ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/4034715

A flyweight UML modelling tool for software development in heterogeneous environments

CONFERENCE PAPER *in* CONFERENCE PROCEEDINGS OF THE EUROMICRO · OCTOBER 2003

DOI: 10.1109/EURMIC.2003.1231600 · Source: IEEE Xplore

CITATIONS
17

READS
43

3 AUTHORS, INCLUDING:

Martin Auer
TU Wien
20 PUBLICATIONS **158** CITATIONS
SEE PROFILE

Stefan Biffl
TU Wien
232 PUBLICATIONS **1,940** CITATIONS
SEE PROFILE

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.

Available from: Martin Auer
Retrieved on: 08 January 2016

A Flyweight UML Modelling Tool for Software Development in Heterogeneous Environments

M. Auer, T. Tschurtschenthaler, S. Biffl
Institute of Software Technology
Vienna University of Technology
martin.auer@tuwien.ac.at

Abstract

A large and growing variety of tools can support all kinds of UML modeling aspects: from model creation to advanced round-trip engineering of UML models and code.

However, such tools aim at supporting specific life-cycle phases, but they often do not meet basic requirements arising in heterogeneous environments, UML education, early life-cycle phases, or agile processes: hassle-free tool deployment, support for fast model sketching, and flexible graphic export features.

This paper presents the freely available modeling tool UMLet we designed to specifically address these basic issues. It is a flyweight Java application that can easily be deployed in various development environments; it features an intuitive and pop-up-free user interface, while still providing output to common high-quality publishing formats.

Thus, the tool UMLet provides an effective way to teach UML and to create and share UML sketches, especially in agile environments and during early life-cycle phases. Its user interface supports intuitive and exploratory modeling, its architecture makes distribution and deployment cost-efficient in heterogeneous environments.

1 Introduction

UML has evolved into a de-facto standard for modeling software systems [11]. This led to a great variety of tools for all kinds of modeling aspects, from the mere creation of UML diagrams or the reverse-engineering of source code for documentation purposes, to round-trip engineering between source code and UML models [10].

While UML tools are loaded with features, the tools often fail to support the most common use cases arising, for example, in early life-cycle phases, in creative modeling processes, or when teaching UML to students:

- When teaching UML, you have to provide or buy a modeling tool, deploy it on the class room computers and provide it to the students so they can work at home. This raises questions about the cost of the tool, its footprint, its platform compatibility etc.

- If you want to quickly sketch a small UML model, you need an intuitive user interface and you don't want to be distracted by seldom-used features. As UML design tools are often perceived as highly complex, many developers still prefer paper notes for this process, which makes sharing the diagrams difficult.

- When sharing models with others, you have to export diagrams in order to use them with Word or Latex, or on a Website. As UML is all about communication, the diagrams are to be exported to various file formats, which can be a tedious task if this is not supported by the UML tool.

The free UML drawing tool UMLet specifically addresses these issues. It is currently being developed by the authors of this paper at the Vienna University of Technology and can be downloaded from `www.umlet.com`; the current version is 1.4.

UMLet is a flyweight and platform-independent application featuring an innovative user interface and flexible output to different file formats. The user interface should allow to quickly and intuitively draw UML sketches; it is based on a simple markup description of UML elements and avoids tedious pop-up dialogs commonly found in other UML tools.

Several issues UMLet adresses arose during the organization of undergraduate software engineering courses at the Vienna University of Technology. The authors are teaching UML-supported software engineering in a one-year obligatory course to several hundreds of students. Other considerations originated in industrial environments, where the authors are responsible for quality assurance.

The following sections present related work, outline important limitations of current tools with regard to tool deployment, user interface simplicity and diagram sharing, and describe how UMLet addresses these issues.

2 Design Considerations for UML Tool Support

For an overview on the Unified Modeling Language UML, please refer to [11]. For a comprehensive and up-to-date overview on currently available UML tools, please refer to M. Jeckle's Web site[1]. Juric and Kuljis [6] present evaluation criteria to assess the extent to which CASE tools support the range of UML language features. A general introduction to software development tools is given in [4]—it puts UML tools in the wider context of the software development environment.

