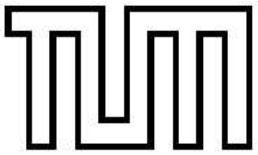
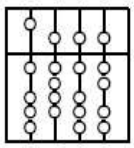


Technische Universität München
Fakultät für Informatik



System Development Project

Development of a Dynamic Visualization using Java2D

Anton Kostov

Aufgabensteller: Univ.-Prof. Dr. Dr. h.c. Manfred Broy

Betreuer: Dipl.-Inf. (Univ.) Sebastian Winter

Abgabedatum: 22. November 2005

Abstract:

The project goal was to develop a dynamical visualization of the great amount of information that a todays auto produces. The format and the presentation of this information are very important and one of the most used tools that help the driver is the Comboview which is usually placed in front of him and displays speed, rpm, oil consumption etc. To improve the interaction between the driver and the car such a tool was developed and integrated into the car-simulator developed from the Department for Ergonomics's (DfE) at the TUM.
Different Comboview templates can be created easily using some graphical editor that have support for the SVG format like Illustrator and thus allowing us to build quickly different user interfaces and explore which is better suited. The project goals were to create such templates, build a viewer that can load them and dynamically update the displayed information and also to deploy it as a service into the car-simulator. The viewer can be used either standalone or as a part of the DWARF system and is based on Batik SVG project.

Table of Contents

1 Introduction

1.1 Motivation

Modern automobiles contain a wide variety of interactive accessories. Although these accessories are designed to enhance the driving experience, they can sometimes draw attention away from the real job behind the wheel, piloting the vehicle. While elaborate stereo systems, climate controls, and even cellular telephones can be used safely during most driving situations, in demanding situations these accessories can possibly become an untimely distraction. However, drivers have grown up with these accessories over time and are accustomed to using the systems found in automobiles today. They have also found a responsible way to manage these systems on their own, and in most situations, they do a good job.
But driver inattention is the most prevalent primary cause of collisions, accounting for an estimated 25-56% . To be able to assist drivers, we need to be able to collect real-time data on driver visual behavior, recognize what the driver is doing (contextual information such as maneuvers, actions, and states), predict what the driver would likely do next, and assist the driver (design an interface). The importance of context is underlined. This system design project tries to help in one of the most important problems in the design of a usable interface and for the dynamic interaction with the driver. This interface can be changed dynamically according the current context in which the driver is situated if we have some sort of driver disattention like talking to other person or not looking at the road thus missing a speed shield we have to update the current interface and to notify the driver and thus to improve the interaction between the machine and the person.
The TUMMIC project (Thoroughly User-centered Man-Machine-Interaction in Cars) is investigating lots of this issues and has build a car-simulator to be able to better explore them. Other important subjects that usually appear when we are talking about improving HMI in the cars are:

- **Rapid Prototyping**: developing new display-interaction-concepts for undeveloped technologies requires easy configurable solutions for evaluation
- **Concept Evaluation**: testing concept for acceptance and support for the driving task requires massive logging of time and task relevant data
- **Technology**: current Head-Up display technology is not capable of stereoscopic rendering. New hardware has to be developed and calibrated
- **Spatial Context**: spatial movement in vehicles requires knowledge about other vehicles and obstacles in the drivers environment
- **Easier Navigations:** alternative methods for input like entering text or using touchscreen front panel to navigate quickly over the tasks that require immediately driver attention.

1.2 Goals of this work

This SEP is inspired to solve some of this common problems and to allow quickly prototyping of new user interfaces, dynamically updating the information on the display according the

current context, to allow easily handling alternative methods for navigation like entering text or using touchscreen displays.
During the development a Comboview control one of the most important tools that assist the driver was developed and integrated into the car-simulator and various templates for this panel were developed and tested. Also a concept was created for the easier developing of such rich view components and was successfully implemented. The principle actually is quite simple you create the desired user interface with your favorite graphics editor that support the SVG file format This can be done actually by anyone you do not have to be a graphics designer or 2D artist or have university diploma. Then group the elements according to the desired behavior that they have like speedometer, left or right knob or auto-pilot button, after that set unique Ids for every such element using any XML editor and write a simple wrapper using Java or Javascript that changes dynamically the desired values of the components.

1.3 Dwarf Overview

The car-simulator is built on the top of DWARF so the following section gives a brief overview of the aspects relevant to creating the Comboview Service but before I can go on explaining the service, it is necessary to understand the basic concepts. For further reading, an in-depth discussion of DWARF can be found in [1, 7, 14].
A DWARF system consists of several cooperating services, located on different stationary or mobile computers. Each service has a set of needs and abilities, which describe how the service can communicate with other services. The most important property of a need or ability is the protocol that is used for communication. In addition, an arbitrary number of attributes can be defined which give more detailed information about what kind of data an ability provides or a need expects. To limit the selection of communication partners, predicates can be given, which the other service has to fulfill. When the supported communication protocols of a need and an ability match and all predicates are satisfied, the DWARF middleware dynamically connects the two services.
DWARF is designed for distributed applications, so this works across the network, but preference is given to local services. When new services are added to the system or existing ones are removed from it, the middleware reacts to these changes and adds or removes connections between services.The part of the DWARF middleware which is responsible for locating and connecting services is called the service manager. Usually one instance is running on each computer, acting on behalf of the local services and advertising them on the network.

1.4 SVG Overview

The following section describes what SVG is and why it is very convenient to use it as for rich 2D templates. SVG stands for SVG = Scalable Vector Graphics and is an XML format for rich, interactive 2D graphics and is developed as a W3C activity that is supported by companies like Sun Microsystems, Adobe, Kodak, Nokia and many others. More and more major suppliers of graphic tools (e.g. Adobe Illustrator, Corel Draw, Together) make export to SVG possible. The last specification is SVG 1.1 from 4 January 2003 more info can be found at http://www.w3.org/TR/SVG/. The goal that SVG aims to follow is to be a rich interoperable graphics format for a wide range of applications and platforms there exists profiles for the desktop (SVG Full) and mobiles (SVG Tiny and SVG Basic).

SVG Key Features are:

- Basic and complex shapes
- Rich paint styles (gradients, patterns)
- Rich text
- Searchable and zoomable
- Opacity
- Filter effects
- Scripting and Animation for dynamic content
- Internationalization (i18n)

SVG is important because enhances the end-user's experience, reduces graphical content management cost is easy to generate and manipulate, blends well with other XML technologies and is useful for both client-side and server side and at last but not least SVG means extremely portable graphical data.
The world of SVG is an infinite canvas. The area of the canvas that the document intends to use is called the viewport. Its size is specified in the height and width attributes of the <*svg*> element. The units can be pixels (default), points, centimeters, millimeters, inches etc.
Defines six basic shapes :

- **circle :** displays a circle element
- **eclipse :** displays a eclipse element
- **line :** display a line element
- **rect :** displays a rectangular element
- **polyline :** displays a series of lines with vertices at the specified points.
- **polygon :** similar to polyline, but adds a line from the last point back to the first, creating a closed shape.

It has a standard text element and the the path element is the most flexible. It has one attribute d (for data). *<path d="M 40 60, L 10 60, L 40 30, Z, M 70 80, L 100 100"/>* .This sample means move to 40 60 then line to 10 60 ... .
SVG shares many of its styling properties with CSS (Cascading Style Sheets) and XSL. Thus makes it extremely easy to change the look and feel of the current template by just providing different values for the CSS style attributes. For a more detailed description for SVG please look further at the [11, 12, 16 , 18].

1.5 Batik Overview

We just cannot go further without a small introduction in the used toolkit for rendering the SVG templates. Batik is a Java language SVG toolkit that started summer of 2000 and is supported by Sun Microsystems, Inc., Eastman Kodak Company, Koala team LOG, CSIRO. It aims to deliver a Java Language toolkit to help developers generate, create, manipulate, view and transform SVG content.
The Batik modules are of one of three types 1.1:

- Application Modules
- Core Modules
- Low Level Modules

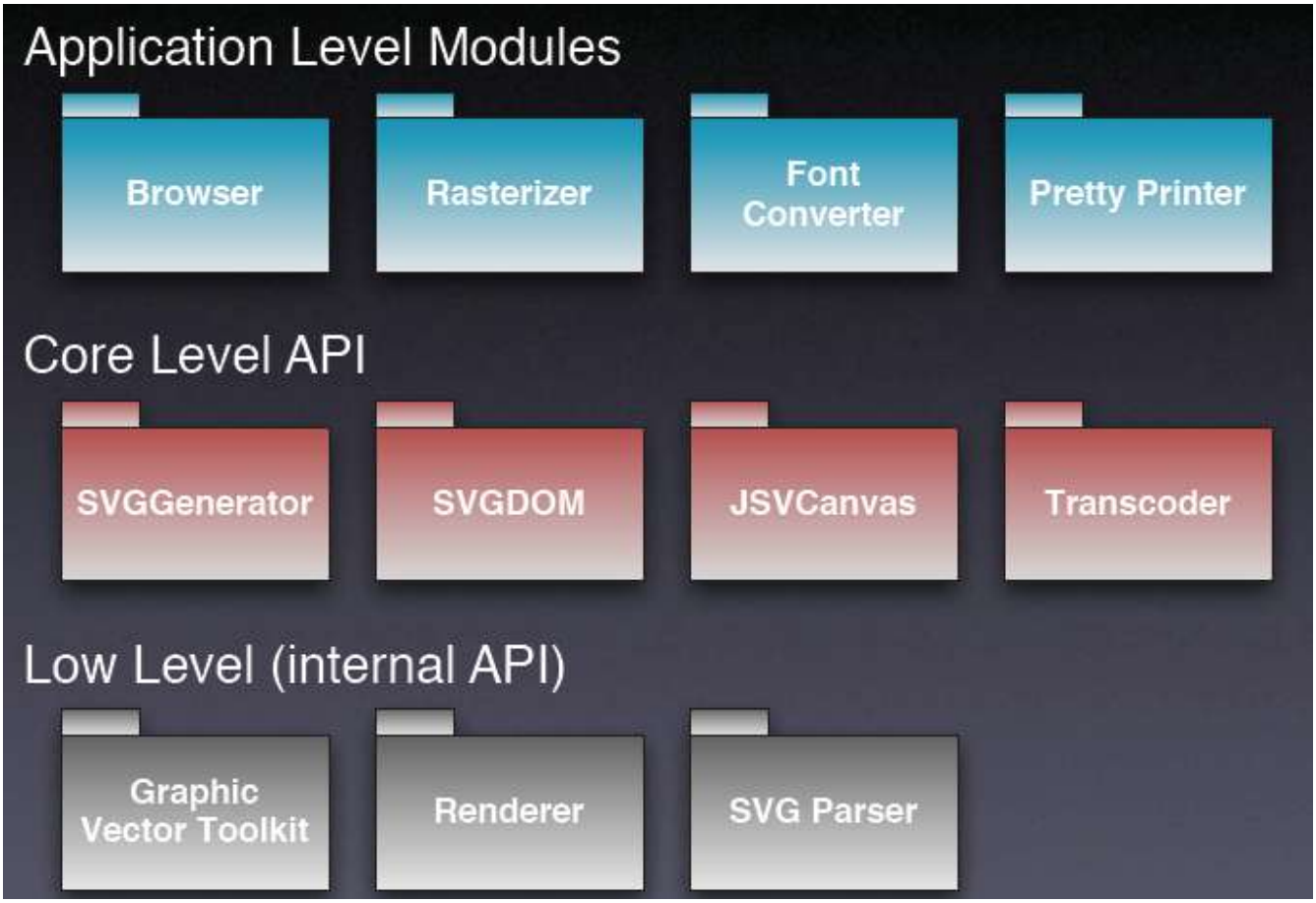


Figure 1.1: Batik Architecture

The Batik core modules are the heart of the Batik architecture and the ones used mainly and by us are

- **SVG Generator** is a modules which contains SVGCanvas2D that lets all Java technology applications easily convert their graphics to the SVG format, as easily as they draw to a screen or a printer, by leveraging the Java 2D API's extensible design.
- **SVG DOM** an implementation of the SVG DOM API defined in the SVG recommendation. It lets the programmer manipulate SVG documents in a Java program.
- **JSVGCanvas** is a UI component that can display SVG content and let the user interact with that content (zoom, pan, rotate, text selection, etc...)
- **Transcoder** is a module that provides a generic API for transcoding an input to an

output. This module transcodes an input stream or a document into a particular output format (e.g. conversion to jpeg).

And if you are curious about the license issues The Apache License allows you to download and use for free, also to modify and redistribute (even for commercial purposes) but does not guarantee anything else, and also provides legal protection for developers (IP and patent issues etc.) Just one restriction: You must state that you're using software created by the ASF.

1.6 Structure of the document

The project documentation contains a nine chapters in total including one for introduction to the system one for used references and one for an appendix. The other six are summarized bellow:

- **Chapter 2** describes in details all the requirements for the dynamic display of such rich view components. The functionality of project was also described from the a user's perspective.
- **Chapter 3** gave an overview of the system design, we will see how the project is divided into into three subsystem and how the main design goals - extensibility performance, and reusability - are achieved. The actual implementation will be described also in this chapter.
- **Chapter 4** describes in details how to setup a working demo and to successfully deploy and start the service or the standalone application. Include all needed information and configuration parameters that can be usually set on the working environment.
- **Chapter 5** is important for everyone that wants to deploy extensions for the DWARF system or to create nifty SVG templates. A number of tools are described, and also how the special Comboview components are created like the speedometer or the oilmeter.
- **Chapter 6** focused on the future of Comboview tool. Despite of it's current usefulness, there still are many extensions which could make the project even more valuable. A number of additional features were proposed.

2 Requirements for the dynamic visualization

This chapter gives a more formal and more detailed description of the requirements for a dynamic visualization tool. The chapter's structure roughly follows the guidelines for Requirements Analysis Documents (RADs) given in [10]. Readers who are new to Comboview service, should read the section about functional requirements in order to get an overview of the functionality of the tool. If you want to learn how to use the service, reading the use cases is recommended as they show step-by-step how to operate the application and also reading the chapter about setting up the demo is highly recommended.
Some sample usage scenarios are also shown which were used in the created demonstrations of the project.

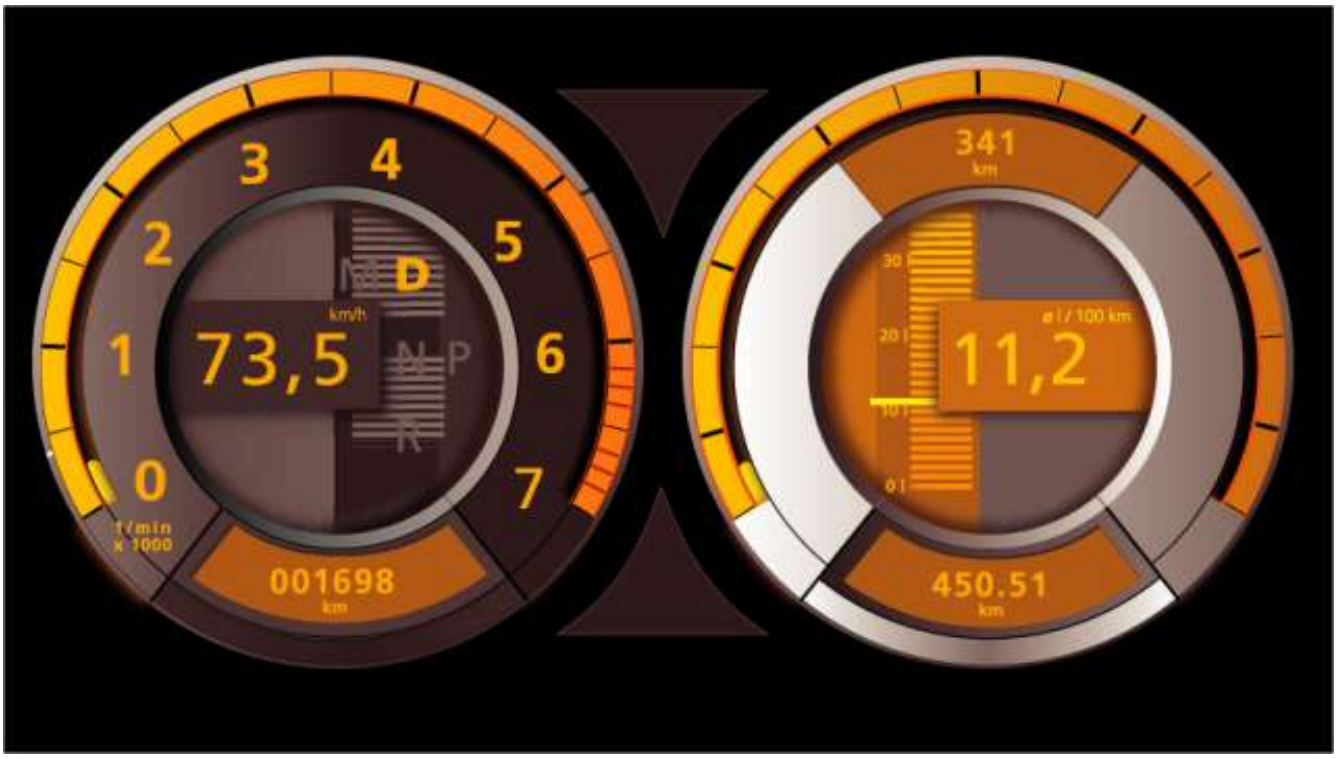


Figure 2.1: Initial Requirement

This is a picture of the initial requirement for the look that should be provided dynamically we achieve more that 95% compatibility with it :) someone may say that our looks more realistic.

2.1 Usage Scenarios

Well there are two main usage scenarios of the project one is like a standalone application and the other one is like a service in the DWARF platform and usually deployed on the car-simulator device that controls the visualization of the front panel. In standalone mode one could be able to manually load a SVG template, display it on the screen, resize it dynamically update the data values on all text elements, hide or show given elements by their Ids, and

modify the rpm values or the values for the oilmeter.
When deployed on the car-simulator the system should load the default template upon start, register it needs for events to the underlying DWARF system and update its view dynamically upon receiving events for the change in the state in some of the listened objects.
One other usage scenarios is creating such rich view 2D templates but it is something that is separate from the implementation and one can follow the available how-to in chapter five for further information. This last usage scenarios was not intended to be presented in the demonstration of the system although it was required to be created.

2.2 Functional Requirements

The following sections give a high-level overview of the functionality of the ComboView tool.

