The values for the bounds of the abstraction appear to have no meaning. Changing them has no effect on the diagram. This is sensible behavior, given the association is tied to particular artifacts.

Source   Standard tab. This contains the single word `realizes`.

Caution

An abstraction should not generate any code, so this entry is probably a mistake.

Constraints   Standard tab. There are no standard constraints defined for `Abstraction` within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, `Abstraction` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the abstraction is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived abstractions still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

16.15.2. Abstraction Property Toolbar

Go up   Navigate up through the package structure.

⇦Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the abstraction from the model

234

Warning

This is a deletion from the model *notnot* just the diagram. To delete an abstraction from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

16.15.3. Property Fields For Abstraction

Name   Text box. The name of the abstraction. There are no constraints on the name of an abstraction, which is not shown on any diagram.

Stereotype   Drop down selector. Abstraction is provided by default with the UML standard stereotypes derive, realize, refine and trace. ArgoUML also provides the additional stereotypes, machine, organization and person.

Caution

ArgoUML automatically selects the stereotype realize when an abstraction is created. The user is free to change the stereotype to use the abstraction to indicate for example a trace relationship. However ArgoUML will not alter the representation on the diagram accordingly.

Navigate to stereotype   Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see

Section 16.4).

Namespace   Text box. Records the namespace for the abstraction. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Supplier   Text area. Lists the artifact that is the supplier end of this abstraction (for a realization this is the end providing the implementation).

Note

Although this is a text area there is no mechanism for adding more than one supplier.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected artifact.

- Move Up. Since there is only ever one supplier this is always grayed out.

- Move Down. Since there is only ever one supplier this is always grayed out.

Clients   Text area. Lists the artifact that is the client end of this abstraction (for a realization this is the end providing the specification).

Note

Although this is a text area there is no mechanism for adding more than one client.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected artifact.

- Move Up. Since there is only ever one client this is always grayed out.

- Move Down. Since there is only ever one client this is always grayed out.

Chapter 17. ArgoUML Sequence Diagram Artifact Reference

Table of Contents

17.1. Introduction

This chapter describes each artifact that can be created within a sequence diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

There is a close relationship between this material and the `Properties` tab of the details pane (see Section 10.4). That section covers properties in general, in this chapter they are linked to specific artifacts.

Caution

Sequence diagrams are not fully developed yet in ArgoUML. Many aspects are not fully implemented, or may not behave as expected.

Figure 17.1 shows a sequence diagram with all possible artifacts displayed.

Figure 17.1. Possible artifacts on a sequence diagram.
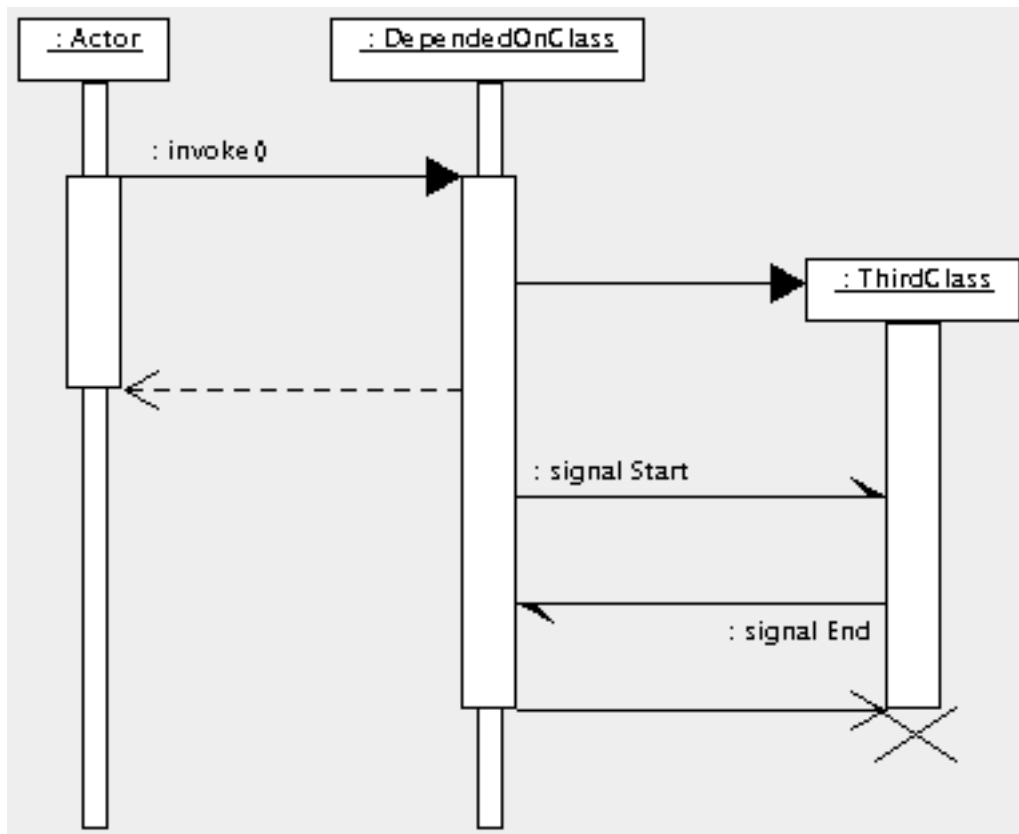
17.2. Object

An object is an instance of a class. In the UML metamodel `Object` is a sub-class of `Instance`. Within a sequence diagram objects may be used to represent a specific instance of a class. Unlike collaboration diagrams (see Chapter 19), sequence diagrams cannot show generic behavior between classifier roles.

An object is represented on a sequence diagram in ArgoUML as a plain box labeled with the object name (if any) and class, separated by a colon (:). As links with stimuli to and from other objects are added, a time line grows down from the object. This is thin where the object does not have control and thick where it does.

Caution

The current release of ArgoUML shows interactions between objects, although the UML standard for sequence diagrams is for interaction between instances of any classifier).

However the actual implementation in ArgoUML permits any classifier to be used with the object, and so the diagram can successfully represent instances of actors for example as well as classes.

17.2.1. Object Details Tabs

The details tabs that are active for objects are as follows.

ToDoItem   Standard tab.

Properties   See Section 17.2.2 and Section 17.2.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the object notionally define the bounding box of

237

the object and its time line. However if you change them it will have no effect, and the original values will be reset when you next revisit the tab.

Caution

In the current release of ArgoUML changing the values of this field is possible, but will make a (temporary) mess of the screen display. Selecting the object on the screen causes the original values to be restored and the screen refreshed.

Source   Standard tab, but with no contents.

Caution

An object should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. In the UML metamodel, `Object` has the following standard constraints defined (through its parent, `Instance`).

- `destroyed`. Applied to an object that is destroyed during the execution.

- `new`. Applied to an object that is created during the execution.

- `transient`. Applied to an object that is both created and destroyed during the execution.

Tagged Values Standard tab. In the UML metamodel, `Object` has the following standard tagged values defined.

- `persistence` (from the superclass, `Instance`. Showing the permanence of the state information associated with the object. Values `transitory` (state is destroyed when the object is destroyed) and `persistent` (state is preserved when the object is destroyed).

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the object is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived objects still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

17.2.2. Object Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

238

⇨Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✖Delete   This deletes the object from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an object from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

17.2.3. Property Fields For Object

Name   Text box. The name of the object. By convention object names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Classifier Drop down selector. The name of the classifier of which this is an object.

Caution

In the current release of ArgoUML the drop down selector will include *allall* classifiers (i.e. interfaces, actors, use cases and datatypes as well), which is what is wanted on the diagram, although it should properly be called an instance, rather than an object. In practice only instances of classes and actors make much sense.

Note

In the current release of ArgoUML the same graphical presentation is used, even if the object is actually representing an instance of an actor (when a stick-man would be more usual).

Stereotype Drop down selector. Object has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, machine, organization and person.

Caution

ArgoUML also provides the stereotype realize for objects. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Navigate Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace Text box. Records the namespace for the object. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Stimuli Sent Text area. Lists the stimuli sent to this object.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- Open. Navigate to the selected stimulus.

239

- Add. Add a new stimulus.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can only be added from the diagram.

- Delete. Delete the selected stimulus.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can only be deleted from the diagram.

- Move Up. Only available where there are two or more stimuli, and the stimulus selected is not at the top. It is moved one up the list.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can not be moved in the order. To change the order a stimulus must be deleted and then recreated on the diagram.

- Move Down. Only available where there are two or more stimuli listed, and the stimulus selected is not at the bottom. It is moved one down the list.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can not be moved in the order. To change the order a stimulus must be deleted and then recreated on the diagram.

StimuliReceived. Text area lists the stimuli received by this object.

Warning

In the current release of ArgoUML this field is completely broken. No entries are every shown.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- Open. Navigate to the selected stimulus.

- Add. Add a new stimulus.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can only be added from the diagram.

- Delete. Delete the selected stimulus.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can only be deleted from the diagram.

- Move Up. Only available where there are two or more stimuli, and the stimulus selected is not at the top. It is moved one up the list.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can not be moved in the order. To change the order a stimulus must be deleted and then recreated on the diagram.

- `Move Down`. Only available where there are two or more stimuli listed, and the stimulus selected is not at the bottom. It is moved one down the list.

Caution

In the current release of ArgoUML this entry is always grayed out. Stimuli can not be moved in the order. To change the order a stimulus must be deleted and then recreated on the diagram.

17.3. Stimulus

A stimulus is a communication between two instances and is generated by an action. On a sequence diagram is associated with a link—an instance of an association linking two object instances. In the UML metamodel `Stimulus` is a sub-class of `ModelElement`.

The link (see Section 17.9) associated with a stimulus is represented on a sequence diagram in ArgoUML as an arrow between the time lines of the object instances (or the object head in the case of stimulus create, described below) labeled with the name of the action (if any), and the action, separated by a colon (:). The type of line and arrowhead depends on the type of action that generated the stimulus:

- `Stimulus Call`. Generated by a call action, itself the result of an operation of a class. Shown as a solid line with a solid arrowhead to the time line of the object instance receiving the stimulus.

- `Stimulus Create`. Generated by a create action for the class for which an instance is to be created Shown as a solid line with a solid arrowhead to the object head of the object instance being created.

- `Stimulus Destroy`. Generated by a destroy action of the originating object. Shown as a solid line with an open arrowhead terminating in a diagonal cross at the end of the time line of the receiving (destroyed) object instance.

- `Stimulus Send`. Generated by a send action, the result of a signal raised by an operation of the sending object instance and handled by the receiving object instance. Shown as a solid line with half an open arrowhead.

- `Stimulus Return`. Generated by an object instance that has received an earlier call stimulus and is returning a result to the calling object instance. Shown as a dotted line with an open arrowhead.

Note

ArgoUML does not allow you to create stimuli directly, but instead provides tools to create stimuli of each of the five types above.

Caution

In the current release of ArgoUML there is no way to show a terminate action where an object instance destroys itself. One way is to draw a destroy action that loops back to the object itself, give it an action with no name and use the style tab to set an invisible line, but this still leaves the arrow head showing, which is unsightly. It is also semantically incorrect anyway to use a destroy action to represent a terminate action.

17.3.1. Stimulus Details Tabs

The details tabs that are active for stimuli are as follows.

ToDoItem   Standard tab.

Properties   See Section 17.3.2 and Section 17.3.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the stimulus notionally define the bounding box of the stimulus and its time line. However if you change them it will have no effect, and the original values will be reset when you next revisit the tab.

Altering the Fill and Shadow entries has no effect. Rather bizarrely you can set the Line entry and it will draw a line around the signal, which is not a standard UML representation.

Tip

To change the color of the line, you should select the associated link (click on it a little way from the stimulus) and use its style tab (see Section 17.9).

Caution

In the current release of ArgoUML changing the values of the Bounds field is possible, but will make only a temporary change to the position of the stimulus. Selecting any artifact on the screen causes the stimulus to return to its original position and the original values to be restored.

Source   Standard tab, but with no contents.

Caution

A stimulus should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. There are no standard constraints defined for Stimulus within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, Stimulus has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the stimulus is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived stimuli still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

17.3.2. Stimulus Property Toolbar

Go up   Navigate up through the package structure.

⇦Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this

button will take you.

⇨Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the stimulus from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an stimulus from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

17.3.3. Property Fields For Stimulus

Name   Text box. There is no convention for naming stimuli, and it is quite normal to leave them unnamed. The action is sufficient identification.

Tip

It is sometimes useful to give simple names to stimuli, so they can be referred to in attached notes giving timing constraints.

Action Text box. This is used to identify the action that generated the stimulus.

Caution

The current release of ArgoUML only implements actions as textual descriptions.

As a practical convention it is suggested that call actions are shown as the name of the operation generating the action with any arguments in parentheses and that send actions are shown as the name of the signal generating the action with any arguments in parentheses. Return actions should be shown as the expression for the value they return, or empty otherwise. Create and destroy actions should be left empty, since they are implied by their representation.

StereotypeDrop down selector. Stimulus has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, machine, organization and person.

Caution

ArgoUML also provides the stereotype realize for stimuli. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Navigate Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Sender Text box. Identifies the instance which sent this stimulus.

Button 1 click takes you to the sender instance, button 2 gives a pop up menu with one entry.

• Open. Navigate to the selected sender instance.

Receiver Text box. Identifies the instance which receives this stimulus.

Button 1 click takes you to the receiver instance, button 2 gives a pop up menu with one entry.

- `Open`. Navigate to the selected receiver instance.

Warning

In the current release of ArgoUML this field is broken. It always shows the entry `none` and the pop-up menu is grayed out.

`Namespace` Text box. Records the namespace for the stimulus. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

17.4. Stimulus Call

This tool creates a stimulus associated with a call action on the diagram, creating at the same time the associated link between sender and receiving instances.

All details tabs and properties are identical to to that of stimulus in general (see Section 17.3). Its graphical representation on the diagram is that of a stimulus associated with a call action, i.e. a solid line with a solid arrow head.

