uml2svg 0.18

user manual

Catalin Hritcu <catalin.hritcu@gmail.com>
Sergiu Dumitriu <sergiu.dumitriu@gmail.com>
Marta Girdea <marta.girdea@gmail.com>

uml2svg 0.18: user manual

by Catalin Hritcu, Sergiu Dumitriu, and Marta Girdea

Table of Contents

Chapter 1. Introduction

Scope

uml2svg is an XSLT-based tool for converting XMI-compliant UML Diagrams into SVG.

Goals

We started the development process with some *goals* in mind:

- Standard conformance
- Modularity and extensibility
- Good documentation
- Readable generated SVG (for both humans and tools)
- Support multiple diagrams per XMI-file

The Chapter 2, *Features* explains how we reach these goals.

Motivation

SVG [http://www.w3.org/Graphics/SVG/] is a standard language for describing two-dimensional vector graphics in XML. As the open SVG standard gains in popularity and gradually replaces proprietary formats for vectorial graphics, the support provided by the Web browsers is getting better. Plugins to display SVG exist for most browsers and it is most likely that the next generation of Web browser will provide built-in support for SVG. When that happens there will be no better way to distribute vector graphics on the web. Furthermore, not only web browsers can process SVG in a meaningful way; in fact that is just the tip of the iceberg. SVG can be easily read in, processed, and then transformed into many other formats, being well suited for tools, web agents and screen readers.

UML [http://www.omg.org/uml/] diagrams are composed of lines, polygons, ellipses and text labels, so they are inherently vectorial. However, the SVG is not very well suited for direct use by UML tools. While some of them can in fact export UML diagrams directly to SVG, they do that by discarding all the information about structure, and converting everything into a shape. Moreover, some tools use the screen-capture function provided by their environment (such as java2d) and then they apply a filter to generate SVG out of the “screenshot”. What comes out of that is a pile of meaningless information, which by accident happens to draw a gorgeous diagram. How will a screen reader interpret such a file? How will a web crawler be able to index it? How will a web agent process it in a meaningful way? A program needs the semantic information that the humans can extract just by looking at a picture. For a machine, an obfuscated SVG file is not easier to process than a PNG file or any other image. Although for humans it is better to be able to scale the image, for a program this is irrelevant.

Programs need a way to “understand” the semantics of the UML models to be able to process and interchange them in a meaningfull way. This was the main idea behind the XML Metadata Interchange (XMI [http://www.omg.org/technology/documents/formal/xmi.htm]), an OMG specification for model interchange. And probably the best use that XMI has found so far is the exchange of UML models between different modeling tools. And while the XMI provides a standard way for tools to represent models as XML documents, it is still limited to the model elements only.

With the introduction of the UML Diagram Interchange Specification [http://www.omg.org/cgi-bin/doc?formal/06-04-04], it will become possible for tools to exchange the models together with the layout of the diagrams. We think that, once this specification appears, XMI will be used averywhere. Not only will the tools be able to exchange diagrams, but could even represent them internaly as DOM trees. Have you ever considered drawing your UML diagrams online, using only a web browser? This could be done even now by using a custom SVG syntax for the DOM tree, but a solution based on XMI could do even better and be a standard at the same time.

Therefore, we believe that with the advent of UML 2.0 and the increase in the use of SVG, the need for transformations between XMI and SVG will be great. However when the uml2svg project was started, there was hardly any good open-source solution to convert XMI diagams into SVG.

The personal webpage of Professor Mario Jeckle [http://www.jeckle.de] provides an online transformation service [http://www.jeckle.de/UML2SVG] capable of dynamically generating SVG from XMI-compliant XML files. The XSL files accomplishing the transformations are also available on that website. These transformations are monolithic and not well documented (the only documentation is in the code, and it is generally written in German). With the tragic accident that took the life of Professor Jeckle, the transformations have no longer been maintained.

The STZ-IDA [http://stz-ida.de] research center in Karlsruhe had to convert UML diagrams to SVG, as part of one of their projects. The XSLT stylesheet [http://stz-ida.de/html/oss/xmi_diagram.html.en] they created for this purpose was named xmi2svg [http://freshmeat.net/projects/xmi2svg] and is available under the terms of the MIT license. At the time we started work on uml2svg the only type of diagrams supported was class diagrams. When the package reached version 0.2 support for more diagram types was added (without major changes in the code; the opposite of what we were expecting). Andreas Junghans, the author of xmi2svg, provided us with a lot of insightful hints which helped us eliminate many glitches in uml2svg. However, at this time, xmi2svg is no longer actively maintained.

We did not like the two existing solutions because they were:

- *incomplete* - just prototypes, not well suited for production environment
- *monolithic* - hard to maintain and extend
- *not documented* - hard to understand

At first sight, we thought we could find a way to improve one of the existing solutions and just add the features we needed. However, we slowly came to the conclusion that it would be better if we started anew. There are things one can fix in a project, but that does not include what we thought is was bad design. The fact that the two implementations presented above are open source helped us get quickly on the way with our own project.

Chapter 2. Features

Standard conformance

Many software vendors exist, each with its own ideas of how things should be done. The only way to prevent complete chaos is to agree on some standards. Not only do standards allow different tools to interoperate, but they also increase the market for products adhering to the standard. In the following subsections we will take a quick look at the more important standards involved in object-oriented modeling and the Web. Standards fall into two categories: *de facto* and *de jure*. *De facto* standards are those that have happened, without any formal plan. For example, the UNIX operating system is the *de facto* standard for operating system in the academic community. *De jure* standards, in contrast, are formal standards adopted by some standardization authority such as the Object Management Group (OMG) or the World Wide Web Consortium (W3C). Although it is sometimes hard to make a clear distinction between the two categories, the following discussion is focused on the *de jure* standards involved.