- The graphic should be shown in 2D using Java.
- The display component should update its view dynamically from the underlying environment.
- At least one of the created templates has to look similar to the fig 2.1
- A generic speedometer, rpm meter and oil meter components have to be created with the ability to change their values dynamically.
- The system has to be integrated in the car-simulator at the Department for Ergonomics
- The view has to be created using vector graphics nor rasterized.
- A concept for creating such components has to be created.

2.3 Non Functional Requirements

We know that the nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Here is a list of them that were relevant to this system.

- **Graphical User Interface** The user interface should be intuitive to use and provide a familiar “look and feel”.
- **Response Time** All tasks should be done maximum quickly to avoid nerving flickering of the screen during repaints.
- **Efficient** The system rendering algorithm has to be made efficient to update just the changed component and its children not the entire view and redundant events has to be removed from the queue not to force unwanted updates.
- **Extensibility** ComboView tool should be extensible, especially with respect to adding new components to the user interfaces and integrating more elements for communication with the driver. More extensions are proposed in chapter six.

2.4 Pseudo Requirements

Pseudo requirements are requirements imposed by the client that restrict the implementation of the system.

The DWARF Software Environment The visualization tool must compile and run in the current DWARF software environment.

Therefore it must be compatible with the following software:

- Linux 2.4
- GNU autotools (automake, autoconf)
- OmniORB

As hardware for the service the computer controlling the view of the front panel of the car-simulator has to be used which was P3 650 and had Windows 98 as an OS.

3 Design and Implementation

The purpose of this chapter is to give an overview over the structure of the application. It is mainly of interest for future developers who wish to extend and change the application. Here I will give general information about how to use the classes that the Comboview tool is made of. If you are interested in the exact signature of each method, you should have a look at the Javadoc that is available in appendix A. A full class and method documentation is extracted from the source in a nicely formatted and cross-referenced documentation which is available externally in HTML format. The chapter describes the classes of the three available subsystems. Classes that do not belong to a particular subsystem are described at the end of this chapter.

3.1 Design Goals

Reusable components: all components of project should work independently of each other, so they can be reused easily in future projects. This also results in a clean design with clear responsibilities.
Extensibility the Comboview should be extensible, especially with respect to the extensions proposed in chapter 6.
Compatibility with DWARF: the Comboview tool must work together with the middleware and existing DWARF services.
Performance: the visualization of the dynamically generated data has to be very quick almost a game like performance is required at least 20 Frames per second are needed for a smooth change between the states.

3.1 Class Design

The system consist of three components the first is responsibilities for the integration of the tool into the DWARF and is called DWARF System Integrator, the second one responsible for the visualization of the created Comboview templates is called Template Viewer and the last one but not for importance is the component responsible for the dynamic modification of the loaded templates is the Template Updater.

Dwarf System Integrator:

Consist of two classes they register the component as a service to the DWARF platform , start and stop the service upon request, keep log of the current executed operations and their status, dispatch the received events to the registered listeners.

The class ComboViewDataReceiver fig 3.1 is who is actually listening to the needed events from the DWARF system and is doing the dispatcher work. The class diagram can be seen beyond and the full javadoc at the end of the document in the Appendix A.

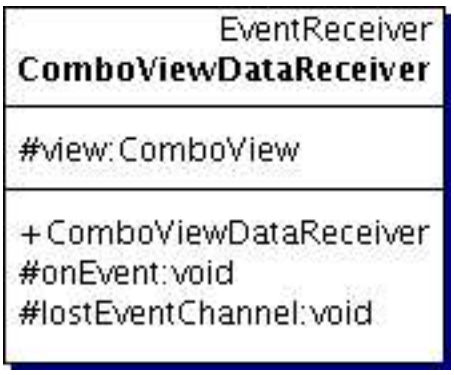


Figure 3.1: Class ComboViewDataReceiver

The class ComboViewService fig 3.2 is the start entry point of the service it is used to register the ComboView service to the DWARF service manager to initialize the other subsystems used for the visualization and updating the content, to register all the listeners for the needed events queues and to administer the execution of the service.

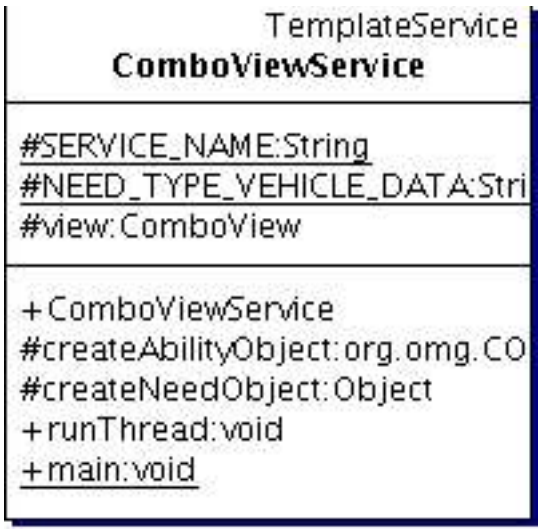


Figure 3.2: Class ComboViewService

Template Viewer:

This module is responsible for the loading and rendering of the SVG templates either from the filesystem or the classpath. It communicates directly with the Batik API for rendering and can be extended to handle specific input devices and controls. It handles everything that has to do with the UI visualization, configuration, calibrating of the display and relays on the update module to change its state dynamically.
The ComboView class fig 3.3 is relevant to the loading and rendering of the SVG templates it does not depend on any specific format of the file can be used as a generic SVG browser and may show either dynamic or static content, it may handle also mouse or keyboard

interaction with the users and support drag & drop. It depends actually on a JSVGCanvas instance to handle most of the jobs done. The actual work for loading and rendering a SVG template is summarized briefly here. After creating a JSVGCanvas instance the setURI() method is called with a URI of the SVG document to load. If the URI is valid the loadSVGDocument() method of the JSVGComponent class is called with the URI. In the loadSVGDocument() method all document processing for the current document is stopped, and then a DocumentLoader and SVGDocumentLoader are created. The SVGDocumentLoader handles the firing of document loading events and the DocumentLoader uses a SAXSVGDocumentFactory to create a SVGDocument instance. The SAXSVGDocumentFactory in turn relies on a SAXDocumentFactory to parse the SVG data referenced by the given URI. The JSVGCanvas relies on the SVGDocumentLoaderListener, GVTTreeBuilderListener, and SVGLoadEventDispatcherListener listeners that are added in the JSVGComponent constructor in order to know what is going on with the loading, building, and rendering of the SVGDocument. The listeners are implemented in the JGVTComponent.Listener and JSVGComponent.SVGListener classes. When the document has finished being loaded and parsed the setSVGDocument() method of the JSVGComponent class is called, which then calls the installSVGDocument() method in the same class. The installSVGDocument() method:

1. Gets rid of any resources from the previous document (if any)
2. Create a new BridgeContext to associate the SVG DOM and the GVT tree
3. Configures the component and BridgeContext instance based on the document state (ALWAYS_STATIC, ALWAYS_DYNAMIC, ALWAYS_INTERACTIVE, AUTODETECT)
4. Creates a GVTTreeBuilder with the SVGDocument and BridgeContext
5. Initializes event handling for mouse and keyboard events

When the GVTTreeBuilder is done building the GVT tree the JSVGComponent performs some additional configuration so that dynamic documents work correctly, and then calls the scheduleGVTRendering() method of the JGVTComponent class. The scheduleGVTRendering() method calls the renderGVTTree() method of the same class and:

1. Sets the size of the visible rect to be the size of the component
2. Creates either a static or dynamic image rendered based on the document state (StaticRenderer or DynamicRenderer)
3. Finds the inverse of the rendering transform, which is set to the identity matrix
4. Creates a shape by transforming the rectangle found in step 1 with the matrix from step 3
5. Creates a GVTTreeRenderer and starts the rendering thread

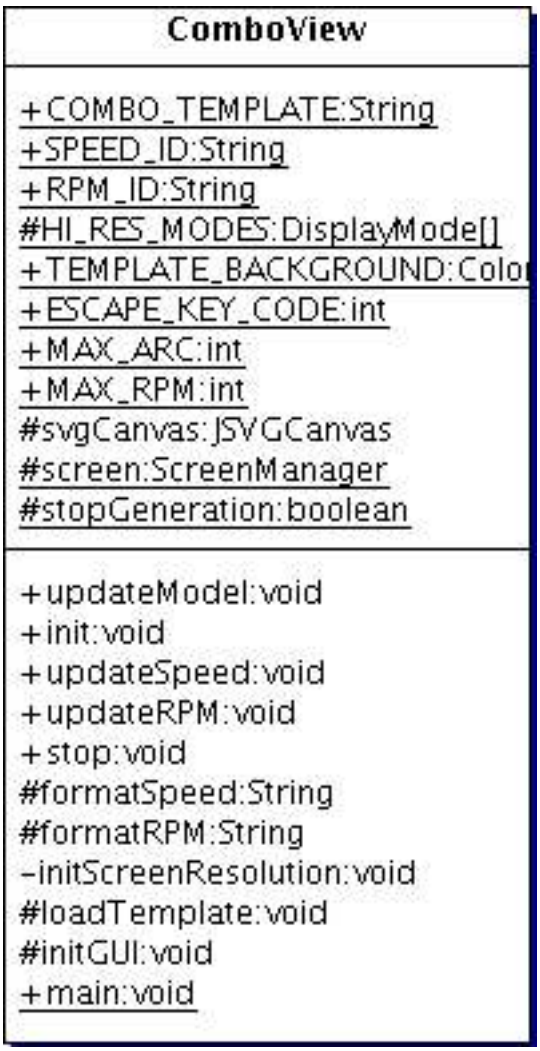


Figure 3.3: Class ComboView

SVGLoader fig 3.4 is the main entry point for the standalone application, it starts with a empty frame and a button Load which displays a File Selection Dialog and after a proper template is selected a Comboview element is created associated with the selected file and integrated into the empty frame and also a GUI that allows to change dynamically the loaded document is created and displayed for use to the user. This data modification dialog relies on the update module to apply the needed modification made by the user and to update the Comboview display appropriately. For more details please refer to the chapter four which describes the functionality of the application and provides screenshots and to the included Javadoc in Appendix A.
The last class from this module is the ScreenManager fig 3.5 which is responsible for the configuration, entering and exiting the fullscreen mode. It can be used either as a part of this module or standalone to allow a given JFrame to be displayed in fullscreen mode or normal mode.