Kelter et al. [7] examine properties of development tools used in heavyweight and agile processes, respectively. The authors mention traditional tools' problems like high complexity and lacking flexibility. They stress seamless access to common data sources, the ability to process sketches as well as full-fledged models, and the principle of keeping tool designs simple as key factors to lightweight tools.

Damm et al. [2] describe gesture-based UML modeling on an electronic whiteboard. The authors point out several CASE tool limitations, especially when it comes to providing support during initial life-cycle phases, when the focus is mainly on understanding, intuition, and flexibility. The authors present a tool to support UML modeling with informal and incomplete UML elements.

Iivari [5] states, that CASE tools in general are perceived by many developers as having a high degree of complexity, which presents a substantial barrier to their usage. IIvari concludes that reducing the user interface's perceived complexity would be profitable.

Lending and Chervany [9] indicate, that usually only a small portion of a CASE tool's functionality is used by the developers, mainly diagram editing and documentation support; a less frequently used functionality was to detect diagram inconsistencies. One conclusion is that UML tools should put the focus on easy and hassle-free diagram creation, and less stress on other, seldom-used features.

On a more general level, several articles in the October 2002 issue of the Communications of the ACM try to explore tool properties to support creative processes. Shneiderman [12] presents a framework of four main activities (collect, relate, create, donate) occurring during a creative process, and of eight tasks that software tools can support in order to accomplish the activities (the tasks include searching, visualizing, and disseminating). Greene [3] describes tool characteristics to support creativity that were implemented in two large applications, the EXPO'92 guest service system and the Museum of Modern Art's "Explore Modern Art" system. Among the mentioned characteristics are (a) supporting pain-free exploration and experimentation and (b) supporting collaboration and idea exchange.

To sum up, current UML tools do support a wide range of features and language elements, yet they fail to address many basic issues, which arise in heterogeneous environments, in UML education, in early life cycle phases, and in agile processes. Those circumstances require (i) fast and easy tool distribution and deployment, an (ii) intuitive user interface for sketching ideas and supporting creative processes, and (iii) easy ways of sharing models and diagrams.

To achieve pain-free deployment in heterogeneous environments and easy distribution of the tool in educational environments, the goal was to design a lightweight, platform-independent tool. To support intuitive model sketching and creative and exploratory processes, the goal was to provide a fast, intuitive and pop-up free user interface. To ease the sharing and publication of models and diagrams, the goal was to support various output formats natively and to design an extensible architecture to add further data formats.

There are of course many other requirements, like collaboration support, version management, access rights managament, however, these are outside the scope of the UMLet project, which aims at addressing the most basic issues.

The next three sections describe problem areas, as well as UMLet's solution approach.

3 Heavyweight vs. Flyweight Approach on Tool Support

When teaching tool-supported software development concepts, both concepts and tools have to be taught. Using heavyweight tools can both distract students from the main educational goals [1, 8] and cause substantial costs—even if UML tool providers often do not charge license fees to universities. Heavyweight tools cause concrete problems:

- Deploying and installing the (often 200+ MB sized) tools in an environment like a computer classroom can be difficult to automate because of interactive installation procedures. Even with replication techniques, one has to deal with license key files, a license server, expiring evaluation periods etc.
- The tool and license keys usually have to be distributed to the students, either by setting up an FTP server—if students have high-speed Internet access—or by copying CDs. This can be a substantial effort, for example, in undergraduate software engineering courses with several hundreds of students.

[1] M. Jeckle describes both CASE and drawing tools at `www.jeckle.de/umltools.html`

- Many UML tools are available on just one platform—the Microsoft Windows family of operating systems, which severely limits the use of alternative operating systems in computer classrooms.
- Many heavyweight tools require high-end computer hardware, which would not otherwise be required, for example, for Linux or introductory Java courses. In addition, students do not necessarily have access to such advanced hardware systems outside the classroom or at home.

UMLet, on the other hand, uses a flyweight appoach, and

- is free and its source code will be released as open source in 2003,
- is a small application (50kb without, 5mb with PDF support) and can be deployed and distributed easily both in binary form or by simply referencing the tool's Web site without bothering about license issues,
- runs on all Java platforms and does not need configuration or special installation procedures, and it
- runs on low-profile hardware—students can therefore work both in the classroom and at home.