Object-oriented modeling history

In 1994, more than 50 object-oriented modeling methods were in widespread use. Though they resembled each other in terms of underlying concepts, they used different graphical notations. Three of the most important ones were:

- Object-oriented design (OOD) (Grady Booch)
- Object-oriented software engineering (OOSE) (Ivar Jacobson, 1992)
- OMT (James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenson, 1991)

The SmartDraw [http://www.smartdraw.com] website briefly describes the notations underlying OOD [http://www.smartdraw.com/tutorials/software-booch/booch.htm], OOSE [http://www.smartdraw.com/tutorials/software-oose/oose.htm] and OMT [http://www.smartdraw.com/tutorials/software-rumbaugh-omt/omt.htm].

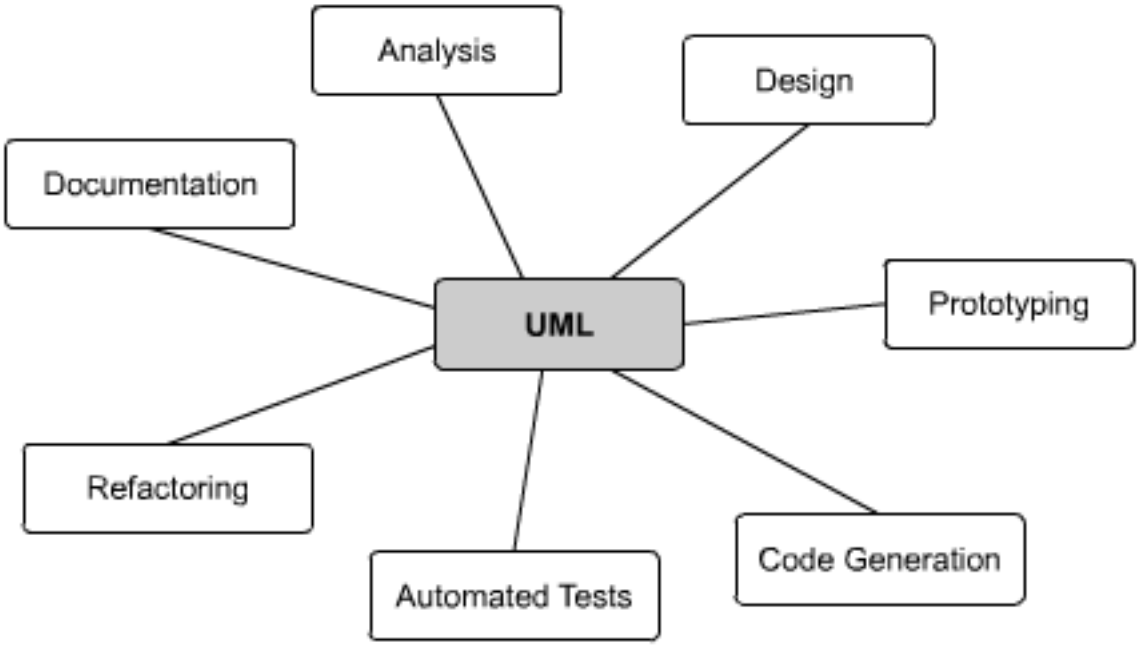
The standardization effort was started in 1994 at Rational by their original chief methodologists (Grady Booch, Ivar Jacobson and James Rumbaugh) who decided to put aside their own methods and notations and try to come out with the standard that the industry needed so badly. The final product was in fact a team effort among many partners under the sponsorship of the OMG. UML version 1.0 was finished in 1997 and ended the object-oriented method wars as it became the formal and *de facto* standard for object-oriented modeling.

UML

The Unified Modeling Language (UML) is an industry standard for modeling software. It provides an object-oriented graphical language for visualizing, specifying, constructing, and documenting a system in which software represents the most significant part. Moreover the UML brings together a collection of concepts and best engineering practices which have proven successful in modeling large, complex systems.

The UML can serve as a central notation for the software development process. Using UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. It is a programming, test and modeling language at the same time.

Figure 2.1. UML as central notation for the software development process



The UML specifies both *structural* models:

- class diagrams
- object diagrams

as well as *behavioral* models:

- use-case diagrams
- interaction diagrams (sequence diagrams and collaboration diagrams)
- statechart diagrams
- activity diagrams

Web standards

Another medium that has been recently subject to impressive standardization efforts is the World Wide Web. Although the most commonly used protocols on the Web are in fact very simple (such as HTML), the way different browsers chose to implement them varied greatly. This has led to the costly and futile practice of creating multiple versions of nonstandard markup and code, each tuned to the nonstandard “features” of a particular browser. By releasing browsers that failed to consistently support standards, manufacturers needlessly fragmented the Web, injuring designers, developers, users, and businesses alike.

At this time, the largest part of websites is obsolete. Spaghetti code, deeply nested table layouts, font tags, and other redundancies make the users wait endlessly for those pages to load, some of them

only to discover with frustration that the site is actually inaccessible to them. And this happens just because they use the “wrong” browser. Among those, most frequently hurt are people with disabilities or special needs. Moreover, the mixing of presentation and content makes automated processing hardy possible, thus making the Web hostile ground for both human and machine alike. However, you should not despair, things have started to change.