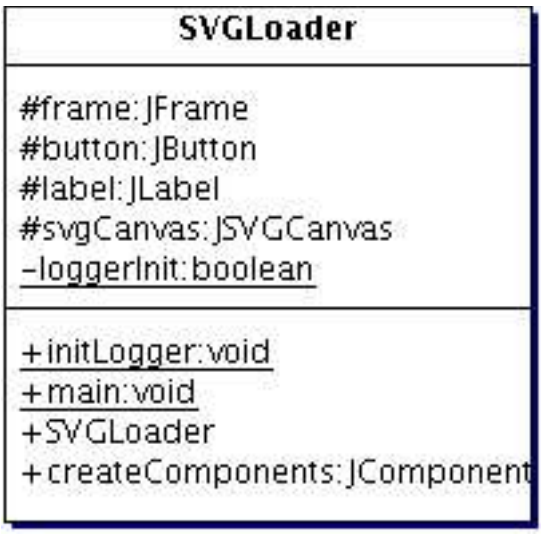


Figure 3.4: Class SVGLoader

Please note that this class is extensively tested on windows and some Linux platforms but the usage of it may bring some side effects especially if you are not using the latest jdks like 1.5+ that have support for OpenGL Java2D.

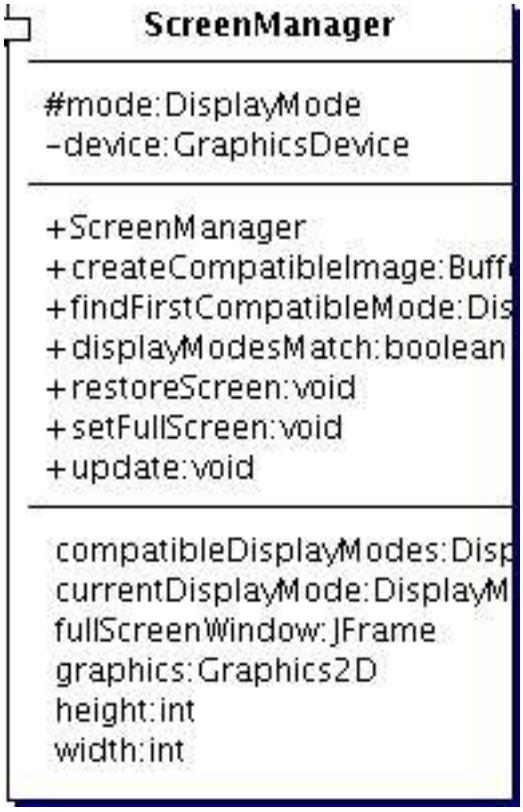


Figure 3.5: Class ScreenManager

Template Updater:

This module is responsible for the updates made on the loaded SVG DOM tree and its task are done by the two classed SVGDomUtil fig 3.6 and G2DUtil. The first one has built in methods for scheduling and update in the Batik rendering system, change attributes to elements given by their Ids like toggle visibility, change text, change position, change font attributes etc. It has convenient functions that can be called everywhere from the code and update any loaded SVG DOM model. Also exist support for the ComboView specific components like the oilmeter, rpm indicators, etc. It defines also some useful constants that are used in other parts of the system.

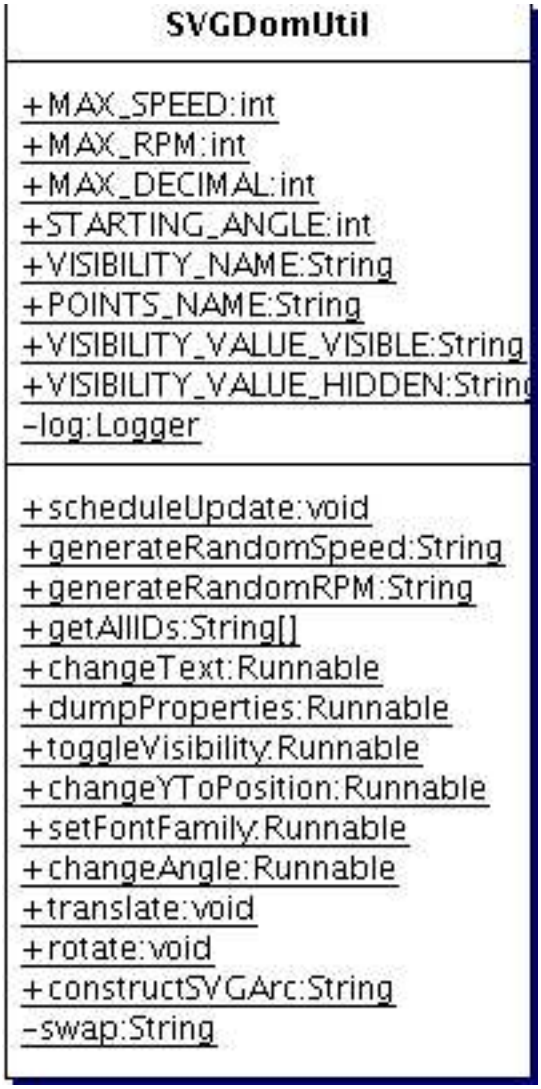


Figure 3.6: Class SVGDomUtil

The G2DUtil class is full of utility functions that help greatly when working directly with the

Java2D API. It has methods to return all drawable points of a line to enable disable double buffering thus increasing performance and allowing compatibility for the printing, to force update in all frames that are started from the current VM, to draw arrows etc.

3.2 Use Cases

This is the only use case available for the Comboview service fig 3.7 there exist similar and for the standalone application but as most parts of them are duplicated just this one will be discussed.

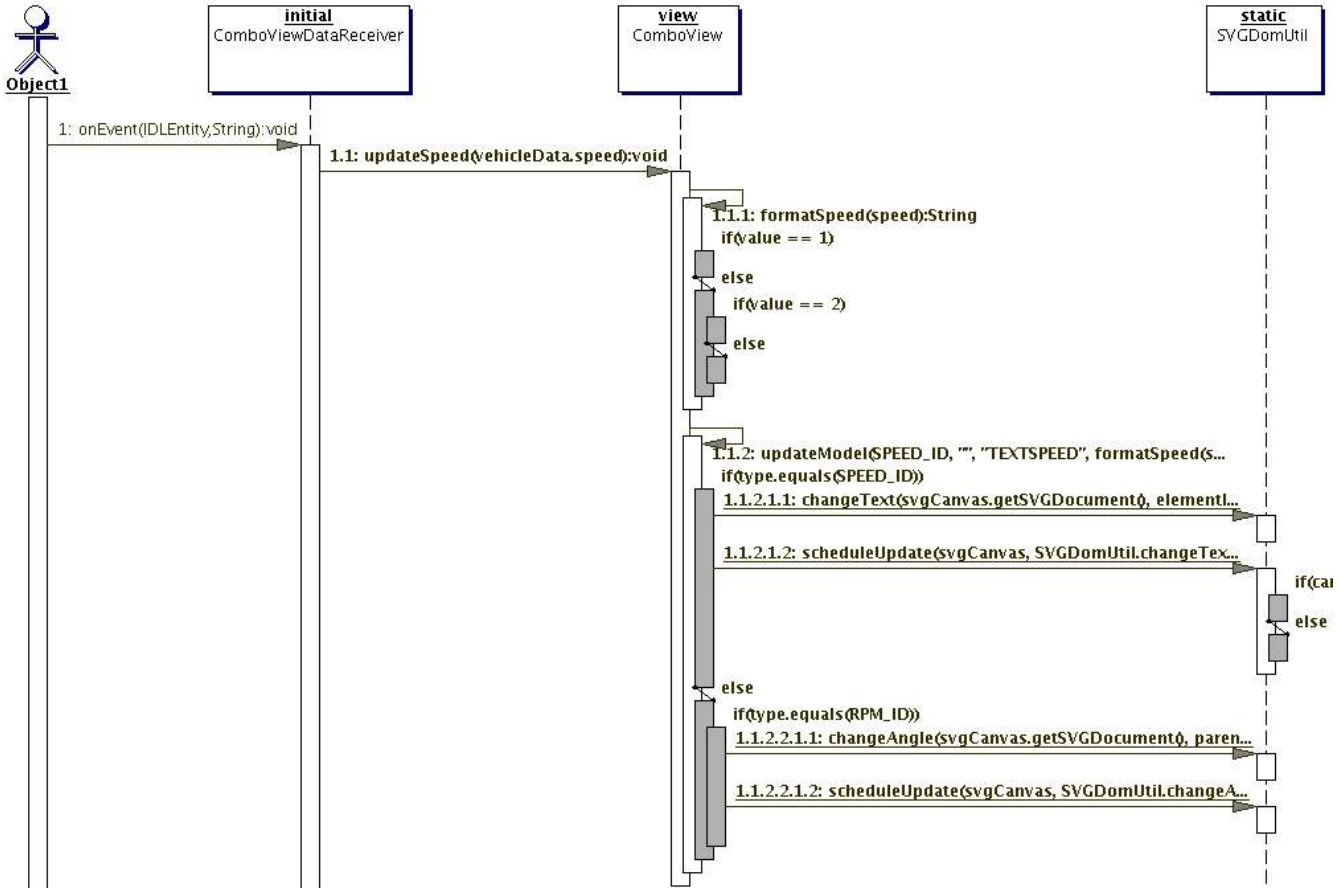


Figure 3.7: Car-simulator Use Case

After successfully registering to a listener for events of type VehicleData and receiving such one in the method onEvent() of the class ComboviewDataReceiver a decision is made upon the type of the event and if he is carrying speed data and rpm data. The appropriate update methods of the Comboview class are called which on their side dispatch them to the updateModule method in the class SVGDOMUtil which is actually making the real modification of the data and then is forcing an update in the Batik rendering.