This eliminates many troubles, especially in teaching environments which usually have to be set up every term or year. But even in industrial environments these properties make the tool's test and deployment easier, cheaper, and less platform dependent. For example, the effort to install an instance of UMLet on a computer system requires only a few minutes for downloading the small application—there is no separate installation procedure. On the other hand, downloading the large packages necessary for heavyweight tools, performing the installation procedure, entering license keys (or even setting up a license key server) requires substantial effort. Similar considerations are valid for memory usage and processor speed requirements—UMLet is far more cost-efficient in that respect.

4 Compliant vs. Swift Tools

Currently available tools strive to comply to the official UML language specification and support most UML language features. This leads to complex and often overloaded user interfaces with extensive property panels and modal pop-up dialogs. Yet this user interface approach substantially complicates even the most basic operations like adding attributes or operations to a class.

Take, for example, the steps necessary to enter relations' multiplicities in Rational Rose[2]: the designer has to open

[2]An evaluation version of Rational Rose is available at `www.rational.com`.

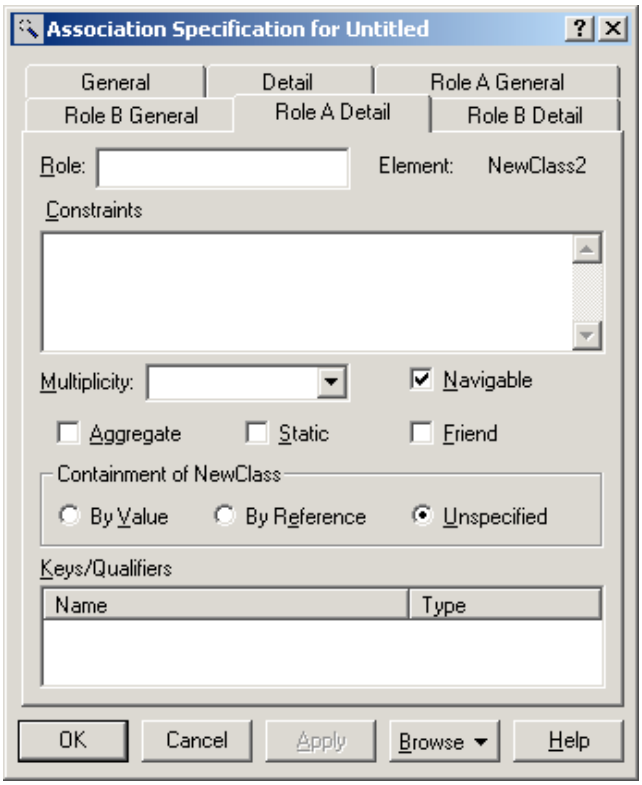

Figure 1. Rational Rose's pop-up dialog to edit relation properties.

a pop-up dialog, switch to one of the tabbed sub windows (either "Role A detail" or "'Role B detail", enter the multiplicity "1..n", close the window, often only to discover that the multiplicity was added to the wrong side of the relation, so you have to re-open the window, delete the previously entered multiplicity, enter the new one, and close the window. This tedious workflow has to be repeated every time a multiplicity is added to a relation; many other editing procedures rely on similar pop-up windows, too.

UMLet's approach to such tasks is different. It makes diagram creation swift by avoiding pop-up dialogs at all, as they interrupt the workflow considerably[3]. Instead, UMLet allows for fast and intuitive UML model modifications by providing a text-based property panel, where UML element attributes are displayed and can be edited using a simple markup language.

Each UML element is described in a simple, human-readable markup string. For example, in UMLet's description of a class, the names of the class, the operations, and the attributes are simply separated by a "– –"-string. Sim-

[3]An article by P. Seebach describing modal dialog problems can be found at `ftp:// www6.software.ibm.com/ software/ developer/ library/ us-cranky12.pdf`

ilar markup strings modify other UML elements' properties like package content, relation multiplicities and types; for example, the string "<."describes a dependency relation (featuring a dotted line) and its direction.