Note

Because the current release of ArgoUML does not fully implement actions, there is no enforcement of the relationship to a call action.

17.5. Stimulus Create

This tool creates a stimulus associated with a create action on the diagram, creating at the same time the associated link between sender and receiving instances.

All details tabs and properties are identical to to that of stimulus in general (see Section 17.3). Its graphical representation on the diagram is that of a stimulus associated with a create action, i.e. a solid line with a solid arrow head terminating at the head of the created instance.

Note

Because the current release of ArgoUML does not fully implement actions, there is no enforcement of the relationship to a create action.

17.6. Stimulus Destroy

This tool creates a stimulus associated with a destroy action on the diagram, creating at the same time the associated link between sender and receiving instances.

All details tabs and properties are identical to to that of stimulus in general (see Section 17.3). Its graphical representation on the diagram is that of a stimulus associated with a destroy action, i.e. a solid line with an open arrow head terminating at a cross at the bottom of the destroyed instance's time line.

Note

Because the current release of ArgoUML does not fully implement actions, there is no enforcement of the relationship to a destroy action.

17.7. Stimulus Send

This tool creates a stimulus associated with a send action on the diagram, creating at the same time the associated link between sender and receiving instances.

All details tabs and properties are identical to to that of stimulus in general (see Section 17.3). Its graphical representation on the diagram is that of a stimulus associated with a send action, i.e. a solid line with half an open arrow head.

Note

Because the current release of ArgoUML does not fully implement actions, there is no enforcement of the relationship to a send action.

17.8. Stimulus Return

This tool creates a stimulus associated with a return action on the diagram, creating at the same time the associated link between sender and receiving instances.

All details tabs and properties are identical to to that of stimulus in general (see Section 17.3). Its graphical representation on the diagram is that of a stimulus associated with a return action, i.e. a dotted line with an open arrow head.

Note

Because the current release of ArgoUML does not fully implement actions, there is no enforcement of the relationship to a return action.

17.9. Link

A link is an instance of an association. In the UML metamodel `Link` is a sub-class of `Instance`. Within a sequence diagram links are created indirectly when an associated stimulus is created.

An link is represented on a sequence diagram in ArgoUML as a line connecting the instances concerned. However on a sequence diagram the representation is modified to reflect the type of action associated with the stimulus carried on the link (see Section 17.3).

17.9.1. Link Details Tabs

The details tabs that are active for links are as follows.

ToDoItem   Standard tab.

Properties   See Section 17.9.2 and Section 17.9.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the link notionally define the bounding box of the link. However if you change them it will have no effect.

Caution

In the current release of ArgoUML changing the values of the line field will alter the color of the line, but *notnot* the arrow head.

Source   Standard tab, but with no contents.

Caution

A link should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. In the UML metamodel, Link has the following standard constraints defined
                (through its parent, Instance).

- destroyed. Applied to a link that is destroyed during the execution.

- new. Applied to a link that is created during the execution.

- transient. Applied to a link that is both created and destroyed during the execution.

Tagged Values Standard tab. In the UML metamodel, Link has the following standard tagged values de-
                fined.

- persistence (from the superclass, Instance. Showing the permanence of the state
  information associated with the link. Values transitory (state is destroyed when the
  link is destroyed) and persistent (state is preserved when the link is destroyed).

- derived (from the superclass, ModelElement). Values true, meaning the link is
  redundant—it can be formally derived from other elements, or false meaning it can-
  not.

Note

Derived links still have their value in analysis and design to introduce useful names or concepts, and in
design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element
documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

17.9.2. Link Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

Delete   This deletes the link from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an link from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

17.9.3. Property Fields For Link

Name   Text box. The name of the link. By convention link names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Stereotype   Drop down selector. Link has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, machine, organization and person.

Caution

ArgoUML also provides the stereotype realize for links. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Navigate Stereotype   Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace   Text box. Records the namespace for the link. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Chapter 18. ArgoUML State Diagram Artifact Reference

Table of Contents

18.1. Introduction

This chapter describes each artifact that can be created within a state diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

There is a close relationship between this material and the Properties Tab of the Details Pane (see Section 10.4). That section covers Properties in general, in this chapter they are linked to specific artifacts.

Note

The whole area of behavioral modeling is still under development in ArgoUML. Many of the facilities described in the UML standard have yet to be implemented or are not fully implemented in the current release.

Figure 18.1 shows a state diagram with all possible artifacts displayed.

Figure 18.1. Possible artifacts on a state diagram.
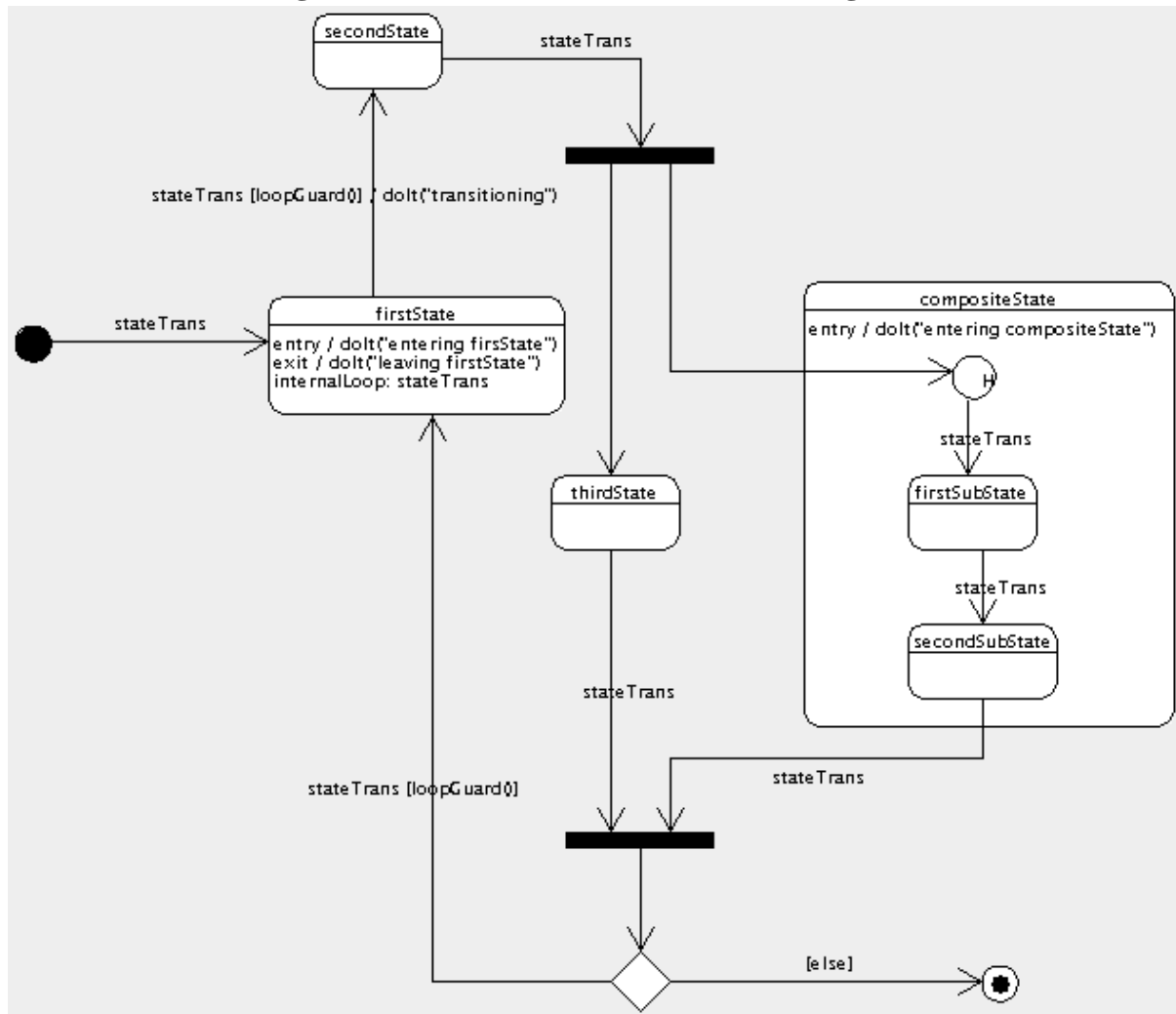
18.2. State

A state models a condition when some (usually implicit) invariant holds for the parent class. This may be a static situation, or some dynamic activity "in progress". Within the UML metamodel, `State` is a sub-class of `StateVertex`.

An state is represented on a state diagram in ArgoUML as a rectangle with rounded corners, with a horizontal line separating the name at the top from the description of the behavior below. The description of the behavior includes the entry and exit actions and any internal transitions.

Caution

The current release of ArgoUML does not record "Do activities" on the diagram.

18.2.1. State Details Tabs

The details tabs that are active for states are as follows.

ToDoItem   Standard tab.

Properties   See Section 18.2.2 and Section 18.2.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the state define the bounding box of the state.

Caution

In the current release of ArgoUML there are known problems with typing in the Bounds box, with the cursor jumping about. Take care that you enter the changes you intend when changing values

Caution

The Shadow field has no effect— in the current release of ArgoUML. There are no shadow effects provided for states on the diagram.

Source   Standard tab, containing just the name of the state.

Note

In the fullness of time this tab will contain code to implement the state (code for actions will be associated with the particular actions). This is not implemented in the current release of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for State within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, State has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the state is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived states still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.2.2. State Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the state from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an state from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

18.2.3. Property Fields For State

Name   Text box. The name of the state. By convention state names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Stereotype Drop down selector. State has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, machine, organization and person.

Caution

ArgoUML also provides the stereotype realize for states. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Navigate Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace Text box. Records the namespace for the state. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Incoming Text area. Lists the transitions that enter this state, including any internal transitions within the state (described below).

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected transition.

Outgoing Text area. Lists the transitions that leave this state, including any internal transitions within the state (described below).

Button 1 click takes you to the selected action, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected transition.

Entry-Action Text box. Lists the action (if any) to be invoked on entry to this state.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected action.

- Add. Add a new action.

- Delete. Delete the selected action.

Exit-Action Text box. Lists the action (if any) to be invoked on exit from this state.

Button 1 click takes you to the selected action, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected action.

- Add. Add a new action.

- Delete. Delete the selected action.

Do-Activity Text box. Lists the action (if any) to be invoked while in this state.

Button 1 click takes you to the selected action, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected action.

- Add. Add a new action.

- Delete. Delete the selected action.

Internal Transitions Text area. Lists the internal transitions of the state. Such transitions neither exit nor enter the state, so only the Do-Activity (see above) will be executed if such a transition is taken.

Button 1 click takes you to the selected transition, button 2 gives a pop up menu with two entries.

- Open. Navigate to the selected transition.

- Add. Add a new internal transition.

18.3. Action

An action specifies an executable statement and is an abstraction of a computational procedure that can change the state of the model. In the UML metamodel it is a child of both Model and ActionSequence. Since in the metamodel an action sequence is itself an action that is an aggregation of other actions, an action sequence may be used anywhere that an action may be.

There are a number of different types of action that are children of action within the UML metamodel.

- CreateAction. Associated with a classifier, this action creates an instance of that classifier.

- CallAction. Associated with an operation of a class, this action calls the given operation.

- ReturnAction. An action used to return a result to an earlier caller.

- SendAction. Associated with a signal, this action causes the signal to be raised.

- TerminateAction. Causes the invoking object to self-destruct.

- DestroyAction. Destroys the specified target object.

An action is represented by the text of its expression.

Caution

The current release of ArgoUML does not show"Do activities" on states, or effects on transitions.

18.3.1. Action Details Tabs

The details tabs that are active for actions are as follows.

ToDoItem   Standard tab.

Properties   See Section 18.3.2 and Section 18.3.3 below.

Documentation Standard tab.

Style   Standard tab. The contents of this tab relate to the owning artifact (state or transition), not the action, and any changes affect that owning artifact.

Source   Standard tab, containing the expression for the action.

Note

In the fullness of time this tab will contain code to implement the action. This is not implemented in the current release of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for Action within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, Action has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the action is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived actions still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.3.2. Action Property Toolbar

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

18.3.3. Property Fields For Action

Name   Text box. The name of the action. By convention action names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Expression Text box. The expression that defines the action.

Caution

The current release of ArgoUML only partially implements actions. As a practical convention it is suggested that call actions are shown as the name of the operation generating the action with any arguments in parentheses and that send actions are shown as the name of the signal generating the action with any arguments in parentheses. Return actions should be shown as the expression for the value they return, or empty otherwise. Create and destroy actions should shown as create(<target>) and destroy(<target>). Terminate action should be shown as terminate.

Language Text box. This is not part of standard UML, but in the partial implementation of actions currently available in UML it may be useful to indicate that the expression is written in a particular programming language.

18.4. Composite State

A composite state is a state that contains other states (known as sub-states), allowing hierarchical state machines to be constructed.

A composite state is represented on a state diagram in ArgoUML as a large rectangle with rounded corners, with a horizontal line separating the name at the top from the description of the behavior and the model of the sub-state machine below. The description of the behavior includes the entry and exit actions and any internal transitions.

Sub-states are placed within a composite machine by placing them entirely within the composite state when creating them for the first time in the editing pane.

The description of a composite state is almost identical to that of a state (see Section 18.2 and so is not duplicated here. The single difference is one additional field below the field for Stereotype, which is described as follows.

Subvertices   Text area. Lists the sub-states contained within this composite state.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected sub-state.

18.5. Transition

A transition is a directed relation between a source state (or composite state) and destination state (or composite state). Within the UML metamodel, Transition is a sub-class of ModelElement.

A transition is represented on a state diagram in ArgoUML as an arrow connecting the source to the destination state. Alongside the state is the name of the transition (if any) separated by a colon (:) from the trigger action,

any guard in square brackets (`[ ]`) separated by a slash (`/` from any effect expression.

