ISSN 0280–5316
ISRN LUTFD2/TFRT--5573--SE

Java Based Picture Editing and Monitoring for Power Control Systems

Anders Henriksson

Department of Automatic Control
Lund Institute of Technology
December 1996

Department of Automatic Control
Lund Institute of Technology
Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 1996

Document Number
ISRN LUTFD2/TFRT--5573--SE

Author(s)
Anders Henriksson

Supervisor
Göran Ekstöm, ABB Network Partner AB
Karl-Erik Årzén

Sponsoring organisation

Title and subtitle
Java Based Picture Editing and Monitoring for Power Control Systems

Abstract

The Java programming language promises to be platform independent. This feature makes graphical user interface (GUI) development in Java interesting. The problem addressed in this paper is whether Java can be used as a core in a new man-machine interface for power control systems.

In order to get an answer to this problem the main objective of the thesis has been a evaluation of the Java-technology based on the building of a prototype editor/viewer application. The editor part should allow design of control pictures based on information supplied by a real-time control system.

The result is a fully functional editor/viewer application that can be downloaded from any WWW-browser, that supports Java. The prototype has been built using the OMT (Object Modeling Technique) method. The prototype supports multiple documents, i.e., many editor and viewer windows. There is a possibility to connect picture elements to a real-time database for information updating. Control pictures generate their own Java code, they can be zoomed and panned, and the design is prepared for information zooming. The development tool used, Symantec Café, as well as some Java performance related questions have also been investigated.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
69

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through:
University Library 2, Box 3, S-221 00 Lund, Sweden
Fax +46 46 222 44 22 E-mail ub2@uub2.lu.se

TABLE OF CONTENTS

1. Introduction _______ *6*

1.1 Motivation _______ **6**

1.2 Objectives _______ **7**

1.2.1 Properties of the editor/viewer application _______ 7

1.3 Scope _______ **9**

1.4 Outline _______ **10**

2. Background - S.P.I.D.E.R _______ *11*

2.1 The S.P.I.D.E.R. System _______ **11**

2.1.1 Man-Machine Communication _______ 11

2.1.2 Picture _______ 11

2.1.3 Picture Format _______ 11

2.1.4 Database _______ 12

3. The OMT Method _______ *13*

3.1 Overview _______ **13**

3.2 Analysis and Design _______ **14**

3.2.1 Analysis _______ 14

3.2.2 System Design _______ 14

3.2.3 Object Design _______ 14

3.3 The Development Process _______ **15**

3.4 The Three Models of the OMT method _______ **16**

3.4.1 Overview _______ 16

3.4.2 The object model _______ 16

3.4.3 The functional model _______ 16

3.4.4 The dynamic model _______ 16

3.4.5 Combining the Three Models - Object Design _______ 16

3.5 Frameworks _______ **18**

4. The Java Language _______ *19*

4.1 Overview _______ **19**

4.2 Introduction _______ **19**

4.2.1 The Java Buzzwords _______ 19

4.2.2 History _______ 19

4.3 Some Java Buzzwords Commented _______ **19**

4.3.1 Simple _______ 19

4.3.2 Network-Savvy _______ 19

4.3.3 Robust _______ 19

4.3.4 Secure _______ 20

4.3.5 Architecture Neutral _______ 20

4.3.6 Portable _______ 20

4.3.7 Interpreted _______ 20

4.3.8 High Performance _______ 20

4.3.9 Multithreaded _______ 20

4.3.10 Dynamic _______ 20

4.4 Applet vs. Application 21

4.5 Java Compile and Runtime Environments 22

4.6 Java API:s Available & Coming 23

5. Methodology 26

5.1 Literature and Other Sources of Knowledge Used 26
5.1.1 The Java Language 26
5.1.2 Design 26

5.2 The Prototype 26
5.2.1 The development process 26
5.2.2 Design 26

5.3 System Configuration 27

6. Design 28

6.1 Overview 28

6.2 The System 28
6.2.1 System Overview 28
6.2.2 Operating the System 28

6.3 Analysis 31
6.3.1 Object Model 31
6.3.2 Functional Model 35
6.3.3 Dynamic Model 35

6.4 System Design 43
6.4.1 Data Management 43
6.4.2 Data Managemnet Limitations 45
6.4.3 Network Communication Limitations 45