To edit a UML element property, it is sufficient to change the markup string describing the element. In the above example, if all strings—including the separators and except the class name—are deleted, the class is displayed in its "canonical" form.

This approach has several advantages:

- Elements can be edited in a fast and unobtrusive way, as tedious modal pop-up dialogs are avoided. Editing UML element attributes is done in a text-panel, which supports all well-known keyboard shortcuts like cutting, pasting etc.

  It is especially easy to create UML element copies and modifications on them, to perform repeated tasks, and to use groups of existing UML elements (like attributes) to derive new elements that depend on them (like access methods for given attributes).

- UML elements can either be described in a very basic way, providing only few properties like, for example, only a class name, or they can be described fully, giving values to UML element properties that are used less often, for example, to a relation's qualification. Thus, simple UML diagrams can be sketched without caring for seldom-used UML element properties (in fact, they seem to be inexistent), while a few keystrokes enhance the diagram with those UML features. This is contrary to other UML tools, which tend to unleash the full UML language range right from the start.

  Furthermore, with a bit of programming, it is easy to add new markup elements in order to display new UML element attributes, for example, when the language's specification changes. Such changes are far more difficult for other applications, which provide rather hard-coded property panels to edit the element attributes.

- Using such a property panel and the markup language, it is even possible to simply change a UML element's type. For example, an existing relation's type could be changed from a dependency to an inheritance type (which is what people sometimes want to do but which is what UML tools normally don't support).

  Usually, UML tools do not allow changing a relation's type after creation, probably because of the undefined mapping of the relation's original property values to the new properties. Using a markup language for the UML element description makes this easy—the element's changed type simply causes another interpretation of the same markup; those parts which make sense for the new element type are reused in the new element description, while the other markup parts specific to the previous type are simply ignored.

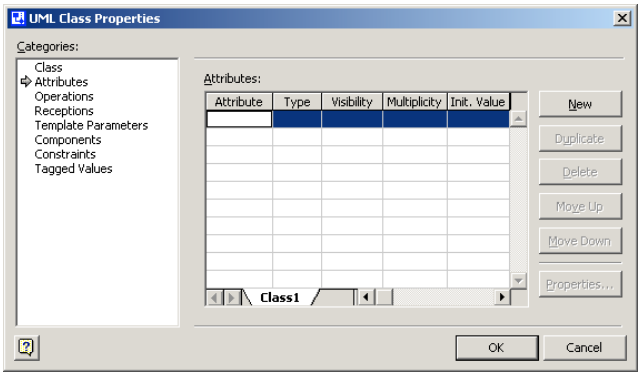

Figure 2. Microsoft Visio's pop-up windows for editing class attributes.

```
<<Stereotype>>
ClassName
--
-attribute1:Integer
attribute2
--
+operation1(param:Double)
```

Figure 3. UMLet's simple class description.

Would conventional, non-modal property panels (used, for example, in ArgoUML[4] or the Microsoft VisualStudio IDE) instead of modal dialogs solve the problems mentioned above? Some of them, but not all.

Take for instance the common case that several attributes and corresponding access methods should be added to a class. This task is difficult to perform in a structured property panel, on the other hand, it is very easy to perform in an unstructured text-based panel: just copy the attributes twice, add the prefixes "get" and "set" to the copies, and define all modified copies as class methods by copying them to the corresponding section of the class markup string.

These and other repetitive procedures are simplified by the unstructured format of the markup string, which makes

[4]The free modeling tool ArgoUML is available at `argouml.tigris.org`.