18.5.1. Transition Details Tabs

The details tabs that are active for transitions are as follows.

`ToDoItem`	Standard tab.
`Properties`	See Section 18.5.2 and Section 18.5.3 below.
`Documentation`	Standard tab.
`Style`	Standard tab. The values for the bounds of the transition have no effect, since the position of the transition is defined by its end points.

Caution

In the current release of ArgoUML changing the color of the line does not change the color of the arrow head.

`Source`	Standard tab, containing just the name of the transition, any guard and any effect in the same format as the diagram.

Note

In the fullness of time this tab will contain any code to implement the transition (code for the effect and guard will be associated with their artifacts). This is not implemented in the current release of ArgoUML.

`Constraints`	Standard tab. There are no standard constraints defined for `Transition` within the UML metamodel.
`Tagged Values`	Standard tab. In the UML metamodel, `Transition` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the transition is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived transitions still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.5.2. Transition Property Toolbar

Go up	Navigate up through the package structure.
⇦ Go back	Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨ Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the transition from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a transition from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

18.5.3. Property Fields For Transition

Name   Text box. The name of the transition. By convention transition names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Stereotype   Drop down selector. Transition has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, machine, organization and person.

Caution

ArgoUML also provides the stereotype realize for transitions. This appears to be an error, since this stereotype properly belongs to the Abstraction metaclass.

Navigat ↑ Stereotype   If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Source   Text box. Lists the source state for the transition.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with one entry.

• Open. Navigate to the selected transition.

Target   Text box. Lists the target state for the transition.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with one entry.

• Open. Navigate to the selected transition.

Trigger   Text box. Lists the trigger event (if any) which invokes this transition.

Note

UML does not require there to be a trigger, if a guard is defined. The transition is taken immediately if the guard is true.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected event.

- Delete. Delete the selected event.

- Add. Add a new event. Grayed out if there is already an event defined.

Guard   Text box. Lists the guard expression (if any) which must be true before this transition can be taken.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected guard.

- Delete. Delete the selected guard.

- Add. Add a new guard. Grayed out if there is already a guard defined.

Effect   Text box. Lists the action (if any) to be invoked as this transition is taken.

Button 1 click takes you to the selected action, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected action.

- Delete. Delete the selected action.

- Add. Add a new action.

18.6. Call Event

An event is an observable occurrence. In the UML metamodel it is a child of ModelElement.

There are a number of different types of event that are children of event within the UML metamodel.

- SignalEvent. Associated with a signal, this event is caused by the signal being raised.

- CallEvent. Associated with an operation of a class, this event is caused by a call to the given operation. The expected effect is that the steps of the operation will be executed.

- TimeEvent. An event cause by expiration of a timing deadline.

- ChangeEvent. An event caused by a particular expression (of attributes and associations) becoming true.

An action is represented by its name.

Caution

The current release of ArgoUML does not show the parameters of the event.

Caution

The current release of ArgoUML has only partially implemented events, and currently indicates that they are always call events. In practice, since only the name and parameters can be specified, the call event tab can be used for any event. The name field should be used to indicate that it is a signal, timing or change event.

18.6.1. Call Event Details Tabs

The details tabs that are active for call events are as follows.

ToDoItem   Standard tab.

Properties   See Section 18.6.2 and Section 18.6.3 below.

Documentation Standard tab.

Style   Standard tab. The contents of this tab relate to the owning artifact (transition), not the call
                  event, and any changes affect that owning artifact.

Source   Standard tab, containing the name of the call event.

Note

In the fullness of time this tab will contain code to implement the call event. This is not implemented in
the current release of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for CallEvent within the UML
                  metamodel.

Tagged Values Standard tab. In the UML metamodel, Call Event has the following standard tagged val-
                  ues defined.

- derived (from the superclass, ModelElement). Values true, meaning the call
  event is redundant—it can be formally derived from other elements, or false meaning
  it cannot.

Note

Derived call events still have their value in analysis and design to introduce useful names or concepts,
and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element
documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.6.2. Call Event Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

Add parameter   This creates a new parameter for the for the event operation as the
                                   current parameter, navigating immediately to the properties tab for
                                   that parameter (see Section 16.8).

18.6.3. Property Fields For Call Event

Name   Text box. The name of the call event. By convention call event names start with a lower case letter and use bumpy caps to divide words within the name in the same way as operations.

Note

ArgoUML does not enforce this naming convention.

Tip

For call events it makes sense to use the name of the associated operation. For signal events it make sense to use the name of the signal, prefixed by [sig]. For time events use the time expression, prefixed by [time] and for change events the change expression, prefixed by [change].

Parameters Text area, with entries for all the actual parameter values of the event (see Section 16.8).

Button 1 click on any of the parameters navigates to that parameter. button 2 click brings up a pop up menu with four entries.

- Open. Navigate to the selected parameter.

- Add. Create a new parameter and navigate to it.

- Move Up. Only available where there are two or more parameters, and the parameter selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more parameters listed, and the parameter selected is not at the bottom. It is moved one down the list.

18.7. Guard

A guard is associated with a transition. At the time an event is dispatched, the guard is evaluated, and if false, its transition is disabled. In the UML metamodel, Guard is a child of ModelElement.

An guard is represented by the text of its expression in square brackets ([ ]).

18.7.1. Guard Details Tabs

The details tabs that are active for guards are as follows.

ToDoItem   Standard tab.

Properties   See Section 18.7.2 and Section 18.7.3 below.

Documentation Standard tab.

Style   Standard tab. The contents of this tab relate to the owning artifact (transition), not the guard, and any changes affect that owning artifact.

Source   Standard tab, containing the expression for the guard.

Note

In the fullness of time this tab will contain code to implement the guard. This is not implemented in the

current release of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for `Guard` within the UML meta-model.

Tagged Values Standard tab. In the UML metamodel, `Guard` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the guard is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived guards still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.7.2. Guard Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

18.7.3. Property Fields For Guard

Name   Text box. The name of the guard. By convention guard names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Expression Text box. The expression that defines the guard.

Language Text box. This is not part of standard UML, but in the partial implementation of guards currently available in UML it may be useful to indicate that the expression is written in a particular programming language.

Transition Text box, listing the transition that owns this guard.

Button 1 click on the transition navigates to that transition. button 2 click brings up a pop up menu with one entry.

- `Open`. Navigate to the selected transition.

18.8. Pseudostate

A pseudostate encompasses a number of different transient vertices on a state machine diagram. They do not have the properties of a full state and serve only as a connection point for transactions (but with some semantic value). Within the UML metamodel, `Pseudostate` is a sub-class of `StateVertex`.

The representation of a pseudostate on a state diagram in ArgoUML depends on the particular type of pseudostate: initial, deepHistory, shallowHistory, join, fork, junction and choice. ArgoUML does not allow you to place a pseudostate directly, but only through tools for the specific types of pseudostate. These are described in separate sections below (see Section 18.9, Section 18.11, Section 18.12, Section 18.13, Section 18.14 and Section 18.15).

Caution

The current release of ArgoUML does not provide support for a junction pseudostate.

18.8.1. Pseudostate Details Tabs

The details tabs that are active for pseudostates are as follows.

`ToDoItem` Standard tab.

`Properties` See Section 18.8.2 and Section 18.8.3 below.

`Documentation` Standard tab.

`Style` Standard tab.

Caution

In the current release of ArgoUML there are known problems with typing in the `Bounds` box, with the cursor jumping about. Take care that you enter the changes you intend when changing values

Caution

The `Shadow` field has no effect— in the current release of ArgoUML. There are no shadow effects provided for states on the diagram.

`Source` Standard tab, containing just the name of the pseudostate.

Note

In the fullness of time this tab will contain any code to implement the pseudostate. This is not implemented in the current release of ArgoUML.

`Constraints` Standard tab. There are no standard constraints defined for `Pseudostate` within the UML metamodel.

`Tagged Values` Standard tab. In the UML metamodel, `Pseudostate` has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the pseu-
  dostate is redundant—it can be formally derived from other elements, or false mean-
  ing it cannot.

Note

Derived pseudostates still have their value in analysis and design to introduce useful names or con-
cepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element
documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.8.2. Pseudostate Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this
button will take you.

Delete   This deletes the pseudostate from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a pseudostate from the diagram,
but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

18.8.3. Property Fields For Pseudostate

Name   Text box. The name of the pseudostate. By convention pseudostate names start with a lower case letter
        and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Tip

Pseudostate names are not shown on the diagram and it is not usually necessary to give them a name.

Kind   Radio buttons. Radio button indicating the type of pseudostate: junction, branch, fork, join,
        deep history and shallow history.

Note

In ArgoUML these buttons are defined by the type of pseudostate created and cannot be changed. As a consequence they are shown grayed out.

Incoming Text area. Lists any incoming transitions for the pseudostate.

Button 1 click takes you to the selected transition, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected transition.

Outgoing Text area. Lists any outgoing transitions for the pseudostate.

Button 1 click takes you to the selected transition, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected transition.

18.9. Initial State

The initial state is a pseudostate (see Section 18.8) representing the default state of a state machine (or composite state) when it is created. It is the state from which any initial transition is made.

As a consequence it is not permissible to have incoming transitions. ArgoUML will not let you create such transitions, and if you import a model that has such transitions, a critic will complain.

An initial state is represented on the diagram as a solid disc.

18.10. Final State

If a transition reaches a final state, it implies completion of the activity associated with that composite state, or at the top level, of the complete state machine. In the UML metamodel FinalState is a child of State.

Note

A final state is a true state (with all its attributes), *not* a pseudostate.

Completion at the top level implies termination (i.e. destruction) of the owning object instance.

The representation of a final state on the diagram is a circle with a small disc at its center.

18.10.1. Final State Details Tabs

The details tabs that are active for final states are as follows.

ToDoItem   Standard tab.

Properties   See Section 18.10.2 and Section 18.10.3 below.

Documentation Standard tab.

Style   Standard tab.

Caution

In the current release of ArgoUML there are known problems with typing in the `Bounds` box, with the cursor jumping about. Take care that you enter the changes you intend when changing values

Caution

The `Shadow` field has no effect— in the current release of ArgoUML. There are no shadow effects provided for states on the diagram.

Source   Standard tab, containing just the name of the final state, any guard and any effect in the same format as the diagram.

Note

In the fullness of time this tab will contain any code to implement the final state. This is not implemented in the current release of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for `Final State` within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, `Final State` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the final state is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived final states still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

18.10.2. Final State Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Delete   This deletes the final state from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a final state from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

18.10.3. Property Fields For Final State

`Name`   Text box. The name of the final state. By convention final state names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Tip

Final state names are not shown on the diagram and it is not usually necessary to give them a name.

`Stereotype`   Drop down selector. Final State has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for final states. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

`Navigate Icon for stereotype`   If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

`Incoming`   Text area. Lists any incoming transitions for the final state.

Button 1 click takes you to the selected transition, button 2 gives a pop up menu with one entry.

- `Open`. Navigate to the selected transition.

18.11. Branch

Branch is a pseudostate (see Section 18.8) which allows a dynamic choice of outgoing transitions. The chosen transition is that whose guard is true at the time of the transition (if more than one is true, one is selected at random).

Note

UML calls this sort of branch a `choice`. UML also defines a static branch, `junction`, based on static conditional expressions, which is not supported in the current release of ArgoUML

A branch is represented on the diagram as a solid (white by default) diamond.

18.12. Fork

Fork is a pseudostate (see Section 18.8) which splits a transition in to two or more concurrent transitions.

Caution

The outgoing transitions should not have guards. However the current release of ArgoUML will not en-

force this.

A fork is represented on the diagram as a solid (black by default) horizontal bar.

18.13. Join

Fork is a pseudostate (see Section 18.8) which joins two or more concurrent transitions into a single transition.

Caution

The incoming transitions should not have guards. However the current release of ArgoUML will not enforce this.

A join is represented on the diagram as a solid (black by default) horizontal bar.

18.14. Shallow History

Shallow History is a pseudostate (see Section 18.8) within a composite state, which causes control to be passed to the state that last had control within the composite state. A transition may show a default to state to use if the composite state has never been entered before.

The shallow history ignores any composite states within this composite states, dealing only with direct substates.

A shallow history is represented on the diagram as a circle containing the letter H.

18.15. Deep History

Shallow History is a pseudostate (see Section 18.8) within a composite state, which causes control to be passed to the state that last had control within the composite state. A transition may show a default to state to use if the composite state has never been entered before.

The deep history will transfer to a state within any hierarchy of composite states within this composite states if that is necessary.

A deep history is represented on the diagram as a circle containing the symbols H*.

Chapter 19. ArgoUML Collaboration Diagram Artifact Reference

Table of Contents

19.1. Introduction

This chapter describes each artifact that can be created within a collaboration diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

There is a close relationship between this material and the properties tab of the details pane (see Section 10.4). That section covers Properties in general, in this chapter they are linked to specific artifacts.

Caution

Collaboration diagrams are not fully developed yet in ArgoUML. Many aspects are not fully implemented, or may not behave as expected. In particular there are some serious problems with layout of the collaboration roles and messages.

Figure 19.1 shows a collaboration diagram with all possible artifacts displayed.