6.5 Object Design 45
6.5.1 Overview 45
6.5.2 The Heart of the Design 46
6.5.3 The Editor and the Viewer Window 49
6.5.4 Design Considerations, due to Applet Security Restrictions: Program Start. 50
6.5.5 The UI model 51
6.5.6 The Domain model 52
6.5.7 Updating of the ViewerWindow 52

7. Implementation 54

7.1 Overview 54

7.2 Extended AWT Functionality 54
7.2.1 The ToolButton Class 54
7.2.2 The PolyThickLine Class 54

7.3 AWT Specific Properties 54
7.3.1 To Catch Resize 54
7.3.2 The Updating of a Window 55
7.3.3 How to Implement Scollbars 55
7.3.4 Bugs to be Patched in the JDK 56

7.4 Points to be Highlighted in the Prototype 57
7.4.1 The ColorDataBase and the MessageDataBase 57

7.4.2 The Implementation of Callbacks 57
7.4.3 To Increase Performance of the Picture Drawing 57

8. Results ***58***

8.1 Overview **58**

8.2 The Prototype **58**
8.2.1 Description 58
8.2.2 Application Appearance 58
8.2.3 Applet Appearance 61
8.2.4 The Generation of Java Code and Parsing 62

8.3 Using Java For Development **63**

8.4 The Development Tool - Symantec Café **63**
8.4.1 Description Of The Symantec IDDE 64
8.4.2 Personal Reflections On Using Symantec Café 64

9. Conclusions ***66***

References and Bibliography

Appendix A

1. Introduction

1.1 Motivation

In most of today's control systems the picture presentation and the editing is often handled by separate programs , which must be installed on the target computer. Usually, the code has to be compiled on the same platform, as on which it executes. Pictures tend to get a different appearance in different environments. To overcome this problem it might be interesting to write a program on one platform and then by some mechanism it will automatically work on all other platforms. Sun Microsystems Incorporated defines its new Java language in the document "The Java Language: A White Paper" [1] as follows:

"Java: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language."

As the Java programming language promises to be platform independent, it would be interesting to investigate, whether Java can be used as a core in a new man-machine system.

Within the scope of the thesis there is a need to develop, evaluate and test different design principles and architectures for an editor/viewer application capable of creating and presenting picture objects, based on OOP design techniques.

The picture objects that are generated by the application should be portable. This means that the generated picture objects should be portable in the sense that they can easily be used by other Java applications than the editor/viewer application.

The editor/viewer application should also be adaptable to the new emerging Intranet technology, so that picture objects easily can be viewed from any computer within the cooperate network. This means that picture objects should be viewable from a Internet browser, i.e. Netscape. The Java language facilitates the development of such applications.

There is also a need to evaluate the Java programming language and its properties. Does the language live up to its promises? How does the language perform as compared to other languages, as the similar C++? It is also important to review the development tools available, as well as the frameworks and literature available for Java.

The question is what can you do in a limited time using Java, is it realistic to build a PED application based on Java? Can Java improve programming productivity? Can Java improve program maintenance?

1.2 Objectives

The main objective of the thesis is an evaluation of the Java-technology based on the building of a editor/viewer application.

The main objective contains a number of sub-objectives. The application should be designed using state-of-the-art object oriented design methods. The application should be developed with the best tools available. There should be a short evaluation and presentation of the Java language, the frameworks used as well as the development tools used, based on the authors personal reflections and available information.

1.2.1 Properties of the editor/viewer application

The editor/viewer application is intended to deal with generic picture elements and symbols that are common in electrical transmission nets. The elements and symbols can be dynamic to reflect a state (e.g. the state of a switch) or a value, or they can be static. When the application is operated in viewer mode, information about the controlled process should be presented in real-time. In edit mode the application should allow you to describe the process to be modeled as a picture object.

Thus, the editor/viewer application is subjected to a number of requirements. These requirements of course also set some limitations in the choice of application design.