The World Wide Web Consortium (W3C) is the most important standard body that produces standards for the World Wide Web. These standards are carefully aimed at technologies that deliver the greatest benefits to the greatest number of web users, while ensuring the long-term viability of any document published on the Web. Designing and building with these standards simplifies and lowers the cost of production, while delivering sites that are accessible to more people and more types of Internet devices. Although the W3C names the documents it issues recommendations, they are the standards that everyone should adhere to when publishing on the Web.

XML

The eXtensible Markup Language (XML) is a W3C standard designed to improve the functionality of the Web by providing more flexible and adaptable information identification. XML is a markup language for documents containing structured information.

An XML document is made out of elements. An element consists of an opening label, text, and a closing label: `<subtitle>example user manual</subtitle>`. This element contains the text “example user manual”. However, an element can also contain other elements. For example the name of a person can be composed out of its first name and its surname:

```
<personname>
 <firstname>Catalin</firstname>
 <surname>Hritcu</surname>
</personname>
```

Elements can also be empty, in which case you can represent them, either as: `<void></void>` or in the shorter form `<void/>` No matter what its content is, an element can have attributes attached to it. Attributes modify information contained in elements, as an example `<person type="student">Catalin Hritcu</person>` states that this person is a student.

Example 2.1, “An XML file using the DocBook DTD” uses a custom XML vocabulary named DocBook that is intended for authoring documents such as articles and books. Following the first line of text, that emphasizes that we are dealing in fact with a XML file, there is a DOCTYPE definition referencing the DocBook DTD file. A DTD is a formal specification of the vocabulary everything in the document should conform to. Unlike earlier markup languages (like HTML) the XML lets you design your own customized markup languages for limitless different types of documents. What follows is a structure of nested elements with one root element named book that contains all the other elements.

Example 2.1. An XML file using the DocBook DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
          "http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd">