Figure 19.1. Possible artifacts on a collaboration diagram.
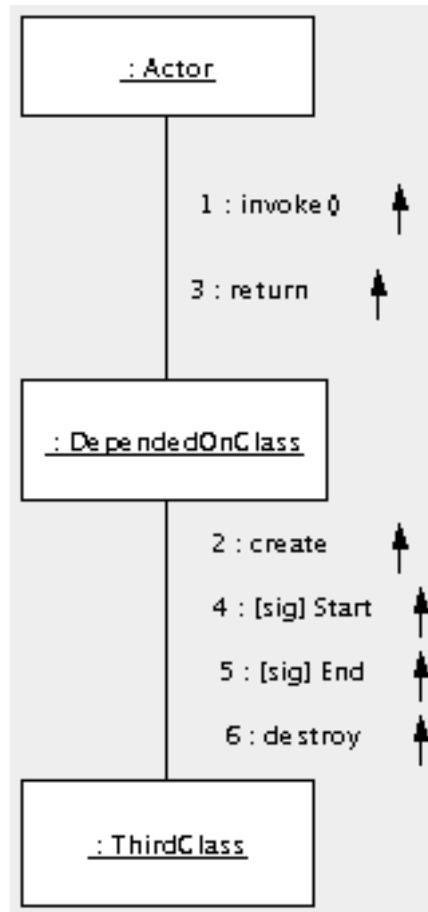
19.2. Classifier Role

A classifier role is a specialization of a classifier, used to show its behavior in a particular context. In the UML metamodel `Classifier Role` is a sub-class of `Classifier`. Within a collaboration diagram classifier roles may be used in one of two ways:

- To represent the classifier in a particular behavioral context (the *specification levelspecification level*); or

- to specify a particular instance of the classifier (the *instance levelinstance level*).

In this latter form, classifier roles are identical to the instances used in sequence diagrams (see Chapter 17) and a collaboration diagram shows the same information as the sequence diagram, but in a different presentation.

Caution

A collaboration diagram should not mix classifier roles used as the specifier level and the instance level.

A classifier role is represented on a sequence diagram in ArgoUML as a plain box labeled with the classifier role name (if any) and classifier, separated by a colon (:).

Caution

A classifier role should properly also show object name (if any) preceding the classifier role name and separated from it by a slash (/). This allows classifier roles in a specification level diagram to be distinguished from instances in an instance level diagram.

This distinction is not implemented in the current release of ArgoUML.

19.2.1. Classifier Role Details Tabs

The details tabs that are active for classifier roles are as follows.

ToDoItem   Standard tab.

Properties   See Section 19.2.2 and Section 19.2.3 below.

Documentation Standard tab.

Style   Standard tab.

Warning

In the current release of ArgoUML changing the values of the `Bounds` field leads to erratic behavior. The `Shadow` field has no effect.

Source   Standard tab, but with no contents.

Caution

A classifier role should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. There are no standard constraints defined for `ClassifierRole` within the UML metamodel.

Note

This is an important distinction from `Instance` used in sequence diagrams, which is able to talk about creation and destruction. Where a collaboration is being used at instance level, it may be appropriate to create these constraints for use on collaboration diagrams.

Tagged Values Standard tab. In the UML metamodel, `Classifier Role` has the following standard tagged values defined.

- `persistence` (from the superclass, `Classifier`. Showing the permanence of the state information associated with the classifier role. Values `transitory` (state is destroyed when the classifier role is destroyed) and `persistent` (state is preserved when the classifier role is destroyed).

- `semantics` (from the superclass, `Classifier`). The value is a specification of the semantics of the classifier role.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the classifier role is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived classifier roles still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

19.2.2. Classifier Role Property Toolbar

`Go up`   Navigate up through the package structure.

`Go back`   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Go forward`   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Delete`   This deletes the classifier role from the model

Warning

This is a deletion from the model *not* just the diagram. To delete an classifier role from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

19.2.3. Property Fields For Classifier Role

`Name`   Text box. The name of the classifier role. By convention classifier role names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

`Classifier` Drop down selector. The name of the classifier of which this is a classifier role.

Note

In the current release of ArgoUML the same graphical presentation is used, even if the classifier role is actually representing an instance of an actor (when a stick-man would be more usual).

`Stereotype` Drop down selector. Classifier Role should be provided by default with the UML standard stereotypes for `Classifier` (`metaclass`, `powerType`, `process`, `thread` and `utility`). ArgoUML also provides the additional stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for classifier roles. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Warning

In the current release of ArgoUML the standard stereotypes for `Classifier` (`metaclass`, `powerType`, `process`, `thread` and `utility`) are not provided.

`Navigate stereotype` If a stereotype has been selected, this will navigate you to the stereotype property panel (see

Section 16.4).

Namespace Text box. Records the namespace for the classifier role. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Association Roles Text area. Lists the association roles that are linked to this classifier role.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- Open. Navigate to the selected association role.

- Add. Add a new association role.

Warning

In the current release of ArgoUML this creates an association, not an association role. In any case there is no mechanism to add a newly created artifact to the diagram.

- Delete. Delete the selected association role.

- Move Up. Only available where there are two or more association roles, and the association role selected is not at the top. It is moved one up the list.

Caution

In the current release of ArgoUML this entry is always grayed out. Association roles can not be moved in the order. To change the order a association role must be deleted and then recreated on the diagram.

- Move Down. Only available where there are two or more association roles listed, and the association role selected is not at the bottom. It is moved one down the list.

Caution

In the current release of ArgoUML this entry is always grayed out. Association roles can not be moved in the order. To change the order a collaboration role must be deleted and then recreated on the diagram.

19.3. Association Role

An association role is a specialization of an association, used to describe an associations behavior in a particular context. In the UML metamodel Association Role is a sub-class of Association.

An association role is represented on a sequence diagram in ArgoUML as a line connecting the instances concerned. However on a sequence diagram the representation is modified to reflect the type of action associated with the stimulus carried on the link (see Section 17.3).

The association role is labeled with the association role name (if any).

Caution

An association role should properly show link name (if any) preceding the association role name and separated from it by a slash (/) and be followed by the association name and separated from it by a colon (:) in the same manner as a classifier role. This allows association roles in a specification level diagram to be distinguished from links in an instance level diagram.

This distinction is not implemented in the current release of ArgoUML.

Caution

Association roles should show the same navigability, aggregation and multiplicity features as their parent association. However these features are no implemented in the current release of ArgoUML.

19.3.1. Association Role Details Tabs

The details tabs that are active for association roles are as follows.

ToDoItem   Standard tab.

Properties   See Section 19.3.2 and Section 19.3.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the association role notionally define the bounding box of the association role. However if you change them it will have no effect.

Source   Standard tab, but with no contents.

Caution

An association role should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. In the UML metamodel, `AssociationEnd` has the following standard constraints defined.

- `xor` (from the superclass `Association`). Applied to a set of association roles, indicating that over that set, exactly one association role is manifest for any particular association instance.

Tagged Values Standard tab. In the UML metamodel, `AssociationRole` has the following standard tagged values defined.

- `persistence` (from the superclass, `Association`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the association is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived association roles still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

19.3.2. Association Role Property Toolbar
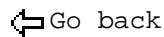
Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Delete   This deletes the association role from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an association role from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

19.3.3. Property Fields For Association Role

Name   Text box. The name of the association role. By convention association role names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Stereotype Drop down selector. Association role should be provided by default with the UML standard stereotype from the superclass, `Association` (`(implicit)`) and the additional Association stereotypes for the Software Engineering Process and Business Modeling in the UML standard (`communicate` and `subscribe`). ArgoUML also provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for association roles. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate Stereotype stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace Text box. Records the namespace for the association role. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Messages Text area. Lists the messages that are associated with this association role.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- `Open`. Navigate to the selected message.

- `Move Up`. Only available where there are two or more messages, and the message selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more messages listed, and the message se-

273

lected is not at the bottom. It is moved one down the list.

`Association End List Roles` are the ends of this association role. An association role can have any number of ends, but two is generally the only useful number (link objects can led to a third end on instance level diagrams, but this is not supported by ArgoUML). For more on association end roles see Section 19.4.

The names are listed, unless the association end role has no name, when it is shown as `anon`.

Warning

Association end roles are not implemented in the current release of ArgoUML. Instead an association end (see Section 15.5) is used. This causes some confusion.

Button 1 click on an association end role will navigate to that end. Button 2 click gives a pop up menu with three options.

- `Open`. Where an association end exists, navigate to it.

- `Move Up`. Only available where there are two or more association end roles, and the association end role selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more association end roles listed, and the association end role selected is not at the bottom. It is moved one down the list.

19.4. Association End Role

Warning

Association end roles are not implemented in the current release of ArgoUML. They are replaced by association ends (see Section 15.5). This does not lead to correct behavior, and the user is advised to avoid them.

19.5. Message

A message is a communication between two instances of an association role on a specification level collaboration diagram. It describes an action which will generate the stimulus associated with the message. On a collaboration diagram a message is associated with an association role. In the UML metamodel `Message` is a subclass of `ModelElement`.

Note

On an instance level collaboration diagram, the stimuli between instances are shown on the links, rather than the messages that describe the communication channel.

The message is represented on a sequence diagram in ArgoUML by its name separated by a colon from the expression defining the associated action, followed by an upward pointing arrow. By convention the name is the message sequence number, either as an integer or as a decimal number to show hierarchy.

Note

ArgoUML will provide a sequence number as the default name for a newly created message.

Caution

The arrow should show the direction of the communication, but in the current release of ArgoUML is always shown pointing upwards. It should also be stereotyped to indicate whether the associated action

is call, synchronous send, asynchronous send or return (usually just implied). The association role may be constrained as `new`, `destroyed` or `transient` if it is created, destroyed or both respectively during the interaction. However these are not shown as messages, since they are actions (create, terminate, destroy) of the associated classifier roles, not communications.

Warning

The current release of ArgoUML has problems with message positioning and will tend to place them on top of each other, refusing to accept moved positions as permanent.

19.5.1. Message Details Tabs

The details tabs that are active for messages are as follows.

ToDoItem   Standard tab.

Properties   See Section 19.5.2 and Section 19.5.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the message define the bounding box of the message. The `Fill` field defines the fill color of the arrow head, and the `Line` field defines the arrow line. Altering the `Shadow` field has no effect.

Caution

In the current release of ArgoUML changing the values of the `Bounds` field is possible, but will make only a temporary change to the position of the message. Selecting any artifact on the screen causes the message to return to its original position and the original values to be restored. This means messages end up on top of each other.

Source   Standard tab, showing the message name and action expression separated by a double colon (`::`).

Caution

A message probably should not generated any code of itself. That should be left to the action and possibly stimulus associated with it. In any case changes to this tab are ignored.

Constraints   Standard tab. There are no standard constraints defined for `Message` within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, `Message` has the following standard tagged values defined.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the message is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived messages still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

275

19.5.2. Message Property Toolbar

Go up   Navigate up through the package structure.
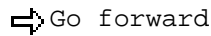
Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

Delete   This deletes the message from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an message from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

19.5.3. Property Fields For Message

Name   Text box. The name of a message is usually its sequence number, either an integer, or a decimal (allowing alternative message hierarchies to be clearly described). ArgoUML will supply an integer sequence number by default.

Stereotype Drop down selector. Message has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for messages. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate Stereotype If one stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace   Text box. Records the namespace for the message. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Sender   Text box. Identifies the classifier role which sent this message.

Button 1 click takes you to the sender classifier role, button 2 gives a pop up menu with one entry.

• `Open`. Navigate to the selected sender classifier role.

Receiver   Text box. Identifies the classifier role which receives this message.

Button 1 click takes you to the receiver classifier role, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected receiver classifier role.

Predecessor. Text area. Identifies the messages, the completion of whose execution enables this message.

Button 1 click takes you to the selected message, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected message.

Warning

This entry is always grayed out in the current release of ArgoUML

Activation. Text area. Identifies the message which invokes the behavior that causes the sending of this message.

Button 1 click takes you to the selected message, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected message.

Warning

This entry is always grayed out in the current release of ArgoUML

Action Text box. Lists the action (see Section 18.3) this message invokes to raise a stimulus.

Button 1 click takes you to the selected action, button 2 gives a pop up menu with three entries.

- Open. Navigate to the selected action.
- Delete. Delete the selected action.
- Add. Add a new action.

Warning

The current release of ArgoUML seems to have some problems in this area. Adding actions seems to replace any existing action, and also causes message graphics to become tangled together. Use with considerable care.

Chapter 20. ArgoUML Activity Diagram Artifact Reference

Table of Contents

20.1. Introduction

This chapter describes each artifact that can be created within an Activity diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

There is a close relationship between this material and the Properties Tab of the Details Pane (see Section 10.4). That section covers Properties in general, in this chapter they are linked to specific artifacts.

Caution

Activity diagrams are not fully developed yet in ArgoUML. Many aspects are not fully implemented, or may not behave as expected. In particular swim lanes, action-object flows, control icons (signals), sub-activities and interactions with other classifiers are not provided. In practice there is little in the current implementation of the activity diagram to distinguish it from the state diagram (see Chapter 18).

Figure 20.1 shows an Activity Diagram with all possible artifacts displayed.