4 Comboview Demo

This chapter gives on overview how the Comboview Service used in DWARF or the standalone template viewer are set up and what hardware was used to realize the different demos.

4.1 The Demo Setup

The developed project consist of two independent usage scenarios one is as a service integrated in the DWARF system and usually deployed directly on the computer that is controlling the video display of the car simulator and the second one is as an independent application that can be used to assist during the development of the different Comboview templates.
The standalone viewer can be used also as a simple SVG browser for any given source SVG file it has also builtin support for JavaScript and mouse events but no drag&drop or any interaction with the browsed content is allowed at this point. We will focus in this chapter at this two steps and discuss in details what are the requirements to successfully execute them.
For both applications is a requirement to have a Java Virtual Machine installed and configured in your path and it has to be at least version 1.4+ . As a hardware a PC with at least 300 Mhz and 64 Mb RAM is required or some degradations in rendering speed or too often swapping will appear.
As a free disk space for the standalone application at least 20 Mb free disk space are required and for the deployment of the DWARF and the Comboview Service at least 200 Mb are required. Both applications are tested and successfully deployed on a Windows XP/Linux (2.6 Kernel) although they should be able to run on any platform that fulfill their minimum requirements for the availability of the JRE and the ability to run the DWARF system on it so I can add to the list of supported but untested OS also Sun Solaris and some of the UNIX variants like (FreeBSD etc.).

4.2 The Standalone Application

4.2.1 Windows

For this OS are prepared specially startup files in the form of a binaries and they load the needed libraries at startup, search for available JREs and if not found shows an error dialog containing a link to the SUN website where one can be downloaded and installed separately. So if you look in the provided archive containing the binaries you will see the two files “ComboView.exe” that start in full screen mode the current default SVG Comboview template and change dynamically the speed and rpm values which are generated randomly from each other and “SVGLoader.exe” that start the so called standalone application and can be used to load various template change some of the properties of the elements and explore in realtime the result. It can modify for example the text value of given elements, toggle the visibility status, generate random speed or rpm values, modify the position of a child element according the number or parent elements change the angle of a given arc element which is actually used to display the rpm values. The next screen fig 4.1 shows all the available options that can be modified using this application the dialog is showed after a successful load of a SVG template from the application. If you would like to develop this application

further please note that only project files for Intellij IDEA are provided but all the needed sources, libraries and resources are included and it should be quite easy to create project for Eclipse, Netbeans or your favorite Java source code editor with support for compilation. Ant build file is also provided so you could easily modify the sources and compile without the need to install IDE or some custom build system. Please follow the “Readme.txt” file in the project directory for more information if you are interested in extending the current sources.

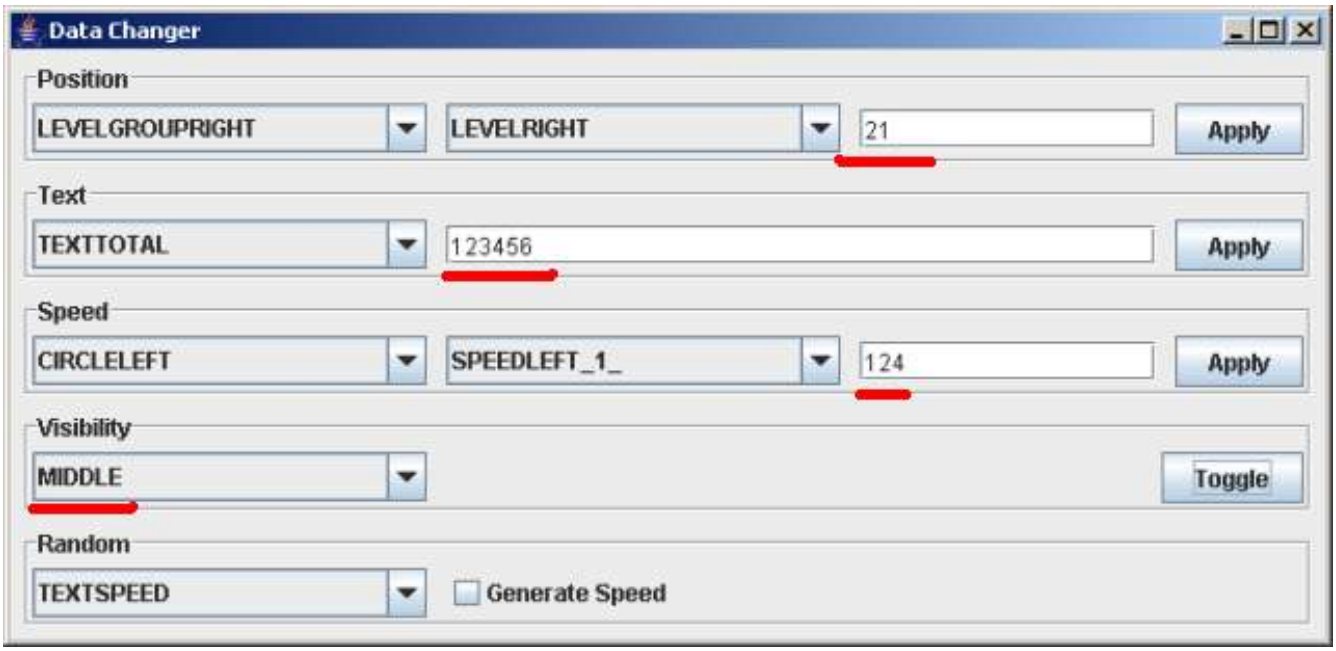


Figure 4.1: Data Changer Window WinXP

The next picture fig 4.2 is showing the changed template according to the provided options from the screen above. The load button allow you to load other templates and the properties screen above is recreated every time on the fly. If you like to open multiple templates at a given time you can easily just start more than one instance of the provided binary file.

The SVGComponent is very performant and can come to updates every 30 mil sec on a newer CPU 1.6+ normally it performs well on a CPU at 500 with updates every 100 mil sec which was one of the test machines. This view is resized dynamically at resize of the parent frame and cause of the nature of the vector graphics looks well even if it is scaled 10 or more time positive. Normally one can select text components with the mouse but moving of elements is not allowed for performant issues although it is supported by Batik. One can also disable this features thus increasing the performance greater but some side effects appear so tweaking that much the Batik platform is not encouraged.

Figure 4.2: SVGView Window WinXP

4.2.2 Linux

There are no startup file for the standalone application for Linux OS but one can easily create such one as all needed files are already provided. They should look something like

!#/bin/sh

<path_to_java_>/java -classpath <path_to_libraries> <full_main_class_name>

Both applications can be easily started from the Intellij IDEA as the project files are already provided and they contain all the needed configuration internally.
There is no difference between the behavior of the applications on Windows or Linux so the following screen-shots could be taken just as a proof that the application is working on Linux. Please note that on some Linux platforms there may occur problems with exiting from full screen mode which is discussed fully on the Java Sun Dev Forums but there is in most of the cases no cure it is better to just ignore the entering the full screen mode just comment the usage of the ScreenManager in the sources if a such problem occure. And the following picture fig 4.3 show the SVG Viewer Control on on Fedora Core 4 with Gnome 2.10 which view is modified according the data entered from the options dialog.

Figure 4.3: SVGView Window FC4

4.3 The Comboview Service

For successfully using the Comboview Service you have to have prior successfully installed or build the DWARF system. If you have precompiled binaries for you platform please skip the next two steps and go directly to Step three.

1. Get the sources from the CVS:

cvs -d :ext:<your_user_here>@cvsnavab.informatik.tu-muenchen.de:/cvs/dwarf co -d test_build -P .

2. Build the DWARF system

cd test_build
./bootstrap
cd build
../configure –prefix=/home/<your_user_here>/test_install --enable-tummicii
make all install

3. Start the DWARF platform

cd
cd test_install/bin

Start three shells and execute:

shell 1> ./run-servicemgr
shell 2> ./DIVE
shell 3> ./java -jar ApplicationStarter.jar -DserviceName=TUMMICII

After successfully doing these three steps and if the car simulator is up and running and generating appropriate events you should see on the deployment machine the following display fig 4.4 which is updated dynamically.