Requirement list:

1. There are two modes: edit and view.
2. A picture object (PO) should be a collection of generic picture elements (PE) and symbols.
3. The PE or symbol can be dynamic or static.
4. The dynamic PE or symbol is to be updated on events, these events come from an information server, connected to the control system or a test program that is generating events.
5. The size and placement of a PE is expressed in world coordinates (absolute coordinates).
6. It should be possible to transfer the world coordinates to pixel coordinates.
7. A PO has a defined width and height in world coordinates.
8. The ratio between a PO:s width and height in world-coordinates should be preserved in pixel coordinates.
9. There can be associated dialog boxes to the PE:s or symbols.
10. A picture object should at least be viewable from an Internet browser.
11. There should be a possibility to zoom.
12. There should be a possibility to pan.
13. The extent of the world coordinate system should at least be X=Y=16284 units.

14. The Java code should be written so that the application can be extended to include information zoom with de-clutter levels (PE:s can then be associated to each de-clutter level).

15. The Java code should be written, so that the application can be extended to include layers. (A PE or symbol can only be associated to one layer.)

16. The editor application should generate the Java source code for the PO.

17. The design should promote the portability of a PO.

The building of a working Java prototype, which fulfills the requirements above, will constitute a foundation for a technology evaluation. The prototype should be built on the basis of the investigation and evaluation of design principles and architecture.

1.3 Scope

This essay will not try to explain the concept of object-orientation. For the reader interested in OO (Object Orientation), there exists a number of web-sites dealing extensively with the topic, one has the web-address *http://iamwww.unibe.ch/~scg/OOinfo/FAQ/oo-faq-S-11.10.0.27.html*, located at the university of Bern. However, a short presentation of the OMT-method (Object Modeling Technique) will be carried through.

1.4 Outline

The thesis is divided into chapters. Chapter 2 covers the background. Chapters 3-4 covers the problem context, i.e. the theoretical principles, which this work is based on. Chapter 3 mainly discusses the OMT method. The Java language is presented in Chapter 4. In Chapter 5 the methology used will be discussed. The motivation for the choice of design method as well as the choice of tools and framework will be discussed. In Chapter 6 the actual design of the editor/viewer application is discussed and illustrated. Chapter 7 discusses the Java-implementation of the application and sheds light on difficult or interesting parts of the implementation as well as some properties of the framework (this includes bugs). Chapter 8 evaluates the implemented editor/viewer application as well as the Java language, the development tools and the JDK framework. Conclusions are made in Chapter 9. Conclusions about the fulfillment of the set out objectives are made.

2. Background - S.P.I.D.E.R

2.1 The S.P.I.D.E.R. System

ABB has a system concept for an open systems architecture called S.P.I.D.E.R, which provides a platform for building network control applications, i.e. systems for Energy Management (EMS), Distribution Management (DMS) and Supervisory Control And Data Acquisition (SCADA).

2.1.1 Man-Machine Communication

The Man-Machine Communication (MMC) function provides common interactive support services for all S.P.I.D.E.R applications. The Graphical User Interface (GUI) used today is based on OSF/Motif.

2.1.2 Picture

The Man-Machine Interface includes a number of picture types for system operation and system maintenance, including graphic diagrams, tables, list's trend curves, tables and menus. Pictures can be defined, and linked to a picture hierarchy, to suit the specific requirements of individual applications.

Pictures can include static information and dynamic information. The dynamic information is updated based on information held in the control systems database according to Figure 2.1. Pictures can be designed with the aid of a application called Picture Editor (PED).

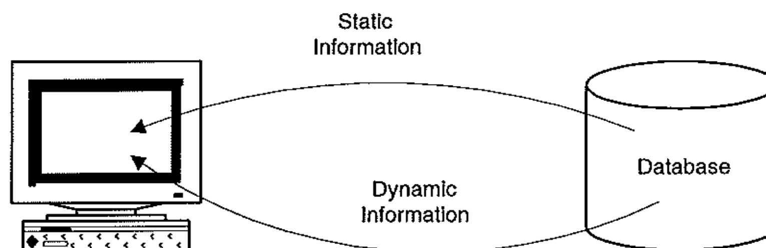


Figure 2.1, Information Pictures.