Figure 20.1. Possible artifacts on an activity diagram.
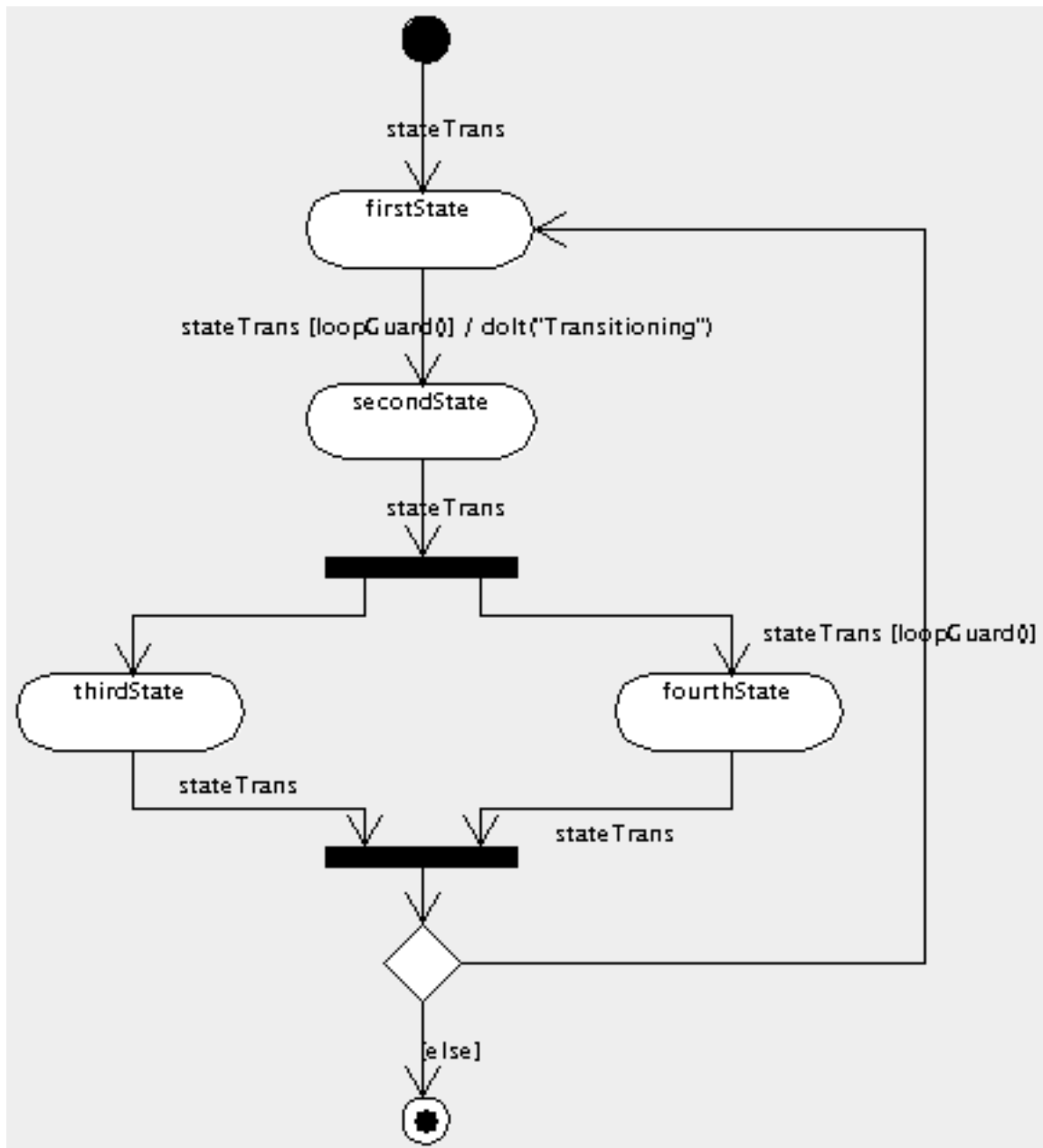
20.2. Action State

An action state represents execution of an atomic action, usually the invocation of an action. Within the UML metamodel, `ActionState` is a sub-class of `SimpleState`. It is a specialized simple state that only has an entry action, and with an implicit trigger as soon as that action is completed.

Caution

As a consequence any outgoing transitions from an action state should not have explicit triggers defined (ArgoUML will not currently check for this). They may have guards to provide a choice where there is more than one transition.

An action state is represented on an activity diagram in ArgoUML as a rectangle with rounded corners containing the name of the action state.

Caution

The UML standard expects that the activity diagram should contain the expression associated with the entry action, rather than the action state name.

20.2.1. Action State Details Tabs

The details tabs that are active for action states are as follows.

ToDoItem   Standard tab.

Properties   See Section 20.2.2 and Section 20.2.3 below.

Documentation Standard tab.

Style   Standard tab. The values for the bounds of the action state define the bounding box of the action state.

Caution

In the current release of ArgoUML there are known problems with typing in the Bounds box, with the cursor jumping about. Take care that you enter the changes you intend when changing values

Caution

The Shadow field has no effect— in the current release of ArgoUML. There are no shadow effects provided for action states on the diagram.

Source   Standard tab that is empty.

Note

In the fullness of time this tab will contain code to implement the action state (code for actions will be associated with the particular actions). This is not implemented in the current release of ArgoUML.

Constraints   Standard tab. There are no standard constraints defined for ActionState within the UML metamodel.

Tagged Values Standard tab. In the UML metamodel, ActionState has the following standard tagged values defined.

- derived (from the superclass, ModelElement). Values true, meaning the action state is redundant—it can be formally derived from other elements, or false meaning it cannot.

Note

Derived action states still have their value in analysis and design to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML Element metaclass from which all other artifacts are derived includes the tagged element documentation which is handled by the *documentation tabdocumentation tab* under ArgoUML

20.2.2. Action State Property ToolBar

Go up   Navigate up through the package structure.

⇦ Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

⇨ Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the action state from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an action state from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

20.2.3. Property fields for action state

Name   Text box. The name of the action state. By convention action state names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Stereotype Drop down selector. Action state has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for action states. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate Stereotype If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Entry-Action Text box. Lists the action (if any) to be invoked on entry to this action state.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- `Open`. Navigate to the selected action.

- `Delete`. Delete the selected action.

- `Add`. Add a new action.

Modifier Check box. Indicates whether the action state's actions may be executed concurrently if there are parallel executions invoked.

Caution

This field is of limited value in the current release of ArgoUML in the absence of fields specifying dynamic arguments and dynamic multiplicity.

Incoming Text area. Lists the transitions that enter this action state.

Note

Unlike an ordinary state, internal transitions are not permitted for action states.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected transition.

Outgoing Text area. Lists the transitions that leave this action state.

Note

Unlike an ordinary state, internal transitions are not permitted for action states.

Button 1 click takes you to the selected action, button 2 gives a pop up menu with one entry.

- Open. Navigate to the selected transition.

20.3. Action

This artifact is described in the context of state diagrams (see Section 18.3).

20.4. Transition

This artifact is described in the context of state diagrams (see Section 18.5).

Caution

Remember that action states do not have explicit triggers. The transition is implicitly triggered as soon as the entry event of the action state is complete. An explicit trigger should not therefore be set.

The current release of ArgoUML will not check that this constraint is met.

20.5. Call Event

This artifact is described in the context of state diagrams (see Section 18.6).

Caution

The only place this is used is for the trigger of a transition. However remember that action states do not have explicit triggers. The transition is implicitly triggered as soon as the entry event of the action state is complete. An explicit trigger should not therefore be set.

The current release of ArgoUML will not check that this constraint is met.

20.6. Guard

This artifact is described in the context of state diagrams (see Section 18.7).

20.7. Initial State

This artifact is described in the context of state diagrams (see Section 18.9).

20.8. Final State

This artifact is described in the context of state diagrams (see Section 18.10).

20.9. Branch

This artifact is described in the context of state diagrams (see Section 18.11).

Caution

As a detailed technicality, a branch on an activity diagram should be static, i.e. implemented as a `junction` pseudostate, rather than a `choice` pseudostate. The current release of ArgoUML uses `choice` which is wrong.

20.10. Fork

This artifact is described in the context of state diagrams (see Section 18.12).

20.11. Join

This artifact is described in the context of state diagrams (see Section 18.13).

Chapter 21. ArgoUML Deployment Diagram Artifact Reference

Table of Contents

21.1. Introduction

This chapter describes each artifact that can be created within a Deployment Diagram. Note that some sub-artifacts of artifacts on the diagram may not actually themselves appear on the diagram.

There is a close relationship between this material and the Properties Tab of the Details Pane (see Section 10.4). That section covers Properties in general, in this chapter they are linked to specific artifacts.

Within ArgoUML, the deployment diagram is also used for both component diagrams (i.e. without instances, showing static dependencies of components) and deployment diagrams (showing how instances of components are handled by instances of nodes at run-time).

Caution

Deployment diagrams are not fully developed yet in ArgoUML. Some aspects are not fully implemented or may not behave as expected. Notable omissions are a correct representation for interfaces and proper stereotyping of the various dependency relationships.

Figure 21.1 shows a component diagram with all possible artifacts displayed.

Figure 21.1. Possible artifacts on a component diagram.
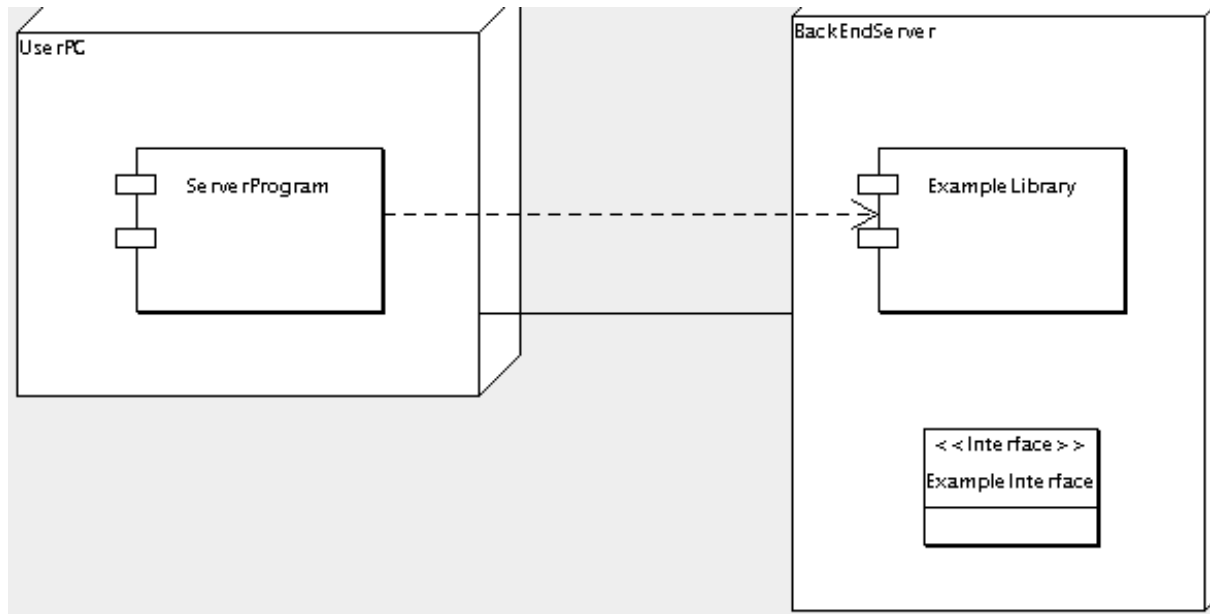
Figure 21.2 shows a deployment diagram with all possible artifacts displayed.

Figure 21.2. Possible artifacts on a deployment diagram.
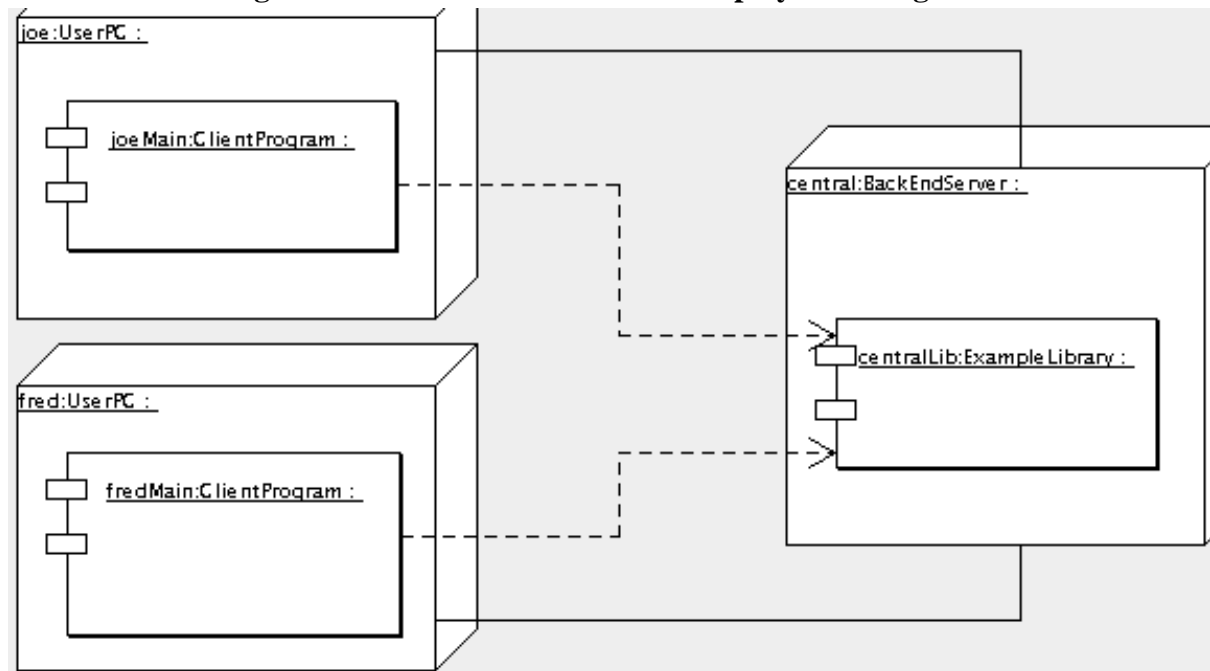
21.2. Node

A node is a run-time physical object on which components may be deployed. In the UML metamodel it is a sub-class of `Classifier`.

A node is represented on a class diagram as a 3-D box, labeled with its name.

21.2.1. Node Details Tabs

The details tabs that are active for nodes are as follows.

ToDoItem   Standard tab.

Properties   See Section 21.2.2 and Section 21.2.3 below.

Documentation Standard tab.

Style   Standard tab. The `Bounds:` field defines the bounding box for the node on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

Beware that in the current release of ArgoUML, the bounding box just refers to the front face of the cube. This means that the 3-D top and side may be ignored, for example when determining the limits of a diagram for saving graphics.

Altering the `Shadow` entry has no effect.

Source   Standard tab, but with no contents.

Caution

A node should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. There are no standard constraints defined for `Node` within the UML meta-model.

Tagged Values Standard tab. In the UML metamodel, `Node` has the following standard tagged values defined.