Figure 4.4: SVGView Car-simulator

5 How-to

In this chapter I will present two tutorials that will help you develop dwarf extensions and beautifully templates using Adobe Illustrator.

5.1 Create & Deploy Dwarf Extensions

As most of the information regarding development of extensions is already collected and well documented at the DWARF project web page and the DWARF Wiki server I have abandoned the creation of this how-to so please look for more information at [1, 7, 14].

5.2 Create SVG Templates with Adobe Illustrator

Till now we all know that the principle behind creating the SVG templates is quite simple we create the interface using some SVG capable graphics editor like Adobe Illustrator then we group the elements according to the desired behavior like speedometer, light bulb, left or right knob, oilmeter etc. We then set unique Ids for each group this can be done in the graphical editor or in an external XML editor if it is not supported by default like if you are using Inkscape not Adobe Illustrator.There are few things that I can summarize and you should keep in mind when creating any type of SVG templates.

5.2.1 How-to create templates

1. **Be Creative** : Do not think that if something does not exist or it does not have any logical structure it can not be simulated somehow. Almost 80 % of all shapes can be created by just using composition of simple ones like rectangle, circle, eclipse, arc. If you have problems simulating the shape take a real image import it into Illustrator and try to trace it there exist also some external tools that do this job for you.
2. **Logical Structure :** Almost think about the logical structure of the template do not just place your elements where you like cause later if you want quickly let say to hide a given component or a sub-component it will be almost impossible without a proper logical structure. Always choose a good human readable Ids for those elements cause it someone else is working on the same document it will be harder for him to understand the structure. So be cautious when creating the different layers. It is also a good practice to separate the dynamic from the static content. The following fig 5.1 is an example of a good logical structure that is following the prescribed rules.

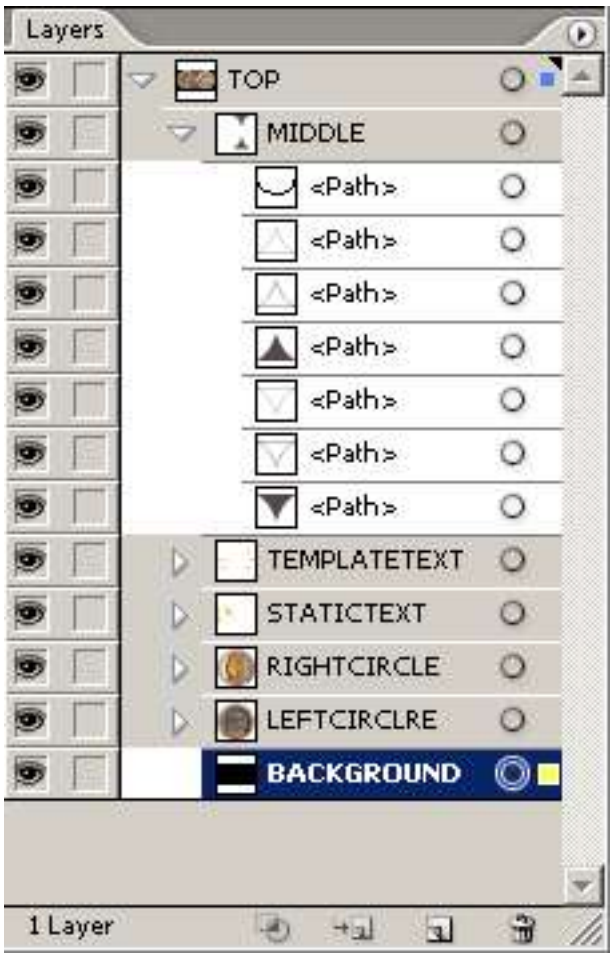


Figure 5.1: Illustrator Document View

3. Look for some existing samples. You do not need to find the wheel a lot of ready forms and shapes and tricks exists and are well documented just do a little googling or see some of the books for the illustrator or svg development.

Next I will show you which are the most used tools in Illustrator in what context can be used and how are created the generic components in the template and the special ones using them.

- **For Drawing simple lines and shapes** use some of the following tools:
  - Line Segment tool
  - Rectangle tool
  - Rounded Rectangle tool
  - Polygon tool

- Ellipse tool
- Star tool
- Arc tool
- Spiral tool
- Rectangular Grid tool
- Polar Grid tool
- They are all available at the tools window and are straightforward to use

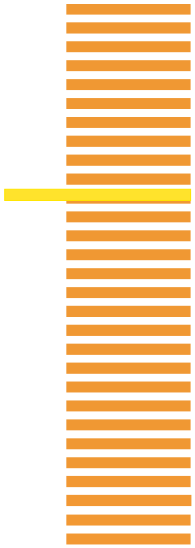
- **Drawing with the Pencil tool** You can do one of the following with the Pencil tool:
  - Draw freeform paths
  - Draw closed paths
  - Add to a path
  - Connect two paths
  - Reshape paths
- **Drawing with the Pen tool**
  - Draw Lines
  - Draw Curves
  - Draw Straight Lines followed by curves
  - or Draw the opposite
  - Draw two curved segments connected by a corner
- **Reshaping paths**
  - To reshape curves select the anchor point on either end of the curved segment you want to adjust
  - To stretch parts of a path select the Reshape tool
- **Adding, deleting, and converting anchor points**
  - Using the Pen tool to add and delete anchor points
  - To convert between smooth points and corner points select the Convert Anchor Point tool
- **Smoothing and simplifying paths**
  - To Smooth paths use the Smooth Tool
  - To Simplify paths use Object/Path/Simplify
- **Erasing, splitting, and joining paths**
  - To erase part of a path use the Erase Tool
  - To split a path use Scissors Tool
  - To join two end points use the Join Tool

- **Tracing artwork** This is very usefully for creating shapes from a raster graphics use the Object/Live Trace/Make. You may need to specify tracing color and the selection area to gain better results.
- **Symbols** A symbol is an art object that you can reuse in a document. For example, if you create a symbol from a flower, you can then add instances of that symbol multiple times to your artwork without actually adding the complex art multiple times. Symbols are grouped usually in symbols libraries and they save time and greatly reduce size of the document.
- **Drawing flares** The Flare tool creates flare objects with a bright center, a halo, and rays and rings. Use this tool to create an effect similar to a lens flare in a photograph.
- **Color/Fill/Strokes/Gradients** are usually the same as you have seen before in any other graphical editing tool so they wont be discussed in details.

Now lets see some concrete examples of creating the Comboview templates components:

- **Simulate Light On/Off** – this is the easiest task you have to change the color or the color intensity of the element in your case P is Off D is on one can make easily toggle buttons or long-short buttons where the pressing the button longer changes the intensity if the element. In such way are made all indicators like the one for the oil presence.

- **Create Dynamic Text Element –** just create a text element and give it an ID starting with “TEXT”
- **Add Bitmap Fonts/Glyphs –** you will want to integrate such bitmap fonts if you want your text to look the same on all platforms and you are using just a limited amount of symbols thus saving space. Please note that the TTF fonts look different on different OS and on different machines too and if lets say a given TTF font is missing it is not clear with which one it will be replaces. So If you want the symbols to look the same on all machines use bitmap fonts or import single glyphs into the template.
- **Speedometer –** the speedometer is created also easily it is actually a smaller circle with dark color and a bigger arc that changes dynamically only the current angle so the not needed part of the arc is cut from the circle that is staying above it. Just look at the two components with ids SPEEDRIGHT and CIRCLERIGHT.
- **Oilmeter –** the oilmeter is also easy to implement you have a rectangular element that is showing the current oil level and which position has to be changed according to the parent child elements which in our case are 30.You see in the picture later on the oil meter is the yellow marker nd the levels are given from the parent element 30 child elements that are brown.



- **Animation –** well there are some things that you cannot specify with Illustrator and that is one of them you have to either manually modify the result SVG files or use JavaScript or do some other custom SVG file transformation.

The next picture fig 5.2 shows a working screen from Illustrator the tools panel and the other panels can be placed where you like and it is a matter more of personal taste.