The PED also include support functions, such as: smooth pan, stepwise or variable zoom, and information zoom.

Dialogues are supported to manage windows, to select pictures, and to select data shown on pictures as well as power system operation, controlling breakers etc.

2.1.3 Picture Format

A picture compromises of a number of elements, some of which are static while others are dynamic and connected to a database and will alter their appearance according to the state in the database, see Figure 2.2.

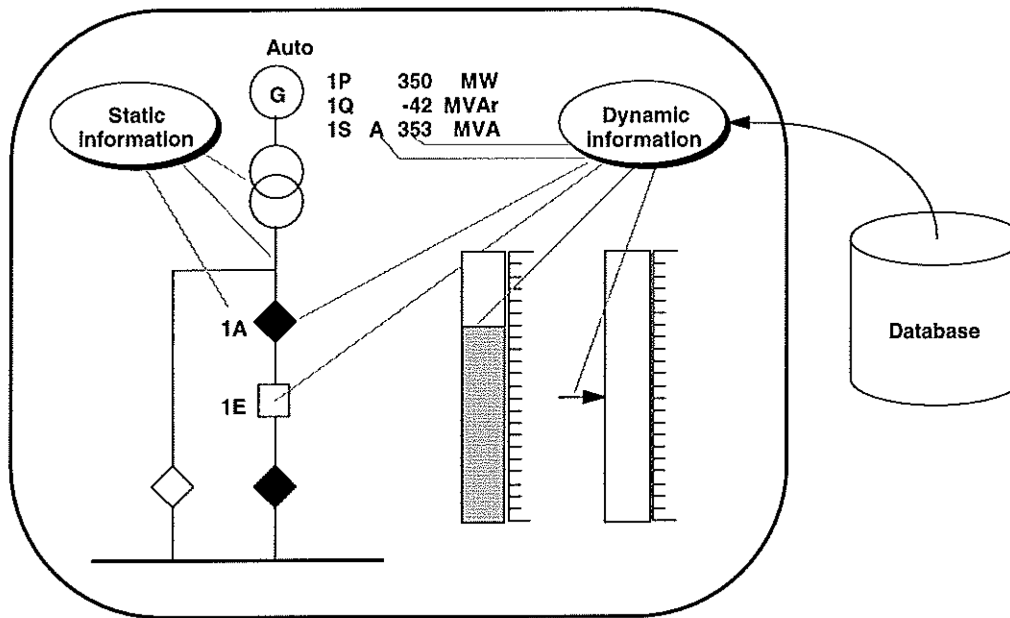


Figure 2.2, A Picture.

2.1.4 Database

The S.P.I.D.E.R system uses a database called Avanti[1] as a central repository of power system data. The internal structure of the database is designed specifically for the S.P.I.D.E.R applications and provides efficient real time data access for the power system monitoring, analysis and control functions.

The Avanti Query Language(AQL) is used to get information from an Avanti database. AQL is a real-time version of Structured Query Language (SQL).

---

[1] Avanti is a propriertary database managements system developed by ABB.

3. The OMT Method

3.1 Overview

This chapter assumes that the reader knows the concept of object orientation. The reader should be acquainted with terms such as classes, instances, methods, encapsulation, inheritance, and polymorphism. The chapter begins with a section dealing with analysis and design. Then follows a section dealing with the program development process. Then the three different models of the OMT (Object Modeling Technique) method are presented, with emphasis on the object model. In the last section the concept of frameworks is introduced. The aim is not to teach the reader the OMT method and its notations, but rather to explain the three models of the method and their interdependence. This means that no examples of the different models will be given, other than in the design.

There exists other object oriented analysis and design models than the OMT method, but OMT is the most wide-spread method.

3.2 Analysis and Design

3.2.1 Analysis

The concept of analysis deals with the question: What is to be done? While, on the other hand the concept of design deals with the question: How shall we do it? In the analysis phase, the description of the system must be free from technical and implementational considerations. The system´s information needs as well as its functions and structures must be described.