<book>
 <bookinfo>
  <title>uml2svg</title>
  <subtitle>example user manual</subtitle>
  <author>
   <personname>
    <firstname>Catalin</firstname>
    <surname>Hritcu</surname>
   </personname>
   <email>catalin.hritcu@gmail.com</email>
  </author>
  <copyright>
   <year>2004</year>
  </copyright>
 </bookinfo>
 <chapter id="intro"><title>Introduction</title>
  <para>uml2svg is an XSLT-based tool for converting
  XMI-compliant UML Diagrams into SVG.</para>
  <para>One of the most important features of uml2svg is
  <link linkend="standards">standard conformance</link>.
  </para>
 </chapter>
 <chapter id="features"><title>Features</title>
  <sect1 id="standards">
   <title>Standard conformance</title>
   <para>...</para>
  </sect1>
 </chapter>
 <appendix><title>Appendix</title>
  <para>This is an appendix</para>
 </appendix>
</book>
```

While this example might help you make a picture of what XML is, if you still feel insecure about it you should try reading a more thorough tutorial.

XMI

With the widespread use of UML a new problem appeared: *interoperability*. While the UML standard specified a standard way to draw the diagrams (many times the tool vendors ignored that too) it

didn't specify an approach to exchange the diagrams between different tools. Almost every UML modeling tool had its own way of saving diagrams to a file, which led to a plethora of proprietary and incompatible file formats. In order to eliminate this shortcoming the OMG came out with a new specification, the XMI.

XML Metadata Interchange (XMI) in an XML-based specification for interchanging UML models. Although it was build in the context of other, more general, modeling specifications of the OMG, the XMI can be used to exchange UML models between UML modeling tools. The XMI provides a standard way for tools to represent models as XML documents. However, XMI is still limited to the model elements only. With the introduction of the UML Diagram Interchange Specification, it will become possible to exchange the models together with the layout of the diagrams.

SVG

Scalable Vector Graphics (SVG) is an XML markup language for describing two-dimensional vector graphics. SVG lets you design Web pages with high-resolution graphics that can contain sophisticated elements, such as gradients, animation, and filter effects, by just using plain-text XML. This doesn’t imply that pages will take more time than they do today, as vectorial graphics have the potential to be much smaller than bitmap pictures, and additional gzip compression can be applied on SVG for excellent results. Moreover, SVG text and graphics can be styled using Cascading Style Sheets (CSS), which makes it a very flexible and powerful document standard.

In addition to the XML-based file format, the SVG platform defines an API for graphical applications. Like many other W3C standards, SVG follows the Document Object Model (DOM) standard. This means that script languages such as JavaScript can be used to access and manipulate SVG page components at runtime. The scripts are used to handle mouse and keyboard events, or even implement the logic of more sophisticated Web applications. With its powerful scripting and event handling support, SVG can be used as a platform upon which to build graphically rich applications and user interfaces. The developers get to use a collection of open standards and are not tied to one particular implementation, vendor or authoring tool.

Standards in uml2svg

uml2svg strives to provide complete conformance with the existing standards. This is the only way that interoperability with different tools can be guaranteed.

Even when these standards are not yet definitive we did our best to provide conformance with the existing drafts and with the other tools already on the market, in an attempt to ease the transition to the definitive versions of the standards. This is the case of the UML Diagram Interchange 1.0. Although OMG calls this document a “specification”, it is nothing close to that. The document is incredibly incomplete, so a more appropriate title could have been “very early draft”. No matter what opinions we might have about this document's completeness, we will keep referring to the document to the name that the OMG chose to give it.

One major difficulty in implementing uml2svg was that there are almost no tools that currently support the UML Diagram Interchange 1.0 Specification. Currently the only one we know is Poseidon for UML [http://www.gentleware.com/products.html] from Gentleware [http://www.gentleware.com/] and although the support that Poseidon provides for XMI is very good, it is still not perfect.

Version 0.18 of uml2svg relies on the following standards:
Name: Unified Modeling Language (UML)
Version: 1.5
Formal: OMG Specification
Date: March 2003
Link: http://www.omg.org/cgi-bin/doc?formal/03-03-01
Name: Unified Modeling Language (UML)
Version: 2.0
Formal: OMG Specification
Date: August 2005
Link: http://www.omg.org/cgi-bin/doc?formal/05-07-04
Name: XML Metadata Interchange (XMI)
Version: 1.2
Formal: OMG Specification
Date: January 2002
Link: http://www.omg.org/cgi-bin/doc?formal/2002-01-01
Name: UML Diagram Interchange
Version: 1.0
Formal: OMG Specification
Date: April 2006
Link: http://www.omg.org/cgi-bin/doc?formal/06-04-04
Name: XSL Transformations (XSLT)
Version: 1.0
Formal: W3C Recommendation
Date: 16 November 1999
Link: http://www.w3.org/TR/1999/REC-xslt-19991116
Name: Scalable Vector Graphics (SVG)
Version: 1.1
Formal: W3C Recommendation
Date: 14 January 2003
Link: http://www.w3.org/TR/2003/REC-SVG11-20030114
Name: JavaScript
Version: 3rd Edition
Formal: Standard ECMA-262
Date: December 1999
Link: http://www.ecma-international.org/publications/standards/Ecma-262.htm

Modularity

Programs that have many direct interrelationships between any two random parts of the code are generally harder to write, debug and maintain than programs composed out of separate modules. By grouping functionality into logical modules that communicate using well-defined interfaces one actually divides the problem into smaller, easier to solve tasks, which can be than easier implemented. And when it comes to changing the code, the physical encapsulation the modules provide helps limiting the changes to a subset of files, which is critical for the convenient extensibility of a large program.

The first step to provide a modular XSLT is to split the transformation into different template rules and named templates.

Template rules identify some source nodes to which they apply by using a pattern. Based on that information they generate result nodes, either explicitly or by calling other templates. As the XSLT processor recursively scans the source tree, it finds all the template rules with patterns that match the current node, and than chooses the best among them.

On the contrary, the named templates are not directly called by the XSLT processor when the source tree is scanned, but they can be invoked by name when processing another template. Named templates can be passed parameters explicitly when called, so they work kind of like procedure calls in an imperative programming language. Using named templates instead of template rules will usually result in clearer and more reusable XSLT files. The disadvantage of named templates is that they generate a more highly-coupled design, because of the dependencies between callers and callees, particularly when arguments are passed. They are also inconvenient when they have a lot of parameters.

uml2svg uses named templates extensively while trying to stop the number of parameters from boosting and providing meaningful default values for them. We consider named templates also easier to understand, especially for those with an imperative programming background.

In our effort to provide modularity each template is stored in a different file and the files are logically grouped into the directories shown in Table 2.1, “uml2svg modules”.

Table 2.1. uml2svg modules

/	Contains the main template, the only one that is used directly: Main.xsl
Common	Elements common to every diagram type, including styles, names, etc.
System	Templates regarding the organization and positioning of diagrams in a tree
ActivityDiagrams	Templates that are specific to a particular diagram type are stored in the corresponding directory. For example the ClassDiagrams directory contains the main template ClassDiagram; the templates to draw different types of classifiers: Class, AssociationClass, Interface, PackageClass; and finally, templates for all the elements specific to these classifiers: AttributeCompartment, OperationCompartment, Attribute, Operation, Parameter etc.
ClassDiagrams	
CollaborationDiagrams	
DeploymentDiagrams	
SequenceDiagrams	
StateDiagrams	
UseCaseDiagrams	

For efficiently reasons in a production environment the different .xsl files are always concatenated into only one file named uml2svg.xsl.

Extensibility