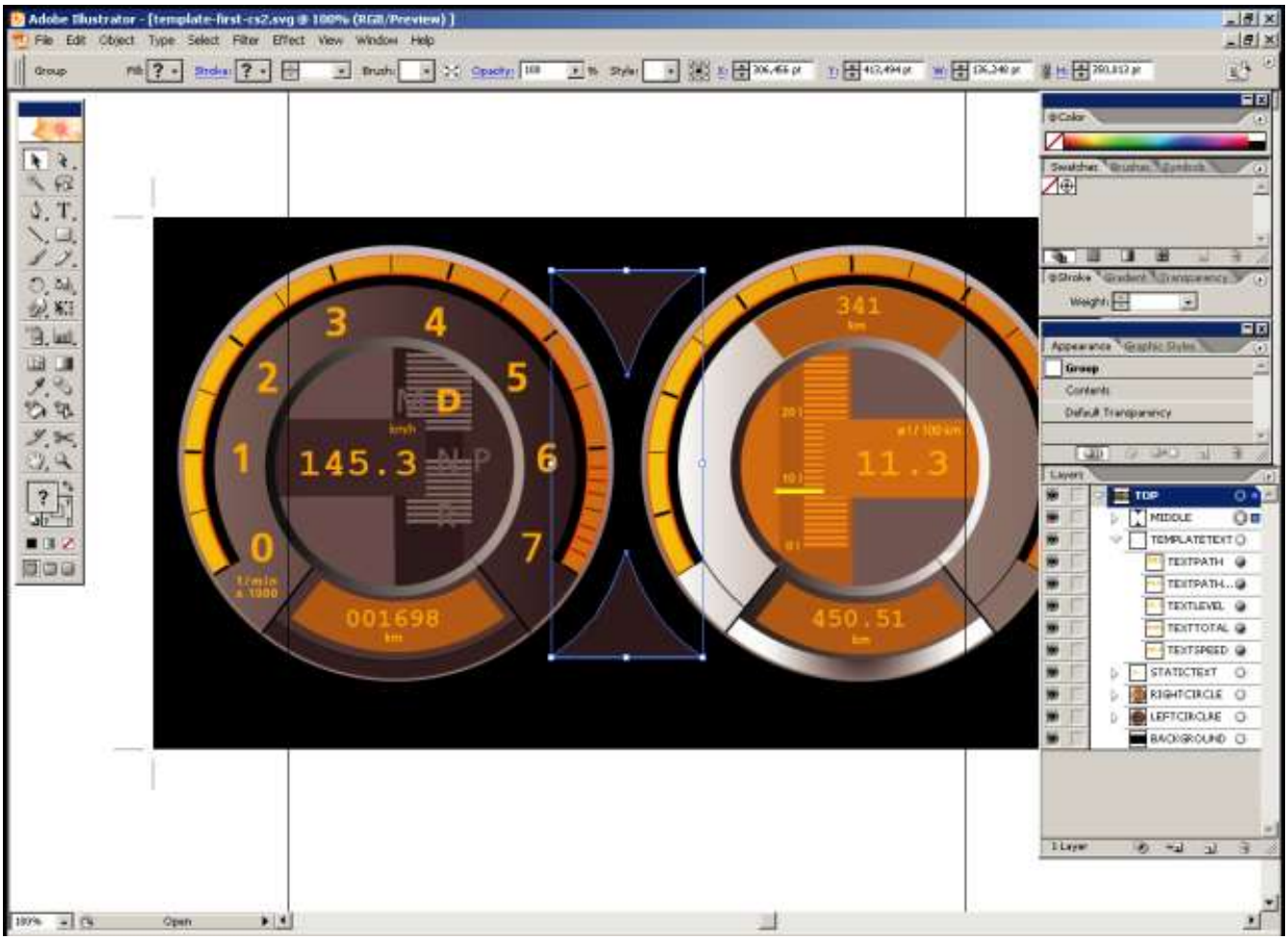


Figure 5.2: Illustrator in Action

The following picture fig 5.3 shows editing of the SVG template using Inkscape on Linux. It does support most of the basic features of Illustrator but does not allow to specify the elements id internally so you will have to use an external XML editor to do it. But for some basic editing is quite impressive and really quick. A windows port of the application is available also and it looks really promising.

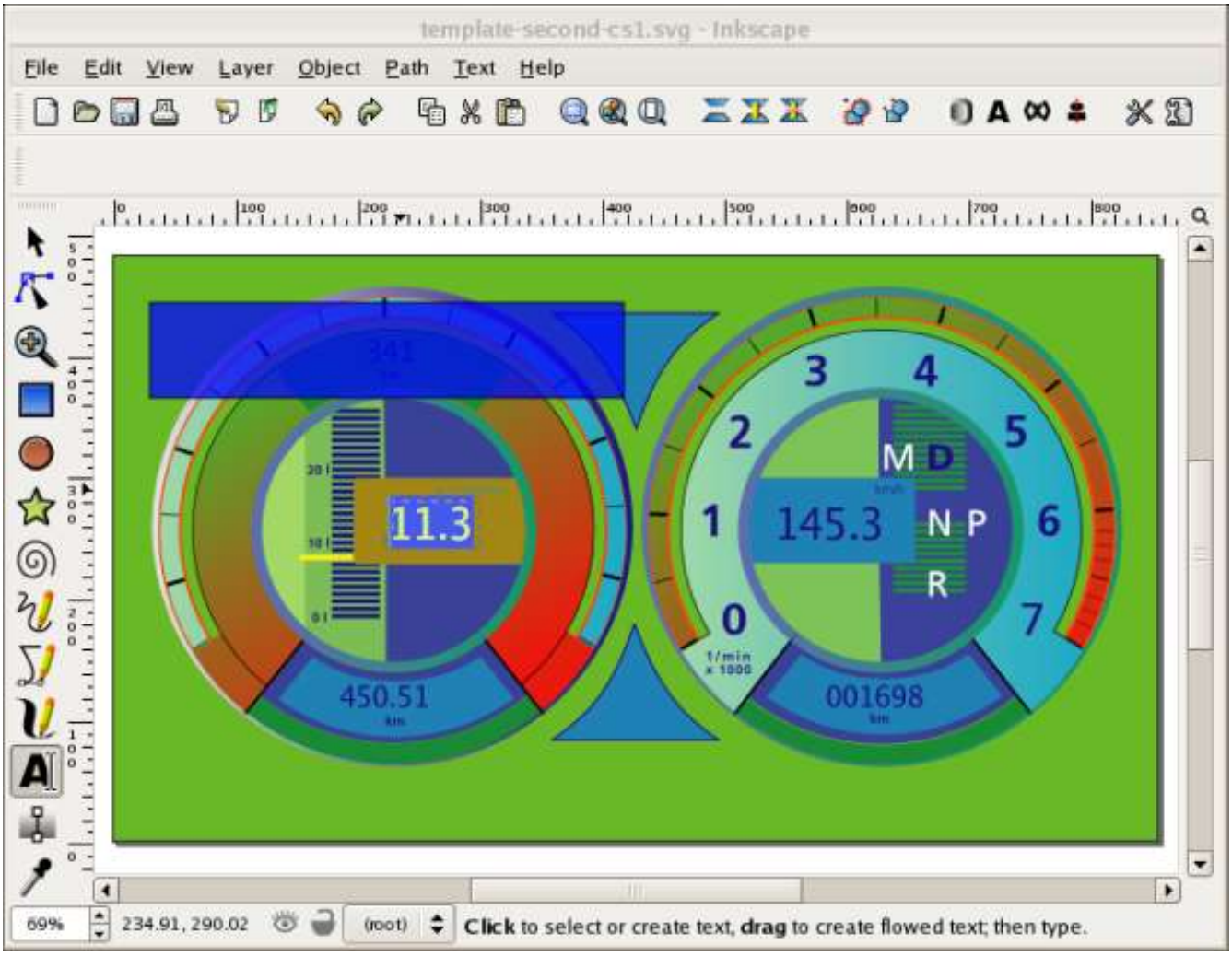


Figure 5.3: Inkscape In Action

You may use various tools when working with the SVG files the next two list contain almost all that are available and are actively developed.

5.2.2 Tools For Viewing SVG

- SVG Viewer released by Adobe
- **Batik SVG Toolkit**
- CSIRO SVG Toolkit
- X-Smiles XML browser

5.2.3 Tools For Creating SVG

- Designing with CorelDRAW
- **Designing with Adobe Illustrator**
- Designing with Jasc WebDraw
- Designing with Mayura Draw
- Designing with Virtual Mechanics' IMS Web Engine
- Designing with XML Spy
- Designing with SVG Studio
- Designing with GraPL
- Designing with Oak Draw
- Designing with IsoDraw
- Designing with SVGmaker
- Designing with Plazmic Workshop Start

Please note that when exporting SVG files with Adobe Illustrator the following options need to be taken into account.

5.2.4 Exporting to SVG

Adobe Illustrator 10/CS/CS2 allows to set SVG options when saving to this format. Some of those options are not supported by Apache Batik (the one bundled with Cocoon 2.0.2), and cause image generation to crash. Please save always using the following correct options.

Options Those options appear when save format is set to ".SVG".

Standard options :

Fonts > sub-settings :**None**

Images > location :**Embed**

Preserve Illustrator editing capabilities :**checked**

Advanced options :

CSS Properties :**Presentation Attributes**

Decimal Places :**3**

Encoding :**ISO 8859-1**

All other options :**unchecked**

6 Future Work

Well the project has fulfilled all its requirements but there are always things that could be improved and may be done either by me in a other phase of this project or by other person that continues my work. Well I will just try to summarize my wish list shortly

- **Drag & Drop support** it will be really cool one to be able to change the position of the created components with the mouse without the need to edit the template in the Illustrator.
- **Property editing** this is another useful extensions. I wish that a seamless properties editor will be made that allows to change the basic properties of the elements either on single selected or over a set of objects lets say to set common color, font etc.
- **Pure JS Implementation** of the updating subsystem which is now implemented in Java it will be really cool to integrate the Comboview Service in a browser using only the Adobe SVG Plugin or any other available. A partial implementation exists but the integration with the Corba system is missing and the speedometer component is not fully implemented.
- **Support Touchscreen devices** although Java supports generically touchscreen displays this will be really interested also from the driver point of view to test with the featured support for mouse event lets say to make the Comboview Display interactive not like now just to be updated from the events received from the car-simulator.
- More **generic support** for other type of templates why not make a whole bunch of usable components switches, buttons, markers etc.
- Support for **mobile devices** there is a specification for Mobile SVG but the rendering part has to be re-written from scratch in some of the rendering libraries that support Symbian OS like tiny Svg.