- `persistence` (from the superclass, `Classifier`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- `semantics` (from the superclass, `Classifier`). The value is a specification of the semantics of the node.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the node is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived nodes still have their value in analysis to introduce useful names or concepts, and in design to avoid re-computation.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

21.2.2. Node Property Toolbar
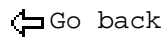
Go up   Navigate up through the package structure.

◁⊐Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.
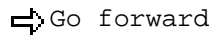
⊏▷Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗Delete   This deletes the node from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a node from the diagram, but keep it within the model, use the main menu Edit/Remove From Diagram (Ctrl-R).

21.2.3. Property Fields For Node

Name   Text box. The name of the node. The name of a node has a leading capital letter, with words separated by "bumpy caps".

Note

ArgoUML does not enforce this naming convention.

Generalizatextareonslists any node that *generalizesgeneralizes* this node.

Button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- Open. Navigate to the selected generalization.

- Add. Add a new generalization. Note that the new generalization is added to the model, but not the diagram. You will need to find it in the navigation pane and add it from there.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- Delete. The link from the generalization to the node (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- Move Up. Only available where there are two or more generalizations listed, and the generalization selected is not at the top. It is moved one up the list.

- Move Down. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

Modified Check box, with entries `abstract`, `final` and `root`.

- `abstract` is used to declare that this node cannot be instantiated, but must always be specialized. The name of an abstract node is displayed in italics on the diagram.

- `final` indicates that this node cannot be further specialized.

- `root` indicates the node can have no generalization.

Namespace Text box. Records the namespace for the node. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Specializations Text box. Lists any specialized node (i.e. for which this node is a generalization.

button 1 click navigates to the generalization and opens its property tab.

Button 2 click yields a drop down menu with five entries

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization. Note that the new generalization is added to the model, but not the diagram.

Tip

It is far easier to add generalizations by drawing them on the relevant diagram in the editing pane.

- `Delete`. The link from the generalization to the node (*but not the generalization itselfbut not the generalization itself*) is removed.

Tip

This *does notdoes not* remove the generalization from the model or the diagram, leading to potentially very confusing diagrams. It is usually far easier to remove generalizations from the editing pane.

- `Move Up`. Only available where there are two or more specializations listed, and the specialization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more specializations listed, and the specialization selected is not at the bottom. It is moved one down the list.

Components Text box. Lists any components (see Section 21.4) designed to be deployed on this type of node.

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with three entries.

- `Open`. Navigate to the selected component.

- `Move Up`. Only available where there are two or more components, and the component selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more components listed, and the component selected is not at the bottom. It is moved one down the list.

21.3. Node Instance

A node instance is an instance of a node where component instances (see Section 21.5) may reside. In the UML metamodel `NodeInstance` is a sub-class of `Instance` and is specifically an instance that is derived from a

node.

A node instance is represented on a sequence diagram in ArgoUML as a 3-D box labeled with the node instance name (if any) and node type, separated by a colon (:).

Tip

It is the presence of the colon (: that distinguishes a node instance from a node.

Caution

The current release of ArgoUML there is no way of specifying the node type, so it must be done in the name. This leads to the representation having two colons, one at the very end.

21.3.1. Node Instance Details Tabs

The details tabs that are active for node instances are as follows.

ToDoItem   Standard tab.

Properties   See Section 21.3.2 and Section 21.3.3 below.

Documentation Standard tab.

Style   Standard tab. The Bounds: field defines the bounding box for the node instance on the diagram.

Warning

There are some bugs remaining in entering data in the Bounds: field. Take care when typing that you do get the values you want in place.

Beware that in the current release of ArgoUML, the bounding box just refers to the front face of the cube. This means that the 3-D top and side may be ignored, for example when determining the limits of a diagram for saving graphics.

Altering the Shadow entry has no effect.

Source   Standard tab, containing just the name of the node instance.

Caution

A node instance should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. In the UML metamodel, NodeInstance has the following standard constraints defined (through its parent, Instance).

- destroyed. Applied to a node instance that is destroyed during the execution.

Caution

It is conceivable this could apply to some robotic systems, or where UML is being used to describe systems with human behavior. However it is unlikely to be of widespread use.

- new. Applied to a node instance that is created during the execution.

Caution

It is conceivable this could apply to some robotic systems, or where UML is being used to describe systems with human behavior. However it is unlikely to be of widespread use.

- `transient`. Applied to an node instance that is both created and destroyed during the execution.

Caution

It is conceivable this could apply to some robotic systems, or where UML is being used to describe systems with human behavior. However it is unlikely to be of widespread use.

`Tagged Values` Standard tab. In the UML metamodel, `NodeInstance` has the following standard tagged values defined.

- `persistence` (from the superclass, `Instance`. Showing the permanence of the state information associated with the node instance. Values `transitory` (state is destroyed when the node instance is destroyed) and `persistent` (state is preserved when the node instance is destroyed).

Caution

It is conceivable this could apply to some robotic systems, or where UML is being used to describe systems with human behavior. However it is unlikely to be of widespread use.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the node instance is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Caution

It is not clear that this tagged value has any great use when applied to a node instance.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

21.3.2. Node Instance Property Toolbar

`Go up`   Navigate up through the package structure.

`Go back`   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Go forward`   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Delete`   This deletes the node instance from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete an node instance from the diagram,

but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

21.3.3. Property Fields For Node Instance

`Name`   Text box. The name of the node instance. By convention node instance names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Caution

The current release of ArgoUML there is no way of specifying the node type, so it must be done in the name. This leads to the representation having two colons, one at the very end.

`Stereotype` Drop down selector. Node instance has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for node instances. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

`Navigate to stereotype` If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

`Namespace` Text box. Records the namespace for the node instance. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

21.4. Component

A component is a physical part of the system (source code, object code, database, library, documentation) that realizes a set of interfaces. In the UML metamodel it is a sub-class of `Classifier`.

A component is represented on a class diagram as a box with two small rectangles protruding from its left side, labeled with its name.

21.4.1. Component Details Tabs

The details tabs that are active for components are as follows.

`ToDoItem`   Standard tab.

`Properties`   See Section 21.4.2 and Section 21.4.3 below.

`Documentation` Standard tab.

`Style`   Standard tab. The `Bounds:` field defines the bounding box for the component on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

In addition when typing or making any other change to a component on the diagram, its left edge will jump to the left.

`Source`   Standard tab, but with no contents.

Caution

A component should not generate any code, so having this tab active is probably a mistake.

`Constraints`   Standard tab. There are no standard constraints defined for `Component` within the UML metamodel.

`Tagged Values` Standard tab. In the UML metamodel, `Component` has the following standard tagged values defined.

- `persistence` (from the superclass, `Classifier`). Values `transitory`, indicating state is destroyed when an instance is destroyed or `persistent`, marking state is preserved when an instance is destroyed.

- `semantics` (from the superclass, `Classifier`). The value is a specification of the semantics of the component.

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the component is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Note

Derived components may have value to indicate that the component represents an optimization.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

21.4.2. Component Property Toolbar

`Go up`   Navigate up through the package structure.

`Go back`   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Go forward`   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

`Delete`   This deletes the component from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a component from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

21.4.3. Property Fields For Component

Name   Text box. The name of the component. The name of a component has a leading capital letter, with words separated by "bumpy caps".

Note

ArgoUML does not enforce this naming convention.

Stereotype   Drop down selector. Package is provided by default with the UML standard stereotypes for component (`document`, `executable`, `file`, `library` and `table`). ArgoUML also provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for components. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate   Icon. If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace   Text box. Records the namespace for the component. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

Extends   Text box. Lists any component that generalizes this component.

Caution

ArgoUML is incorrect in using the designation "Extends" for this field. It should correctly be called "Generalizations".

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with five entries.

- `Open`. Navigate to the selected generalization.

- `Add`. Add a new generalization.

Caution

In the current release of ArgoUML generalizations of components are not properly implemented. This button will add a generalization, named null with no parent, that can not be added to the diagram. Its value is somewhat limited!

- `Delete`. Removes the link to the selected generalization.

- `Move Up`. Only available where there are two or more generalizations, and the generalization selected is not at the top. It is moved one up the list.

- `Move Down`. Only available where there are two or more generalizations listed, and the generalization selected is not at the bottom. It is moved one down the list.

Modified Check box, with entries `abstract`, `final` and `root`.

- `abstract` is used to declare that this component cannot be instantiated, but must always be specialized.

- `final` indicates that this component cannot be further specialized.

- `root` indicates the node can have no generalization.

Derived Text area. Lists any derived components, i.e those for which this component is a generalization.

Caution

ArgoUML is incorrect in using the designation "Derived" for this field. It should correctly be called "Specializations".

Button 1 click takes you to the selected entry, button 2 gives a pop up menu with one entry.

- `Open`. Navigate to the selected specialization.

Caution

In the current release of ArgoUML generalizations of components are not properly implemented, and this button is always grayed out.

21.5. Component Instance

A component instance is an instance of a component (see Section 21.4) which may reside on a node instance (see Section 21.3). In the UML metamodel `ComponentInstance` is a sub-class of `Instance` and is specifically an instance that is derived from a component.

A component is represented on a class diagram as a box with two small rectangles protruding from its left side, labeled with its name.

A component instance is represented on a sequence diagram in ArgoUML as a box with two small rectangles protruding from its left side labeled with the component instance name (if any) and component type, separated by a colon (:).

Tip

It is the presence of the colon (: that distinguishes a component instance from a component.

Caution

The current release of ArgoUML there is no way of specifying the component type, so it must be done in the name. This leads to the representation having two colons, one at the very end.

21.5.1. Component Instance Details Tabs

The details tabs that are active for component instances are as follows.

ToDoItem   Standard tab.

Properties   See Section 21.5.2 and Section 21.5.3 below.

Documentation Standard tab.

Style   Standard tab. The `Bounds:` field defines the bounding box for the component on the diagram.

Warning

There are some bugs remaining in entering data in the `Bounds:` field. Take care when typing that you do get the values you want in place.

In addition when typing or making any other change to a component on the diagram, its left edge will jump to the left.

Source   Standard tab, containing just the name of the component instance.

Caution

A component instance should not generate any code, so having this tab active is probably a mistake.

Constraints   Standard tab. In the UML metamodel, `ComponentInstance` has the following standard constraints defined (through its parent, `Instance`).

- `destroyed`. Applied to a component instance that is destroyed during the execution.

- `new`. Applied to a component instance that is created during the execution.

- `transient`. Applied to an component instance that is both created and destroyed during the execution.

Tagged Values Standard tab. In the UML metamodel, `ComponentInstance` has the following standard tagged values defined.

- `persistence` (from the superclass, `Instance`. Showing the permanence of the state information associated with the component instance. Values `transitory` (state is destroyed when the component instance is destroyed) and `persistent` (state is preserved when the component instance is destroyed).

- `derived` (from the superclass, `ModelElement`). Values `true`, meaning the component instance is redundant—it can be formally derived from other elements, or `false` meaning it cannot.

Caution

It is not clear that this tagged value has any great use when applied to a component instance.

Note

The UML `Element` metaclass from which all other artifacts are derived includes the tagged element `documentation` which is handled by the *documentation tabdocumentation tab* under ArgoUML

21.5.2. Component Instance Property Toolbar

Go up   Navigate up through the package structure.

Go back   Navigate back to the previous artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.
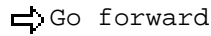
⇨ Go forward   Navigate forward to the next artifact.

Warning

This feature is currently broken within ArgoUML and there is no guarantee where (if anywhere) this button will take you.

✗ Delete   This deletes the component instance from the model

Warning

This is a deletion from the model *notnot* just the diagram. To delete a component instance from the diagram, but keep it within the model, use the main menu `Edit/Remove From Diagram` (Ctrl-R).

21.5.3. Property Fields For Component Instance

Name   Text box. The name of the component instance. By convention component instance names start with a lower case letter and use bumpy caps to divide words within the name.

Note

ArgoUML does not enforce this naming convention.

Caution

The current release of ArgoUML there is no way of specifying the component type, so it must be done in the name. This leads to the representation having two colons, one at the very end.

Stereotype   Drop down selector. Component instance has no stereotypes by default in the UML standard, but ArgoUML provides the stereotypes, `machine`, `organization` and `person`.

Caution

ArgoUML also provides the stereotype `realize` for component instances. This appears to be an error, since this stereotype properly belongs to the `Abstraction` metaclass.

Navigate stereotype   If a stereotype has been selected, this will navigate you to the stereotype property panel (see Section 16.4).

Namespace   Text box. Records the namespace for the component instance. This is the package hierarchy.

Button 1 click on the entry will take you to the package defining this namespace (or the model for the top level namespace).

21.6. Dependency

A key part of any component or deployment diagram is to show dependencies. For details see Section 15.10.

Caution

UML relies on stereotyping of dependencies on component and deployment diagrams to characterize the types of relationship. In the current release of ArgoUML there are limitations in the implementation of dependencies which limit this functionality.

21.7. Class

A component diagram may show the key internal structure of components, including classes within the component. For details see Section 16.5.

21.8. Interface

A component or deployment diagram may show components or component instances which implement interfaces. For details see Section 16.14.

Caution

The current release of ArgoUML uses the same representation of an interface as a class diagram. The UML standard suggests that an interface on a component or deployment diagram should just be shown as a small open circle, connected to the component which realizes that interface.

Warning

There is no way to show the linking of an interface to a component or component instance in the current release of ArgoUML.

21.9. Association

Where classes or interfaces are shown within components on component diagrams, they may be shown linked by associations. For details see Section 15.4.

21.10. Object

Just as components may show the classifiers that make up their internal structure, component instances on deployment diagrams may show the classifier instances that make up their internal structure. In practice the only instance that is of use is an object (an instance of a class). For details see Section 17.2.

21.11. Link

Where objects are shown within component instances on deployment diagrams, their inter-relationships may be shown as links (an instance of an association). See Section 17.9 for details.