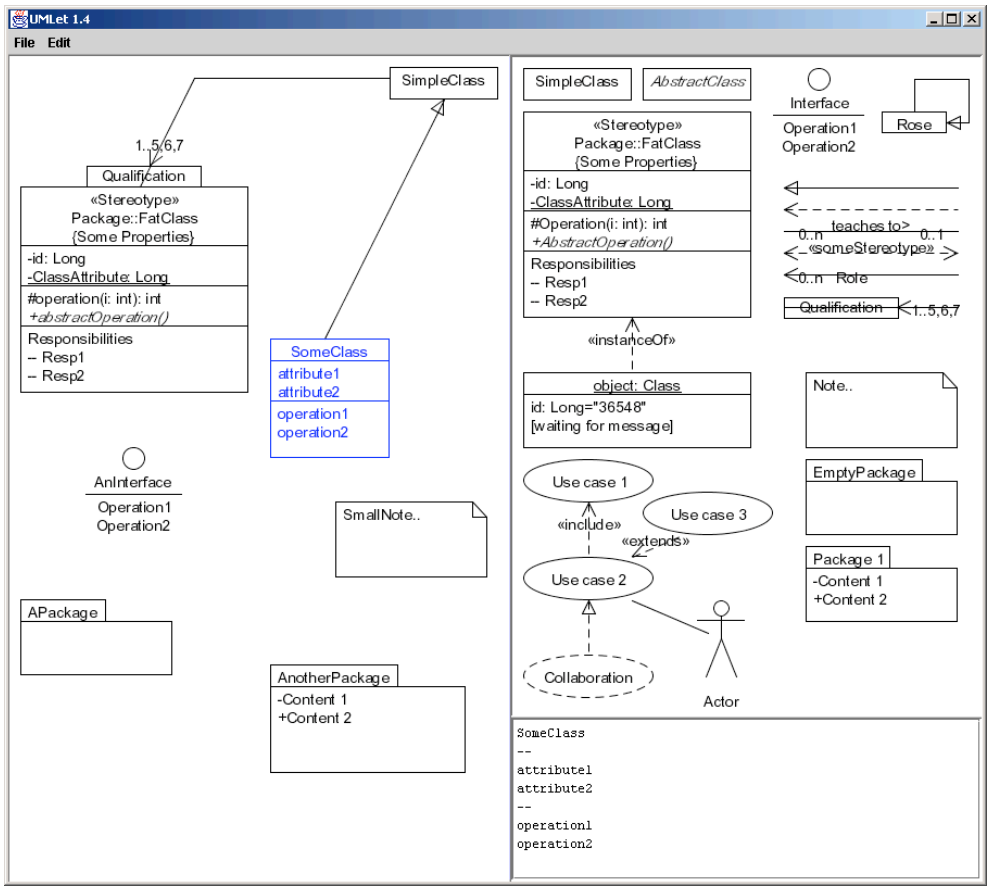

Figure 4. UMLet's element palette and property editor.

editing flexible and unrestricted by a specific property panel structure.

There is however another problem with UML editor user interfaces—the editors often provide toolbars with small and cryptic mini icons which hardly resemble their UML element counterpart.

UMLet avoids toolbars with such icons by using a palette panel, where the UML elements are displayed the same way they appear in the diagram. This speeds up diagram creation and helps both novices and UML experts to recall the UML syntax. UMLet's approach to the UML element palette was inspired by the graphic tool Glyphix on Mac OS X, which unfortunately doesn't seem to be maintained any more.

Using such a palette has another advantage: as the palette is itself just an ordinary UML diagram, it can easily be adapted to one's requirements by using UMLet to rearrange the elements on it. In addition, UMLet's internal architecture is based on a few simple design patterns like the prototype pattern, so that new UML elements can easily be created, added to the palette, and used in UML diagrams.

A screenshot of UMLet's user interface is given in figure 4. The large left part of the screen is the diagram area; this diagram contains a complex class element including stereotypes and responsibilities, as well as a simple class element in its canonical form. The markup string for the simple class just consists of the class name, hiding the more complex UML features from the user.

The upper right portion is the UML element palette. Note, that the elements in the palette are graphically identical to the elements displayed on the diagram. The palette elements can also be rearranged like the elements on the diagram, which makes palette customization very simple.

The lower right part of the screen is the text panel which allows to view and edit a UML element's markup string. Both element properties on the diagram and on the palette can be edited; common keyboard shortcuts like copy and paste are supported.

5 Print vs. Share

UML model are usually shared between developers of the same company, which are equipped with the same set of

tools. But many times parts of the models must be inserted in Word documents, processed with Latex, or just pasted as low-resolution bitmaps to a PowerPoint presentation. After all, UML is all about communication.