7 Conclusion

The previous chapters have given an overview of the functionality and the internal structure of the Comboview Service and Standalone application. The focus of Chapter 2 were the requirements for the visualization tool for SVG Templates. The functionality of project was described from the a user's perspective. This description included use cases and screen shots. It has fully described all the requirements of the system. Chapter 3 gave an overview of the system design, we saw how the project is divided into into three subsystem and how the main design goals – extensibility, reusability and performance - are achieved. The actual implementation was described also in this chapter. It gaves an overview over each class and showed how the ComboView handles some interesting technical tasks. How to setup a working demo or to successfully deploy and start the service is described with details in Chapter 4 it includes all the needed information and configuration parameters that can be usually set on the working environment. Chapter 5 is important for everyone that wants to deploy extensions for the DWARF system or to create SVG templates. A number of tools are described, also the way that the special Comboview components are created like the speedometer or the oilmeter .Chapter 6 focused on the future of Comboview Service. Despite of it's current usefulness, there still are many extensions which could make the project even more valuable. A number of additional features were proposed. The chapter also explained how these features can be integrated into the current design.

The current version of the Comboview Service and Standalone application works reasonably stable and has proven to be a valuable tool for the demonstration and improvement of the human-machine interface on the car systems. It is used currently in the car-simulator and has been successfully demonstrated on a number of occasions. More research has to be conducted on what tools and processes are necessary in order to allow an easy development of new SVG templates using other development tools and not the expensive Illustrator. One other interesting question is how exactly to handle input from the driver.

As we have seen in chapter 6, the Comboview service can be extended to partly achieve these goals, making it an even more useful tool for the exploring of the different issues in the human-machine interface.

8 Appendix

This chapter includes some useful information that is summarized and is aimed to help you understand the structure and the content of the document better. It contains a Javadoc Index that describe all public members and functions of the implemented classes and also a glossary of all used therms that are not straightforward to understand.

A : Javadoc

A

asPath(Point2D[]) - Static method in class de.comboview.util.G2DUtil
Creates a GeneralPath object from the current Point array.
asPoints(PathIterator) - Static method in class de.comboview.util.G2DUtil
Creates a Point array representation of the PathIterator object.

B

button - Variable in class de.comboview.ui.SVGLoader
The button that triggers the load

C

calculateArea(Polygon) - Static method in class de.comboview.util.G2DUtil
Calculates the area of a given Polygon.
changeAngle(SVGDocument, String, String, String) - Static method in class de.comboview.dom.SVGDomUtil
Creates a Runnable object that changes the angle of a given Arc Element of a given SVGDocument.
changeText(SVGDocument, String, String) - Static method in class de.comboview.dom.SVGDomUtil
Creates a Runnable object that changes the value of a given element from a given SVGDocument.
changeYToPosition(SVGDocument, String, String, String) - Static method in class de.comboview.dom.SVGDomUtil
Creates a Runnable object that changes the y possition attribute a given Element to some other Element from a given SVGDocument.
COMBO_TEMPLATE - Static variable in class de.comboview.ui.ComboView

ComboView - Class in de.comboview.ui
The Combo view class is used to display the combo panel in full screen mode and to update it upon receiving events from DWARF framework.
ComboView() - Constructor for class de.comboview.ui.ComboView

ComboViewDataReceiver - Class in de.comboview

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

view - Variable in class de.comboview.ComboViewDataReceiver

view - Variable in class de.comboview.ComboViewService
The combo viewer of the service
VISIBILITY_NAME - Static variable in class de.comboview.dom.SVGDomUtil

VISIBILITY_VALUE_HIDDEN - Static variable in class de.comboview.dom.SVGDomUtil

VISIBILITY_VALUE_VISIBLE - Static variable in class de.comboview.dom.SVGDomUtil

B: Glossary

Bitmap graphics Bitmap or raster formats treat each graphic as a collection of dots called bitmap, assigning a specific color to each pixel. When viewed as a whole, this collection makes up an image.

Filter effects “A filter effect consists of a series of graphics operations that are applied to a given source graphic to produce a modified graphical result.”(W3C, 2001)

SMIL “The Synchronized Multimedia Integration Language (SMIL, pronounced "smile") enables simple authoring of interactive audiovisual presentations. SMIL is typically used for "rich media"/multimedia presentations which integrate streaming audio and video with images, text or any other media type. SMIL is an easy-to-learn HTML-like language, and many SMIL presentations are written using a simple text-editor.” (W3C, 2002)

SVG SVG is the acronym for “Scalable Vector Graphics”, which is a XML- based vector graphic standard for Web graphics developed by the World Wide Web consortium (W3C).

Vector graphics Vector images are built up of many individual, scalable geometric objects defined by mathematical functions. Hence they consist of lines, curves, paths and other shapes. Therefore it is possible to render them at the highest feasible quality level.

W3C “The World Wide Web Consortium was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has more than 500 Member organizations from around the world and has earned international recognition for its contributions to the growth of the Web.” (W3C, 1999).

XHTML “XHTML 1.0 is the W3C's first Recommendation for XHTML, following on from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML 2.0. With a wealth of features, XHTML 1.0 is a reformulation of HTML 4.01 in XML, and combines the strength of HTML 4 with the power of XML.

XHTML 1.0 is the first major change to HTML since HTML 4.0 was released in 1997. It brings the rigor of XML to Web pages and is the keystone in W3C's work to create standards that provide richer Web pages on an ever increasing range of browser platforms including cell phones, televisions, cars, wallet sized wireless communicators, kiosks, and desktops.”

(W3C, 2002)

XML Stands for “Extensible Markup Language”. XML was also developed by the W3C in 1996. More information can be found at http://www.w3.org/XML/

XSL-FO With the help of XSL-FO stylesheets you can transform your XML document to any desired output format intended for printing like PostScript or PDF.

XSLT XSLT is a markup language for transforming XML documents into other XML documents, HTML or WML documents or any other imaginable formats.

DOM The DOM (Document Object Model) provides a unique identifier and access to all objects of a web page, including tables, images, graphical objects, etc. This is implemented in a hierarchical way so that a script can identify and modify each single object. The main issue of a clean DOM is that web−based applications usually use a lot of different formats, engines and plugins. However, communication often stops at applications, based on proprietary plugins.

Java2D This is not a graphics format but a very sophisticated graphics library for Java developers.

Transformations Each element and group may be transformed: translated, resized, rotated and skewed. Transformations may be nested (be careful to use the correct order!) or defined using transformation matrices. This way, it is possible to do all affine transformations

9 Bibliography

[1] Dwarf Project Homepage. Technische Universität München.
http://www.augmentedreality.de
[2] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riss, C. Sandor, and M. Wagner, Design of a Component–Based Augmented Reality Framework, in Proceedings of the International Symposium on Augmented Reality – ISAR 2001, New York, USA, 2001.
[3] Object Management Group, The Common Object Request Broker: Architecture and Specification.
http://www.omg.org/technology/documents/vault.htm#CORBA_IIOP
[4] Object Management Group, Corba Notification Specification.
http://www.omg.org/technology/documents/vault.htm#svc_and_fac
[5] B. Stroustrup, Die C++-Programmiersprache, Addison-Wesley-Longman, Bonn, 3rd ed., 1998.
[6] JUnit Homepage.
http://www.junit.org
[7] Log4j.
http://logging.apache.org/log4j/docs/
[8] K. BECK and E. GAMMA, Test Infected – Programmers Love Writing Tests.
http://members.pingnet.ch/gamma/junit.htm
[9] I.E. Sutherland: The Ultimate Display, invited lecture, IFIP Congress 65
[10] Bernd Brügge, Allen H Dutoit: Object-Oriented Software Engineering, Prentice Hall, New Jersey 2000
[11] W3C: Scalable Vector Graphics.
http://www.w3.org/Graphics/SVG/
[12] W3C: Web Accessibility Initiative.
http://www.w3.org/WAI/
[13] Apache XML Project: Batik.
http://xml.apache.org/batik/
[14] IBM Developer Works.
http://www.ibm.com/developerworks/web/library/x-svgint/
[15] SVG Developers Yahoo Group.
http://groups.yahoo.com/group/svg-developers/
[16] Document Object Model (DOM) Level 2 Core Specification. World Wide Web Consortium recommendation.
http://www.w3.org/TR/DOM-Level-2-Core
[17] SVG Test Suite.
http://www.w3.org/Graphics/SVG/Test/
[18] CSVG Home Page.
http://www.cs.washington.edu/homes/gjb/CSVG/
[19] Java 2D API.
http://java.sun.com/products/java-media/2D/index.html
[20] Adobe SVG Plugin
http://www.adobe.com/svg
[21] Adobe SVG Tutorial
http://www.adobe.com/svg/tutorial/intro.html

[22] Sun XML Developer Connection - Introduction to SVG
http://www.sun.com/software/xml/developers/svg
[23] Overview presentation of SVGs
http://www3/Consortium/Offices/Presentations/SVG
[24] X-Smiles – an open xml browser for exotic devices.
http://www.xsmiles.org/index.html
[25] Apache Cocoon.
http://cocoon.apache.org
[25] DWARF Project
http://wwwnavab.in.tum.de/Chair/ProjectDwarf
[26] David Kiley and Earle Eldridge, USA Today May 30,2002, "Car dashboards look more and more like jet cockpits; High-tech doodads tempt drivers to multi-task"
[27] S.J. Buckley and B.L. McClanahan, "Rear Speed Display: A Mnemonic Device to Improve Driver Situational Awareness", Proceedings, ISATA '97, June 1997.