Chapter 22. ArgoUML Built In DataTypes, Classes, Intefaces and Stereotypes

Table of Contents

22.1. Introduction

This chapter describes the datatypes, classes, interfaces and stereotypes, which by default, are built in to ArgoUML.

Datatypes, classes and interfaces are generally available for use anywhere a class may be selected in the properties tab. The most common use is for types of methods and parameters.

22.1.1. Package Structure

The datatypes, classes and interfaces are effectively organised as a hierarchy beneath the overall model itself. They are in four packages, `lang`, `math`, `net` and `util`, themselves subpackages of `java`, which is a subpackage of the model itself. Figure 22.1 shows this as a UML package diagram.

Figure 22.1. Hierarchy of datatypes, classes and interfaces within ArgoUML
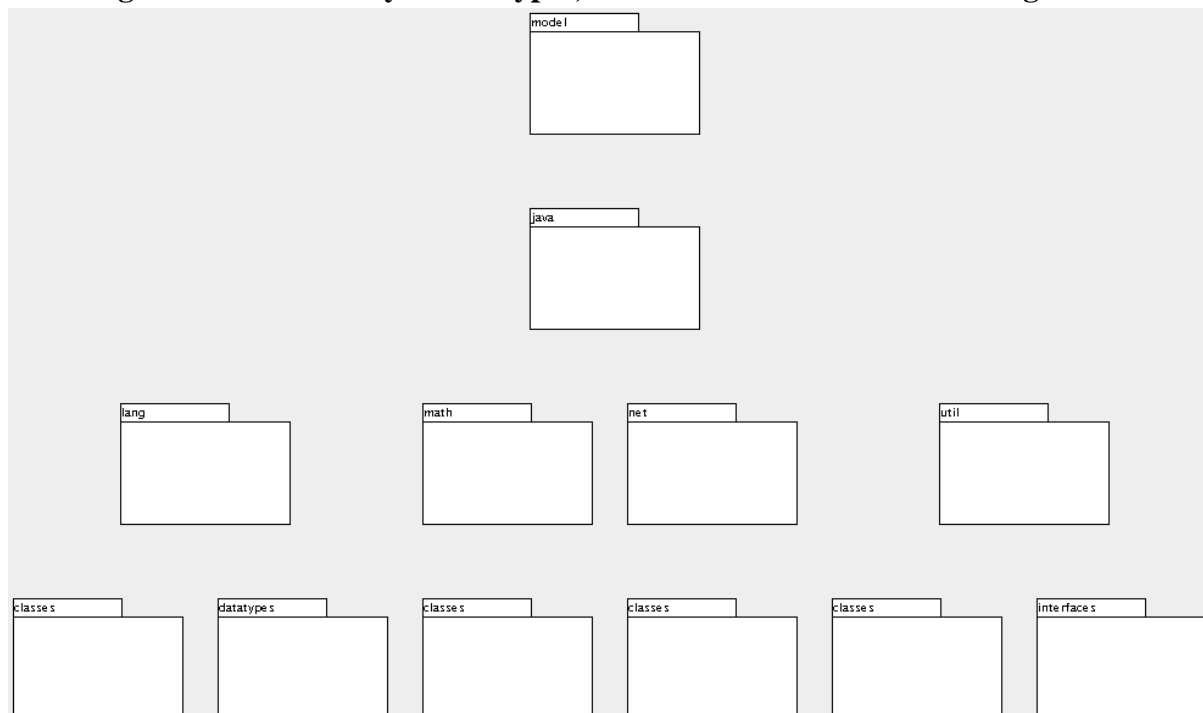
22.2. Built In Datatypes

These are the built in atomic types. You can change them if you wish. However this is not good practice

All these can be found in the `java.lang` subpackage of the main model.

Note

You will not find these exposed within the top level package by default. However if you select a datatype (say from within a class parameter property sheet), and navigate up, you can navigate up to the top package. Thereafter you will find that the package has appeared as a subpackage of the model, and you can navigate back down (but only to the type you came up from).

Caution

You should be aware that these are Java datatypes. They are not mandated by the UML standard.

These are the standard datatypes. For their definition refer to the Java standard.

- `boolean`
- `byte`
- `char`
- `double`
- `float`
- `int`
- `long`
- `short`
- `void`

Note

`void` is not strictly speaking a type, but the absence of type. ArgoUML knows about void and allows it as an option where a datatype may be selected.

22.3. Built In Classes

These are the common classes, corresponding to classes defined within the standard Java environment. It is up to you if you wish to change them.

These are found in all four subpackages of the `java` subpackage.

Note

As with datatypes, you will not find these exposed within the top level package by default. However if you select a datatype (say from within a class parameter property sheet), and navigate up, you can navigate up to the top package. Thereafter you will find that the package has appeared as a subpackage of the model, and you can navigate back down (but only to the type you came up from).

For a definition of these classes see the Java language and library defintions.

22.3.1. Built In Classes Within java.lang

These are the classes within the `java.lang` package.

- `Boolean`

- `Byte`

- `Char`

- `Double`

- `Float`

- `Integer`

- `Long`

- `Object`

- `Short`

- `String`

22.3.2. Built In Classes Within java.math

These are the classes within the `java.math` package.

- `Big Decimal`

- `Big Integer`

22.3.3. Built In Classes Within java.net

These are the classes within the `java.net` package.

- `URL`

22.3.4. Built In Classes Within java.util

These are the classes within the `java.util` package.

- `Vector`

- `Date`

- `Time`

22.4. Built In Interfaces

These are some useful interfaces, corresponding to classes defined within the standard Java environment. Interfaces have many of the properties of classes (like all types) and you can change them if you wish.

All these can be found in the `java.util` subpackage of the main model.

Note

You will not find these exposed within the top level package by default. However if you select a datatype (say from within a class parameter property sheet), and navigate up, you can navigate up to the top package. Thereafter you will find that the package has appeared as a subpackage of the model, and you can navigate back down (but only to the type you came up from).

These are the interfaces defined within the `java.util` package. For their definition consult the Java language and library references.

- `Collection`

- `Iterator`

- `List`

- `Set`

- `Sorted Set`

22.5. Built In Stereotypes

UML defines a large number of stereotypes which are (mostly) supported by ArgoUML. In addition ArgoUML defines three additional stereotypes that may be of use.

- `machine`. Available to artifacts derived from the top level UML metaclass, `Element`.

- `person`. Available to artifacts derived from the top level UML metaclass, `Element`.

- `organization`. Available to artifacts derived from the UML metaclass `ModelElement`.

Caution

Not all stereotypes defined by UML 1.3 appear in ArgoUML, due to limitations in the current implementation of some artifacts.

Glossary

A

Activity Diagram	A UML diagram capturing the dynamic behavior of a system or sub-system. See for more information.
Action	Behavior associated with *StatesStates* or *TransitionsTransitions* in *State DiagramState Diagram*. These actions are invocations of *MethodsMethods* and appear on *SequenceSequence* and *Collaboration DiagramsCollaboration Diagrams*. See Also Action, Collaboration Diagram, Sequence Diagram, State, State Diagram, Transition.
Actor	A representation of an agent (animate or inanimate) on a *Use Case DiagramUse Case Diagram* external to the system being designed. See Also Use Case, Use Case Diagram.
Analysis	Analysis is the process of taking the "customer" requirements and re-casting them in the language of, and from the perspective of, a putative solution.
Association Class	A class that characterises the association between two other clases. See Also Class.
Association	A relationship between two classes in a Class Diagram or between Use Cases or Use Cases and Actors in a Use Case Diagram. See Also Actor, Class Diagram, Use Case, Use Case Diagram.
Attribute (of a Class or Object)	An attribute of a class or object is a specification of a data element encapsulated by that object. See Also Class, Method (of a Class or Object), Object.

C

CASE	Computer Aided Sofware Engineering.
Class	The encapsulation of the data associated with an artifact (its *at-attributestributes*) and the actions associated with the artifact (its *methodsmethods*). A class specifies the characteristics of an artifact. An *objectobject* represents an instance of the artifact. Classes and objects in UML are represented on *Activity DiagramsActivity Diagrams*, *Class DiagramsClass Diagrams*, *Collaboration Col-Diagramslaboration Diagrams* and *Sequence DiagramsSequence Diagrams*. See Also Activity Diagram, Attribute (of a Class or Object), Class Diagram, Collaboration Diagram, Method (of a Class or Object), Object, Sequence Diagram.
Class Diagram	A UML Diagram showing the structural relationship between classes. See Section 4.3 for more information. See Also Class, Class Diagram.
Collaboration Diagram	A UML Diagram showing the dyanamic behavior as messages are passed between objects. Equivalent to a *Sequence DiagramSequence Diagram*. Which representation is appropriate depends on the problem under consideration. See for more information. See Also Sequence Diagram.
Comprehension and Problem Solving	A design visualization theory within cognitive psychology. The theory notes that designers must bridge a gap between their mental model of the problem

or situation and the formal model of a solution or system.

This theory suggests that programmers will benefit from:

1. Multiple representations such as program syntactic decomposition, state transitions, control flow, and data flow. These allow the programmer to better identify elements and relationships in the problem and solution and thus more readily create a mapping between their situation models and working system models.

2. Familiar aspects of a situation model, which improve designers' abilities to formulate solutions.
See Also Critic.

Concept Class Diagram

A Class Diagram constructed during the Analysis Phase to show the main structural components of the problem identified in the Requirements Phase. See Chapter 4 for more information.
See Also Class Diagram.

Critic

A process within ArgoUML that provides suggestions as to how the design might be improved. Suggestions are based on principles within three theories of cognitive psychology, *reflection-in actionreflection-in action*, *opportunistic designopportunistic design* and *comprehension and problem solvingcomprehension and problem solving*.
See Also Reflection-in-Action, Opportunistic Design, Comprehension and Problem Solving.

E

Extends Relationship

A relationship between two Use Cases, where the *extendedextended* Use Case describes a special variant of the *extendingextending* Use Case.
See Also Generalization Relationship, Includes Relationship, Use Case.

G

Generalization Relationship

A relationship between one *generalisinggeneralising* Use Cases and one or more *generalisedgeneralised* Use Cases, where the *generalisedgeneralised* Use Cases are particular examples of the *generalisinggeneralising* Use Case.
See Also Extends Relationship, Includes Relationship, Use Case.

GNU

"GNU's Not Unix". One of the first open/free source projects. Originally best known for its GNU version of the emacs editor, it now has many other projects, usually closely allied to Linux.
See Also Linux.

GUI

Graphical User Interface.

H

Hierarchical State Diagram

A *State DiagramState Diagram* that contains subsidiary state diagrams within individual *StatesStates*.
See Also State, State Diagram.

I

Includes Relationship

A relationship between two Use Cases, where the *includedincluded* Use Case describes part of the functionality of the *includingincluding* Use Case.
See Also Extends Relationship, Generalization Relationship, Use Case.

Iterative Design Process	A design process where each all phases (requirements, analysis, design, build, test) are tackled partially in a series of iterations. See Section 2.3.1 for more information. See Also Waterfall Design Process.

J

Java	A fully object oriented programming language introduced by Sun Microsystems. More strongly typed than C++, it compiles to an interpreted code, the Java Virtual Machine (JVM). The JVM means that Java code should run on any machine that has implemented the JVM. The most significant component of Java was integration of the JVM into web browsers, allowing code (Applets) to be download and run over the web. ArgoUML is written in Java.

L

Linux	An open source version of the Unix operating system. See Also GNU.

M

Mealy Machine	A *State DiagramState Diagram* where actions are associated with *StatesStates*. See Also Action, Moore Machine, State, State Diagram, Transition.
Method (of a Class or Object)	A method of a class or object is a specification of behavior encapsulated by that object. See Also Attribute (of a Class or Object), Class, Object.
Moore Machine	A *State DiagramState Diagram* where actions are associated with *TransitionsTransitions*. See Also Action, Mealy Machine, State, State Diagram, Transition.

O

Object	An instance of a *ClassClass*. Classes and objects in UML are represented on *Activity DiagramsActivity Diagrams*, *Class DiagramsClass Diagrams*, *Collaboration Col-Diagramslaboration Diagrams* and *Sequence DiagramsSequence Diagrams*. See Also Activity Diagram, Attribute (of a Class or Object), Class Diagram, Collaboration Diagram, Method (of a Class or Object), Object, Sequence Diagram.
OCL	Object Constraint Language. A language for describing constraints within UML. See Also UML.
OMG	The Object Management Group. An international industry standardisation body. Best known for CORBA and UML. See Also UML.
OOA&D	Object Oriented Analysis and Design. An approach to software problem analysis and design based on objects, which encapsulate both data and code. See See Section 1.1.1 or any standard textbook on Software Engineering. UML is a notation to support OOA&D.

See Also UML.

Opportunistic Design | A theory within cognitive psychology suggesting that although designers plan and describe their work in an ordered, hierarchical fashion, in actuality, they choose successive tasks based on the criteria of cognitive cost. Simply stated, designers do not follow even their own plans in order, but choose steps that are mentally least expensive among alternatives.
See Also Critic.

P

Pane | A subwindow within the main window of the ArgoUML user interface.

R

Realisation Use Case | A *Use CaseUse Case* where the *Use Case DiagramUse Case Diagram* and *Use Case SpecificationUse Case Specification* are in the language of the solution domain, rather than the problem domain.
See Also Use Case.

Reflection-in-Action | A theory within cognitive psychology which observes that designers of complex systems do not conceive a design fully-formed. Instead, they must construct a partial design, evaluate, reflect on, and revise it, until they are ready to extend it further. As developers work hands-on with the design, their mental model of the problem situation improves, hence improving their design.
See Also Critic.

Requirements Capture | Requirements capture is the process of identifying what the "customer" wants from the proposed system. See Chapter 3 for a fuller description.
See Also Use Case.

S

Sequence Diagram | A UML Diagram showing the dyanamic behavior as messages are passed between objects. Equivalent to a *Collaboration DiagramCollaboration Diagram*. Which representation is appropriate depends on the problem under consideration. See Section 4.5 for more information.
See Also Collaboration Diagram.