However, few UML tools currently support a wide range of file formats. Often one has to rely on additional third-party tools like Acrobat to process UML diagrams outside their creation tool; few tools natively support high-quality publishing formats. Often the only way to share diagrams is to print them.

UMLet features JPEG and PDF export, as well as support for scalable vector graphics (SVG), which is a promising way to publish graphical data on the Web. In addition, UMLet supports the use of a system-wide clipboard, thus avoiding the usual screenshot procedures.

Furthermore, as UMLet relies on open-source projects like Batik[5] and FOP[6], it is easy to extend this output flexibility to other file formats.

Earlier versions of UMLet even supported the XML Metadata Interchange (XMI) format, which is used to exchange UML model descriptions between tools. Future versions might again provide support for a stable XMI specification.

6 Conclusion

Many advanced UML tools are available today to support all kinds of UML modeling aspects, providing ever more sophisticated features. It is desirable for those UML tools to accurately support all UML language elements, or to provide timesaving round-trip engineering capabilities between UML models and code.

However, in many circumstances other, more basic aspects are probably even more important:

- Provide a lightweight tool that can easily be distributed and deployed in different software development environments.
- Provide a fast and intuitive user interface to quickly create UML sketches and diagrams without distracting the user with seldom-used UML language features.
- Provide flexible ways to share the UML diagrams and to reuse them in different workflows.

UMLet—a free and flyweight Java application—tries to address these issues. It is (i) small and platform-independent and therefore easy to distribute and deploy in different environments; it features a (ii) simple and customizable user interface with a fast, markup-based way of editing UML element properties; and it relies on (iii) open standards to create various output formats of the UML diagrams created.

Therefore, UMLet is a cheap and elegant alternative to traditional heavyweight UML tools. It is especially suited for educational environments, as well as for industrial development environments during early life-cycle phases, where intuitive modeling and model sharing are key issues.

Future enhancements to UMLet are likely to provide simple means of customizing the graphical elements, to further streamline the graphical user interface, and to add new UML diagram types. The code base of the upcoming UMLet version 2.0 will be released as open source in 2003.

[5] Batik is a toolset to handle scalable vector graphics (`xml.apache.org/batik`).

[6] FOP (formatting objects processor) is a framework to support output independent formatting (`xml.apache.org/fop`).

References

[1] E. Allen, R. Cartwright, and B. Stoler. DrJava: a lightweight pedagogic environment for Java. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 137–141. ACM Press, 2002.

[2] C. Damm, K. Hansen, M. Thomsen, and M. Tyrsted. Creative object-oriented modelling: Support for intuition, flexibility, and collaboration in CASE tools. In *Proceedings of ECOOP2002*, June 2000.

[3] S. L. Greene. Characteristics of applications that support creativity. *Communications of the ACM*, 45(10), October 2002.

[4] J. Grundy and J. Hosking. Software tools. In *Wiley Encyclopaedia of Software Engineering, 2nd edition*. Wiley InterScience, December 2001.

[5] J. IIvari. Why are CASE tools not used? *Communications of the ACM*, 39(10), 1996.

[6] R. Juric and J. Kuljis. Building an evaluation instrument for OO CASE tool assessment for Unified Modelling Language support. In *Proceedings of the 32nd Annual Hawaii Conference on System Sciences*, 1999.

[7] U. Kelter, M. Monecke, and M. Schild. Do we need 'agile' software development tools? In *Proceedings of the Net.ObjectDays 2002 (NODe'02)*, pages 408–421, 2002.

[8] C. LeBlanc. UML for undergraduate software engineering. In *Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 8–18. The Consortium for Computing in Small Colleges, 2000.

[9] D. Lending and N. L. Chervany. The use of CASE tools. In *Proceedings of the 1998 ACM SIGCPR Conference*, 1998.

[10] U. Nickel, J. Niere, and A. Zundorf. The FUJIBA environment. In *Proceedings of the International Conference on Software Engineering*, 2000.

[11] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[12] B. Shneiderman. Creativity support tools. *Communications of the ACM*, 45(10), October 2002.