The easiest method to change the way the generated SVG files look like is to attach a CSS stylesheet to them. This way a user can modify the style of the diagram by changing the colors of elements, the

thickness of the lines, even hiding elements that are not needed. For more advanced transformations (that have to do with the structure of the drawing) more advanced knowledge of XSLT is necessary. One can easily modify the transformations we provide and make things appear in a different way.

To add a new type of diagram you should create a template you will call explicitly from uml2svg.xsl. It is best to group all the template files needed for your new diagram type into a special directory, and put every template files that are common to other diagram types in the Common directory. This way you ensure that the code will remain well structured.

Good Documentation

This user-manual is the main part of the documentation of uml2svg. The source code is also available, so that the most experienced users can always examine it and experiment with it. That is why we have tried to keep things simple and clear, while including comments whenever that was not possible.

Finally, the uml2svg website [http://uml2svg.sourceforge.net] is the best place to find the latest news about the evolution of the project or get in touch with us. It is there you should ask for help if you get into trouble using uml2svg. By doing this not only will you find a solution to your problem, but also supply the motivation we need to keep improving the software.

Readable generated SVG

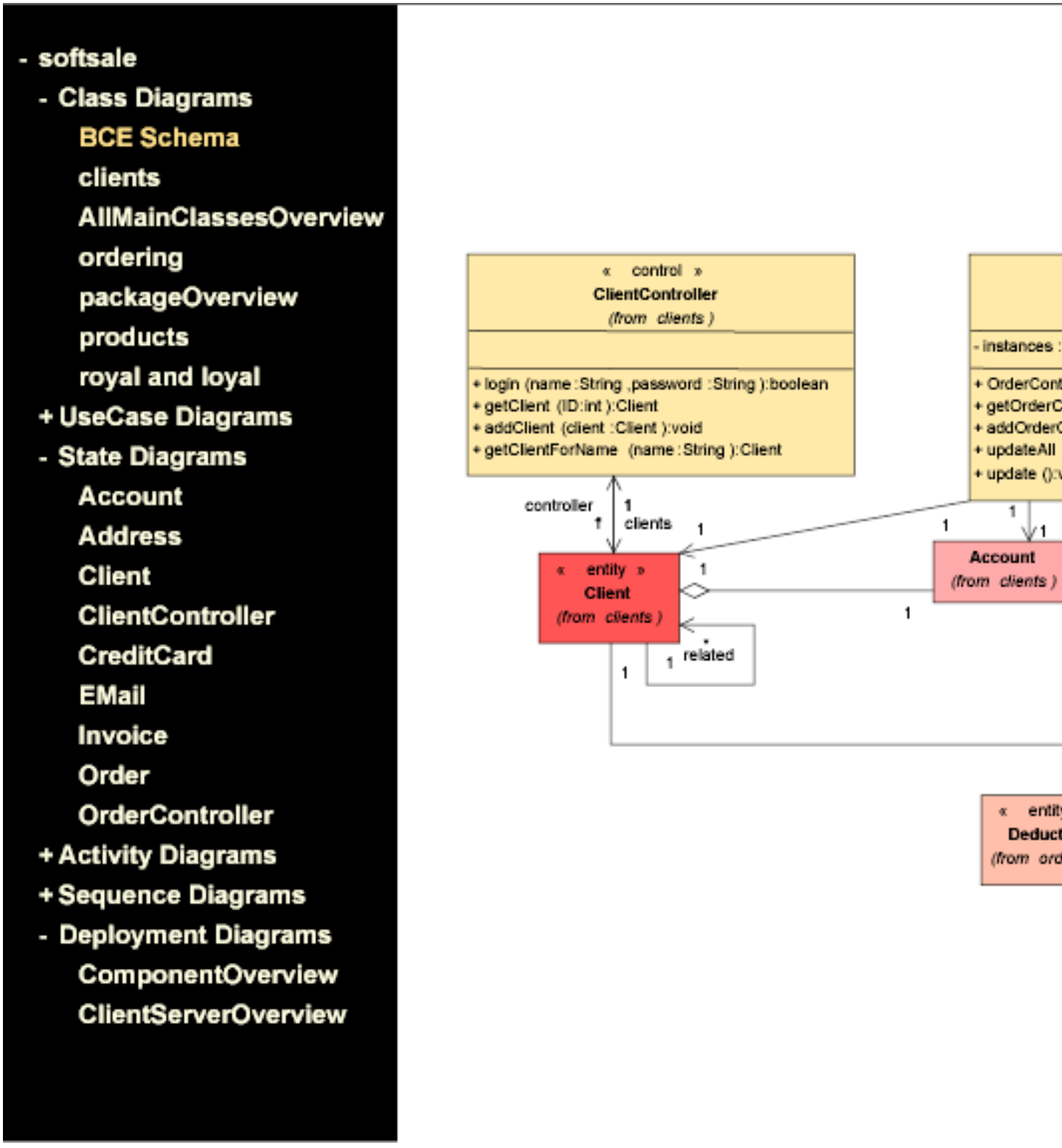
As we explained in the introduction, semantic information should not be discarded when generating SVG out of UML Diagrams. To ensure this uml2svg groups diagram nodes and edges with the *g* (group) element. This is most helpful for making the generated SVG more human readable but at the same time more easy to process by tools. It also makes our generated SVG files smaller and our XSL transformation easier to write as many style attributes can be applied only once on a group, rather than on every child of a component.

uml2svg also sets the *id* attribute of every group to the id of the node/edge in the diagram. This way, someone writing a CSS stylesheet can exactly pinpoint the diagram element to which he wants to apply a custom style. The *desc* and *title* elements are also used to provide a textual description of the group. When a SVG document is rendered on screen or printed, the desc and title elements are not rendered as part of the graphics. User agents may, however, display the title element as a tooltip, and screen readers could read it aloud. This approach leads to no loss in the semantic information provided by the diagram. The UML model itself is however discarded because SVG is not well suited for holding such information in an appropriate way.

Multiple diagrams per XMI-file

Although XMI files can contain only one model, multiple diagrams may be referring to it. If that is the case and uml2svg is called without parameters, it will export a SVG file that contains all the diagrams. A tree with all the existing diagrams, sorted by their type, is provided and the user can choose the one to display (see Figure 2.2, “uml2svg generated SVG containing a tree with all the existing diagrams”). On the other hand, a parameter called SelectedDiagram can be passed to the transformation to select only a diagram, in the case more of them are present.

Figure 2.2. uml2svg generated SVG containing a tree with all the existing diagrams



Chapter 3. Choosing the right edition

Online Edition

Our website offers a free transformation service based on uml2svg: http://uml2svg.sourceforge.net/online. This is the ideal approach when you wish to try uml2svg without the trouble of having to install an XSLT processor. While this solution works fine when you have only a couple of small XMI files you wish to process, when your files are large or your Internet connection is slow you will have to use the Standalone Edition. Another reason for not choosing the online edition is security. If the diagrams you wish to process are supposed to be secret for your corporation, than sending them over the Internet is not a wise decision at all.

The Online Edition of uml2svg is actually different from the Standalone Edition in only one way. It is run on our web server via a script. We have installed the Standalone Edition ourselves so that you can access it via a web interface.

Standalone Edition

The Standalone Edition is the main edition of uml2svg that you can download from: http://uml2svg.sourceforge.net/download

Although there might be multiple versions available you will usually want to download the latest. You can also chose the archiving method: Windows users will usually prefer `zip` files over `tar.gz` files that make the delight of UNIX users. There are no structural differences between the archives; just the compression method is different.

So, aside from these issues, what you need to remember is that uml2svg only comes in only one flavor: XSLT source code. And you will need an XSLT processor to make it work.

Chapter 4. System Requirements

Online Edition

Although the Online Edition of uml2svg requires no installation on your computer, there are two things you will need to ensure before you can use it properly: a fast internet connection and a SVG-enabled web browser. If your web browser does not support SVG (and many browsers don't) you will have to install a specific plug-in (see the section called “SVG Viewers”).

Standalone Edition

uml2svg is operating system independent. It has been used successfully on UNIX, Mac and Windows, and it will work just the same on any other operating system. The first prerequisite for using the Standalone Edition of uml2svg is an XSLT processor. Some modern web-browsers offer some limited support for XSLT transformations, usually via JavaScript. This support is not sufficient to run uml2svg. You need *a real XSLT processor* (see the section called “XSLT processors”).

The second prerequisite is a viewer for the SVG files that result from the conversion (see the section called “SVG Viewers”).

SVG Viewers

Here are three SVG Viewers that come for free:
Name: Adobe SVG Viewer
Comment: Browser plugin for Internet Explorer and Netscape Navigator. (Freeware)
Link: http://www.adobe.com/svg/viewer/install/main.html
Name: Apache Batik Squiggle
Comment: SVG-only browser, part of the Batik toolkit. (Open Source, Apache Lincese)
Link: http://xml.apache.org/batik
Name: Firefox
Comment: Starting with version 1.5 Mozilla added SVG support for their browser (Open Source, Mozilla License)
Link: http://www.mozilla.org/projects/svg

XSLT processors

Many good XSLT processors are freely available. You will need one when using the Standalone Edition of uml2svg. the section called “Installing an XSLT processor” explains in detail how to install three of them.
Name: Apache Xalan-C++
Comment: based on Apache Xerces-C++ for parsing
Link: http://xml.apache.org/xalan-c
Name: Apache Xalan-Java
Comment: very popular Java XSLT processor that comes together with the recent Java SDKs
Link: http://xml.apache.org/xalan-j
Name: MSXML

Comment: not really free software, but comes as a bonus with Windows XP or later
Link:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/xmmscXML.asp
Name: Sablotron
Comment: based on expat for parsing
Link: http://www.gingerall.com/charlie/ga/xml/p_sab.xml
Name: Saxon
Comment: one of the best XSLT processors
Link: http://saxon.sourceforge.net
Name: xsltproc
Comment: based on libxslt (so indirectly on libxml2 for parsing)
Link: http://xmlsoft.org/XSLT/xsltproc2.html

Installing an XSLT processor

This section describes how to install three of the free processors on Windows and Linux. The instructions provided here are informational only and the only definitive source of information on the projects is the official websites and documentation. You should check those sources as the steps presented here may change over time.

Installing xsltproc

The installation of xsltproc is platform dependent since it is a compiled C program.

Installing xsltproc on Windows

You can download precompiled versions for Windows from Igor Zlatkovic's website [http://www.zlatkovic.com/libxml.en.html]. The website also describes how to install the files and use xsltproc on Windows. You need to download at least the following packages:

- libxml2, the XML parser and processor.
- libxslt, the XSL and EXSL Transformations processor.
- iconv, the character encoding toolkit.
- zlib, the compression toolkit.

Once you have unpacked the .zip files, the PATH variable must include the locations of the `xsltproc.exe` file and the following dynamic load libraries: `libxslt.dll`, `libxml2.dll`, `libexslt.dll`, `iconv.dll` and `zlib1.dll`. Since they install into separate directories, it is perhaps simplest to just copy the files into a standard location. For example, find and copy all the files into C:\Windows\System32.

You will know it is working if you can execute the following command to list the version information:

```
xsltproc -version
```

Installing xsltproc on Linux

If you are running a recent version of Linux, there is a good chance you will already have xsltproc installed on your system. Try the following command to see if you do:

```
xsltproc -version
```

If the command fails, you can install the files you need using the RPM packages: libxml2 [http://rpmfind.net/linux/rpm2html/search.php?query=libxml2] and libxslt [http://rpmfind.net/linux/rpm2html/search.php?query=libxslt]

Just run the following commands while logged in as root (don't forget to substitute the file names with the ones you actually downloaded):

```
rpm -Uv libxml2-2.6.16-5.i386.rpm
```

```
rpm -Uv libxslt-1.1.12-3.i386.rpm
```

Type `xsltproc -version` once more to see if the installation worked.

Compiling xsltproc

If you cannot find a precompiled version of xsltproc for your platform, or if you want the very latest version, then you can compile it yourself from source. It is pretty easy to compile xslproc if you use the GNU compiler. That compiler is generally available on all Linux distributions, and is also available for many UNIX systems. You might need to search the Internet to find one for your system if it doesn't already have one.

Once you have gcc set up, download and unpack the latest xsltproc source archives from http://xmlsoft.org/XSLT. To run the xsltproc processor, you need to download the libxml2 and libxslt packages. Then do the following:

1. Unpack the tar.gz archives:

   ```
   tar zxvf libxml2-2.6.16.tar.gz
   ```

   ```
   tar zxvf libxslt-1.1.9.tar.gz
   ```

2. Compile libxml2:

   ```
   cd libxml2-2.6.16
   ```

   ```
   ./configure
   ```

   ```
   .make
   ```

   ```
   make install
   ```

   ```
   cd ..
   ```

3. Compile libxslt:

   ```
   cd libxslt-1.0.18
   ```

   ```
   ./configure
   ```

   ```
   .make
   ```

   ```
   make install
   ```

You will need to have root permission to run the make install step. If these steps proceed without error, you should be able to run this command to test it:

```
xsltproc -version
```

If you get a Command Not Found error message, then you need to find where xsltproc is installed and add that location to your PATH environment variable.

Installing Saxon

Saxon comes in two packages: Saxon-B ("basic") and Saxon-SA ("schema-aware"). The main difference between Saxon-SA and Saxon-B is that Saxon-SA supports XML Schema validation. Only Saxon-B is an open source product so you will probably chose Saxon-B. To use uml2svg you don't need any of the more advanced features of Saxon-SA so we will describe only the installation of Saxon-B.

Installing a JavaVM

Saxon is a Java-based XSLT processor so your system must have a Java Development Kit (JDK) installed. Saxon 8.2 will run with JDK 1.4 and JDK 1.5. It is likely that future Saxon releases will require JDK 1.5, so we advise you to use this version. You can find out which Java version is installed on your system by executing `java -version`. If you get an error or an earlier version number you will need to install a JDK, which is available for download from Sun Microsystems [http://java.sun.com/j2se]. The installation process is fully automated, but if you run into trouble you can consult the Installation Instructions [http://java.sun.com/j2se/1.5.0/install.html]

If you choose to install JDK 1.4 (or you have it already installed on your system) and want to use Saxon, you will have to additionally install JAXP 1.3. You can download it from java.net [https://jaxp.dev.java.net]. This download includes two JAR files: `jaxp-api.jar` and `dom.jar`, which must be added to the class path (also see the section called "Updating the Class Path").

Downloading Saxon

Saxon-B can be downloaded from the SourceForge page [http://sourceforge.net/project/showfiles.php?group_id=29872] as a single `.zip` file.

Uncompressing

Installation of Saxon simply involves unzipping the supplied download file into a suitable directory. For this purpose you can use the Jar archiver that comes with Java:

```
cd /usr/saxon
```

```
jar xf saxonb8-2.zip
```

Updating the Class Path

One of the files that will be created in this directory is `saxon8.jar`. When running Saxon, this principal JAR file should be on the class path. The class path is normally represented by an environment variable named CLASSPATH: see the Java documentation

[http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/classpath.html] for details. Note that the JAR file itself (not the directory that contains it) should be listed on the class path.

To update your CLASSPATH on Linux, put these lines in your `.profile` file (replacing the directory with the one you used to extract the Saxon-B .zip file):

```
CLASSPATH=$CLASSPATH:/usr/saxon/saxon8.jar

export CLASSPATH
```

On Windows, use the Control Panel to open the System icon, where you can set environment variables for Windows. Use semicolons instead of colons to separate CLASSPATH items on Windows. Once again you are encouraged the see the Java documentation [http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/classpath.html] for details.

Installing Xalan

Starting with JDK 1.4 Xalan is bundled with Java. If you are running Java 1.4.0 or newer, you do not need to install Xalan at all. If you are using an older version of Java, you will need to install Xalan yourself. Even if you have a current Java version, you may want to install a newer version of Xalan than the one that is bundled with it. You can find out which Java version is installed on your system by executing `java -version`.

Installing a JavaVM

Since Xalan is a java program you must have a Java Runtime Environment (JRE) or Java Development Environment (JRE) installed. Xalan 2.6.0 requires JDK/JRE 1.2.2 or later to run, however we advise you to install the latest JDK. (see the section called “Installing a JavaVM” part of the section called “Installing Saxon”)

Downloading Xalan

To download Xalan-J, go to http://xml.apache.org/xalan-j and locate the latest stable binary version for download. As of version 2.6.0, Xalan is available in two binary distributions, one that includes the Xalan Compiled processor (XSLTC) in `xalan.jar`, and in the other, it does not. Either one will work fine for running uml2svg. The descriptions below are valid no matter which one you decided to use.

Be sure to get Xalan Java (Xalan-J), not the Xalan C++.

Uncompressing

Xalan-J is packaged as a `.zip` file which can be unpacked using the Java jar command:

```
cd /usr

jar xf xalan-j_2_6_0-bin.zip
```

This command will create a sub-directory called `xalan-j_2_6_0` in the `/usr` directory containing the Xalan distribution.

Endorsing the `.jar` files.

If you have Java version 1.4.0 or later, you already have a working Xalan. But if you want to install a newer version, you need to put the files `xalan.jar`, `xml-apis.jar` and `xercesImpl.jar` in the `lib/endorsed` directory in your Java directory. Then the new Xalan will be used in place of the built-in Xalan. If you don't register the newer Xalan as the endorsed version, then your processing will use the older version of Xalan and you may not get the results you expected.

Alternatively you could specify an option to force the use of the the newer versions: *`-Djava.endorsed.dirs=/usr/xalan-j_2_6_0/bin`* every time you start the java virtual machine. You should replace `/usr/xalan-j_2_6_0` with the directory that contains the Xalan's `.jar` files.

More information about the endorsing mechanism is provided by the Java documentation [http://java.sun.com/j2se/1.5.0/docs/guide/standards].

If your Java version is earlier than 1.4.0, you can put the `.jar` files in any convenient location for creating a CLASSPATH (directory names don't contain spaces). You don't have to worry about the endorsing process, unless you upgrade your Java at a later date to 1.4 or higher.

Updating the Class Path

If your Java version is earlier than 1.4.0 you need to include the full path to the `.jar` files in your CLASSPATH environment variable (see the Java documentation [http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/classpath.html] for details). To update your CLASSPATH on Linux, put these lines in your `.profile` file:

```
CLASSPATH=$CLASSPATH:/usr/xalan-j_2_6_0/xalan.jar:/usr/xalan-j_2_6_0/xml-ap
/usr/xalan-j_2_6_0/xercesImpl.jar

export CLASSPATH
```

On Windows, use the Control Panel to open the System icon, where you can set environment variables for Windows. Use semicolons instead of colons to separate items in the CLASSPATH. Once again you are encouraged the see the Java documentation [http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/classpath.html] for details.

Chapter 5. Using uml2svg

Online Edition

Using the online edition is as easy as one, two, three:

1. Open your internet browser and go to http://uml2svg.sourceforge.net/online
2. Choose the XMI file you wish to convert
3. Click the submit button and be prepared to wait

The time you will have to wait can vary greatly depending on the size of the file, the speed of your connection, the traffic load on the Internet and, most important, the load on our server. It can take anything between several seconds to a couple of minutes. As with all the free services on the web this is a best effort service, so the quality-of-service is not guaranteed. If you think that the transformation takes too much, you can use the Standalone Edition. You can even host your own transformation service on your own 1024-processor server and provide free access for everyone.

Standalone Edition

Installing

Before you can use the Standalone Edition of uml2svg you have downloaded from: http://uml2svg.sourceforge.net/download you will have to extract the contents of the archive to a directory of your choice. You are done! The Online Edition needs no installation at all as it will run on the server.

Command prompt

Most XSLT processors allow you to perform a transformation by entering the right thing at the command prompt. “The right thing” is however dependent on which XSLT processor you are using. Suppose your diagram is in a file called `Diagram.xmi` located in the same directory you installed uml2svg, we give you the command to obtain the corresponding `Image.svg`.
Processor: xsltproc
Command: `xsltproc --output Image.svg uml2svg.xsl Diagram.xmi`
Processor: Apache Xalan-Java
Command: `java org.apache.xalan.xslt.Process -IN Diagram.xmi -XSL uml2svg.xsl -OUT Image.svg`
Processor: Saxon
Command: `java net.sf.saxon.Transform -o Image.svg Diagram.xmi uml2svg.xsl`

By default Saxon uses the XML parser that comes with Java but it can be set up to use a wide variety of XML parsers. All the relevant classes must be installed on your Java class path and the -x and -y options select the parser used for source files and the XSLT file respectively. Here is an example using Xerces:

```
java net.sf.saxon.Transform -o Image.svg
-x org.apache.xerces.parsers.SAXParser -y
org.apache.xerces.parsers.SAXParser Diagram.xmi uml2svg.xsl
```

Parameters

Parameters may be used to customize the uml2svg transformation. They can be passed as command line arguments when invoking an XSLT processor. For example the parameter that selects the name of the diagram that is to be processed by uml2svg is *`SelectedDiagram`*. Suppose you want to extract the "ClassDiagram1" class diagram from `Model.xmi`. Depending on the XSLT processor, the transformation could be invoked like this:

```
xsltproc --output ClassDiagram1.svg --stringparam SelectedDiagram
ClassDiagram1 uml2svg.xsl Model.xmi
```

```
java net.sf.saxon.Transform -o ClassDiagram1.svg Model.xmi
uml2svg.xsl SelectedDiagram=ClassDiagram1
```

```
java org.apache.xalan.xslt.Process -IN Model.xmi -XSL uml2svg.xsl
-OUT ClassDiagram1.svg -param SelectedDiagram ClassDiagram1
```

These commands will write only one diagram into the SVG file and not generate a navigation-tree, which will make it more easily usable for further editing and publishing.

Other two parameters *`SelectedDiagramNumber`* and *`SelectedDiagramId`* allow the selection of a diagram by its number or XMI identifier. The diagram number is given by the sequence of the diagrams in the file; the first position is 1. The XMI identifier of a diagram can be obtained from the is *`xmi.id`* attribute of the desired UML:Diagram element. While these two methods seem more difficult to use than *`SelectedDiagram`* they are very helpful for automated processing.

Another useful parameter is *`UmlStdVersion`* that allows you to select the version of the the UML standard to use (i.e. 1.5 or 2.0).

Programming APIs

Additional to running XSL transformations on the command line, most XSLT processors offer an API which can be called from a programming language. No matter which programming language you are using there will probably be a way to do XST transformations. Other than the tools described in the section called "XSLT processors" you can also use:
Name: libxslt
Link: http://xmlsoft.org/XSLT
Name: The XML::XSLT Perl module
Link: www.cpan.org
Name: xslt module for PHP
Link: http://us2.php.net/xslt

Chapter 6. Feedback

Although we already provide a working solution for the conversion of XMI to SVG, the work on the uml2svg project is far from over. You can use uml2svg to prepare your diagrams for the web. You can integrate what we have built into more advanced software. You can add new features and extend it in any way you like. The source is released under the terms of the LGPL so feel free to experiment.

It is our turn to ask for your assistance. You can offer the valuable feedback we need to keep improving uml2svg by sending us your opinions and comments, requesting new features or reporting bugs. We assure you that we will be responsive to your needs as long as you communicate them. To do this you can use: mailto:uml2svg@gmail.com

Appendix A. GNU Free Documentation License

Version 1.2, November 2002

Free Software Foundation, Inc.
59 Temple Place, Suite 330,
Boston,
MA
02111-1307
USA

Version 1.2, November 2002

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying In Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

GNU FDL Modification Conditions

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Addendum: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Sample Invariant Sections list

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

Sample Invariant Sections list

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.