SGML | Standard Graphical Markup Language. Defined by ISO 8879:1986.
See Also XML.

Simula 67 | A procedural programming language intended for simulation. Noted for its introduction of *objectsobjects* and *coroutinescoroutines*.

State | Within a *State DiagramState Diagram* a one of the possible configurations of the machine.
See Also State Diagram.

State Diagram | A UML Diagram showing the dyanamic behavior of an active *ObjectObject*.
See Section 4.7 for more information.
See Also Object.

Stereotypes and Stereotyping | Any artifact within UML can be given a *stereotypestereotype* to indicate its association with a particular role in the design. A stereotype spqr is generally indicated with the notation <<spqr>>.

A stereotype defines a namespace within the design. Examples of stereotypes are <<business>> and <<realisation>> for Use Cases, used to distinguish between Use Cases at the requirements phase defined in terms of the problem domain, and Use Cases at the analysis phase defined in terms

of the solution domain.
See Also UML.

Supplementary Specification	The document capturing non-functional requirements that cannot be associated with Use Cases. See Also Use Case.
SVG	Standard Vector Graphics format. A standard representation of graphics diagrams that use vectors. ArgoUML uses SVG to store its diagrams.
System Sequence Diagram	A *Sequence DiagramSequence Diagram* used in the *AnalysisAnalysis* Phase showing the dyanamic behavior of the overall system. See Chapter 4 for more information. See Also Sequence Diagram.
System State Diagram	A *State DiagramState Diagram* used in the *AnalysisAnalysis* Phase showing the dyanamic behavior of an active top level system objects. See Chapter 4 for more information. See Also State Diagram.

T

| To-Do List | A feature of ArgoUML allowing the user to record activities that are yet to be completed. |
| Transition | The change between *StatesStates* in a *State DiagramState Diagram.*.
See Also State, State Diagram. |

U

UML	Universal Modelling Language. A graphical notation for OOA&D processes, standardised by the OMG. ArgoUML follows UML v1.3. Version 1.4 was released during 2001, with v2.0 planned for 2002. See Also OMG.
Use Case	A UML notation for capturing requirements of a system or sub-system. See Section 3.3 for more information. See Also UML, Use Case Specification.
Use Case Diagram	A UML diagram showing the relationships between Actors and Use Cases. See Section 3.3 for more information. See Also Actor, Use Case.
Use Case Specification	The document capturing the detailed requirements behind a Use Case. See Also Use Case.

V

| Vision Document | The top level document describing what the system being developed is to achieve. |

W

| W3C | The World Wide Web Consortium, www.w3c.org [http:/-/-www.w3c.org]. An international standardisation body for all things to do with the World Wide Web. |
| Waterfall Design Process | A design process where each phase (requirements, analysis, design, build, test) is completed before the next starts. See Section 2.3.1 for more information.
See Also Iterative Design Process. |

X

XMI   XML Model Interchange format. A format for file storage of UML models. Currently incomplete, since it does not carry all graphical layout information, so must be supplemented by files carrying that information.
See Also UML, XML.

XML   eXtensible Markup Language. A simplified derivative of SGML defined by W3C
See Also SGML, W3C.

Appendix A. Supplementary Material for the Case Study

Table of Contents

A.1. Introduction

The case study requires various material (mostly documents) that live alongside the design diagram

A.2. Requirements Documents

To be written...

A.2.1. Vision Document

To be written...

A.2.2. Use Case Specifications

To be written...

A.2.2.1. UC Specication 1

To be written...

A.2.3. Supplementary Requirements Specification

To be written...

Appendix B. UML resources

Table of Contents

B.1. The UML specs

To be written…

B.2. UML related papers

To be written…

B.2.1. UML action specifications

To be written…

B.3. UML related websites

To be written…

Appendix C. UML Conforming CASE Tools

Table of Contents

C.1. Other Open Source Projects

To be written…

C.2. Commercial Tools

To be written…

Appendix D. User Manual Plan

Table of Contents

D.1. Target Audiences

Target audiences are the following.

- Experienced users of UML in OOA&D (perhaps with other tools) who wish to transfer to ArgoUML.

- Designers who know OOA&D, and wish to adopt a UML based process.

In the longer term it would be desirable to also target the following.

- Those who are learning design and wish to start with a UML based OOA&D process.

- People interested in modularised code design with a GUI.

D.2. Goals

We want to keep the boundaries clearly defined, to avoid confusing duplication with the CookBook, FAQ, Quick Guide, online help etc. There are two goals (in priority order).

1. A comprehensive tutorial style explanation of ArgoUML in the context of an OOA&D process.

2. *DescriptiveDescriptive* reference material on all components of ArgoUML

I think the existing User Manual is a good start particularly towards the second of these goals.

D.2.1. Non-Goals

To keep the effort feasible the user manual should avoid the following (at least initially).

- Providing a quick overview—the Quick Guide already does this.

- Listing all the errors and what they mean. The help system does this—one day the manual will link to that.

- Explaining the internal workings of ArgoUML. The cookbook, combined with Jason Robbins dissertation is already a good start for this.

D.3. Suggested Manual Structure

Here are my thoughts. I think the user manual is really a set of two books, the tutorial manual (corresponding to Chapters 1–3 of the current manual), and the reference manual (Chapter 4 of the current manual)

I suggest that the tutorial book be based around an OOA&D process (any preferences), and that each UML concept is introduced with each step of the process, followed by an explanation of how to do it under ArgoUML. A *simplesimple*case study will be needed throughout. A lot of this material can come from existing Chapters 1–3, with some additional material from chapter 4 of the Cookbook, answers in the FAQ and Jason Robbins dissertation.

D.3.1. Tutorial Manual Structure

1. Introduction

    a. Origins and overview of ArgoUML

    b. Scope of the User Manual. Include cross-reference to other documentation (Cookbook, FAQ, Quick Guide, online help, ArgoUML website etc).

    c. Overview of the User Manual. Explains that ArgoUML will be explained in the context of an OOA&D process, and with an example running through.

    d. Assumptions. At this stage assume the user knows OOA&D, but not UML.

2. UML Based OOA&D

    a. Background to UML—what it is, history etc.

    b. UML based processes for OOA&D

    c. ArgoUML Basics—projects, drawing, navigation, details

    d. What ArgoUML has that other tools are missing (critics, to-do list, based in cognitive psychology theory).

    e. The Case Study

3. Requirements Capture

    a. Use Case Diagrams (this section will be relatively large, because its the first time we use ArgoUML to create something).

4. Analysis

    a. Concept Class Diagrams

    b. System Sequence Charts and Collaboration Diagrams

    c. System State Diagrams

5. Design

    a. Class Diagrams for Realisation

    b. Sequence Charts and Collaboration Diagrams for Realisation

    c. State Diagrams for realisation

  d. Package Diagrams

6. Build

  a. Deployment Diagrams

  b. Code Generation in ArgoUML

D.3.2. Reference Manual Structure

This section largely follows Chapter 4 of the existing User Manual, including the Gloassary. However it needs a few extra things.

1. material on each of the diagram types, each of the artifacts that can appear on the diagrams and details of the features of each artifact type.

2. An Index

D.4. Actions, Priorities and Questions

D.4.1. Actions and priorities

Here's my first call for what needs to be done in priority order. From the comments made over the last few days I think the first 5 items won't take very long, meaning effort can concentrate on the main stuff.

1. Get buy-in for the approach

2. Agree document structure (broadly)

3. Choose a suitable example to run throughout.

4. Break into several files (XML entities) to make the manual more manageable.

5. Identify all existing sources of material to be reused

6. Get writing! I suggest the priorities here are:

  a. User Manual sections relating to ArgoUML diagrams and artifacts (assume the reader knows UML already, and allows a quick advance by pulling together a lot of existing material).

  b. User Manual examples

  c. User Manual sections relating to additional ArgoUML cognitive design features.

  d. User Manual sections relating to UML (for readers who don't know UML).

  e. Completion of Reference Manual material.

7. Create an index.

D.4.2. Remaining Questions

1. The current manual shows copyright held by Phillipe, and no legal notice. What is the position of this material?

Appendix E. Documentation Standards for the User Manual

Table of Contents

E.1. Introduction

The entire manual is written using DocBook XML V4.1.2 [http:/-/-www.oasis-open.org/-docbook]. This section covers some conventions for use of DocBook and for the documentation in general. It also includes some information for emacs users and the psgml package.

E.2. Document Conventions

- All titles of chapters, sections etc are capitalised throughout.

- All titles of figures, tables etc have the first word only capitalised.

- Spelling is US English. One day we might abuse XML entities to the extent of making this parameterisable!

E.3. DocBook Conventions

- The top level document is argouml.xml. Each chapter (or preface, glossary, appendix etc) is a separate file, defined as a system entity and included from this top level file.

- There are some useful entities defined for common terms at the top of argouml.xml. Of these, use of `ar-gouml` will ensure consistent naming of the project (Argo/UML) and allow us to change it later (to ArgoUML, Argouml or whatever).

- XML comments are used throughout to explain what various sections are trying to achieve.

- Cross-referencing requires use of `id.` attributes. To avoid confusion, use a prefix of `ch.` for `chapter`, `app.` for `appendix`, `s1.` through `s5.` for `sect1` through `sect5`, `fig.` for `figure`, `tab.` for `table` and `gl` for `glossentry`.

  A second prefix of `tut.` or `ref.` is used to distinguish tutorial and reference material. The remainder of the tag should be descriptive, but concise with words separate by underscore. Where a graphic is involved this remainder should correspond to the file name. For example `fig.ref.navigation_pane` for a figure showing the navigation pane, with the diagram in `navigation_pane.gif`

- Only use `glossterm` (for the term *oror* its abbreviation/acronym), `glossdef` and `glossseealso` within `glossentry`. Other entries are not implemented in the style sheets and so do not appear in the glossary!

- Use spaces (2 per level) rather than tabs. Tabs are generally set so large the text moves over to the right of the page, and are not set the same everywhere (emacs uses 8 spaces, some MS editors use 6 spaces), making documents unreadable between users.

- All block graphics should be encapsulated within `figure`, allowing reference from around the text. Set attribute `float` to 1 to allow the figure to float (makes life easier for printed version).

315

- All block graphics should be provided through `mediaobject` and provided with both an `imageobject` and comprehensive description in a `textobject`. This gives the potential of meaningful content where a diagram cannot be displayed for any reason. Where appropriate the `mediaobject` should be wrapped by `screenshot`.

- Inline graphics can be done through `inlinegraphic`, rather `inlinemediaobject`. A textual alternative is of little value in these circumstances. Where appropriate the `mediaobject` should be wrapped by `guiicon`

- In general use `literal` for computer related text (input and output). There are a huge number of possible specialist tags, but I suggest restriction to the following list: `filename` and `application`.

E.4. For emacs Users

If you use the psgml library within emacs, then editing and verifying XML gets easier. Information on using this facility is inluded with psgml.

- emacs local variables appear in a few lines of comment at the bottom of each XML file. Please don't delete these if you don't use emacs!

- Adding `(setq sgml-set-face t)` to your `.emacs` file will cause all tags and entities to appear in boldface.

- Adding `(setq sgml-auto-activate-dtd t)` to your `.emacs` file will ensure the DocBook DTD is parsed as soon as the file is loaded.

Appendix F. Open Publication Licence

Table of Contents

F.I. REQUIREMENTS ON BOTH UNMODIFIED AND MODIFIED VERSIONS

The Open Publication works may be reproduced and distributed in whole or in part, in any medium physical or electronic, provided that the terms of this license are adhered to, and that this license or an incorporation of it by reference (with any options elected by the author(s) and/or publisher) is displayed in the reproduction.

Proper form for an incorporation by reference is as follows:

> Copyright (c) <year> by <author's name or designee>. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, vX.Y or later (the latest version is presently available at http://www.opencontent.org/openpub/ [http:/-/-www.opencontent.org/-openpub/-]).

The reference must be immediately followed with any options elected by the author(s) and/or publisher of the document (see section VI).

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the title of the work and cited as possessive with respect to the title.

F.II. COPYRIGHT

The copyright to each Open Publication is owned by its author(s) or designee.

F.III. SCOPE OF LICENSE

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided "as is" without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

F.IV. REQUIREMENTS ON MODIFIED WORKS

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements:

1. The modified version must be labeled as such.

2. The person making the modifications must be identified and the modifications dated.

3. Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices.

4. The location of the original unmodified document must be identified.

5. The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

F.V. GOOD-PRACTICE RECOMMENDATIONS

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

1. If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.

2. All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.

3. Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

F.VI. LICENSE OPTIONS

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

A. To prohibit distribution of substantively modified versions without the explicit permission of the author(s). "Substantive modification" is defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections.

To accomplish this, add the phrase "Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder." to the license reference or copy.

B. To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

To accomplish this, add the phrase "Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder." to the license reference or copy.

F.. OPEN PUBLICATION POLICY APPENDIX:

(This is not considered part of the license.)

Open Publication works are available in source format via the Open Publication home page at http://works.opencontent.org/ [http:/-/-works.opencontent.org/-].

Open Publication authors who want to include their own license on Open Publication works may do so, as long as their terms are not more restrictive than the Open Publication license.

If you have questions about the Open Publication License, please contact David Wiley [mailto:dw2@opencontent.org], and/or the Open Publication Authors' List at opal@opencontent.org [mailto:opal@opencontent.org], via email.

To *subscribesubscribe* to the Open Publication Authors' List: Send E-mail to opal-request@opencontent.org with the word "subscribe" in the body.

To *postpost* to the Open Publication Authors' List: Send E-mail to opal@opencontent.org or simply reply to a previous post.

To *unsubscribeunsubscribe* from the Open Publication Authors' List: Send E-mail to opal-request@opencontent.org with the word "unsubscribe" in the body.