To aid the analysis process, models of the reality must be built. A model is an abstraction, which only considers system relevant information. One of the great advantages of object oriented analysis is that it tends to create systems, which model reality. The modeling activities can be described by Figure 3.1.

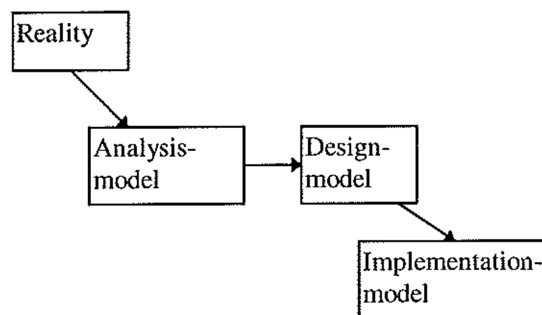


Figure 3.1, Modeling activities.

One of the advantages of the system being a model of reality, is that the same notion of the system prevails through all the stages of development: reality, analysis, design and implementation. This notion also forms the basis for iterative development, where prototypes with an increasing degree of refinement can be built.

3.2.2 System Design

If the analysis part looked at what the system should do , the system design part deals with the question: How should it be done? In the system design you deal with the whole system, such as division of the system into subsystems and processes.

3.2.3 Object Design

The object design deals with the design of the different classes, based on the analysis and the system design part. One could call the object design an integration of previous work.

3.3 The Development Process

There exists different approaches to the development process. The two main approaches are development according to the waterfall model and development according to some iterative model. Object Oriented analysis and design facilitates iterative development. The two methods are illustrated in the figures below.

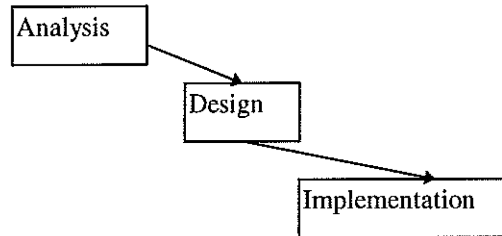


Figure 3.2, Development according to the waterfall model.

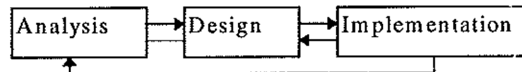


Figure 3.3, Development according to the iterative model.

The iterative model will allow stepwise refinements of the system. A prototype of the system could be implemented. If the prototype does not meet the expectations, then feedback is given to a new analysis of the system. These iterations can go on until a satisfactory system is implemented.

3.4 The Three Models of the OMT method

3.4.1 Overview

OMT comprises three models:

- The object model represents the static, structural, 'data' aspects of a system;
- The dynamic model represents the temporal, behavioral, 'control' aspects of a system;
- The functional model represents the transformational, 'functional' aspects of a system.

The three kinds of models separate a system into orthogonal views, they all describe one aspect of the system, but contain references to the other models.

3.4.2 The object model

The object model provides the essential data framework into which the dynamic and functional models can be placed, it defines the data structure that the other models operate on.

The result of object modeling is a object diagram. The object modeling activities are specially important during the analysis phase of the development.

3.4.3 The functional model

The functional model shows the computation and functional derivation of the data values in the system, without indication how (implementation), when (dynamic model), or why these values are computed. The relationship between values in a computation are showed in a data flow diagram.

3.4.4 The dynamic model

Those aspects of a system that are concerned with time and changes are captured in the dynamic model. It defines the control structure: the aspects of a system that describes the sequences of operations that occur in response to external stimuli, without consideration of what the operations do (functional model), what they operate on (object model), or how they are implemented. The major dynamic modeling concepts are events (external stimuli), and states (values and links of an object). The pattern of events, states, and state transitions for a given object class, can be represented as a state diagram. The dynamic model consists of multiple state diagrams, one for each object class with important dynamic behavior.

3.4.5 Combining the Three Models - Object Design

The last part of the design is the object design. In the analysis part and the system design part, work is done parallelly on the three different models. As mentioned before, in the Object design part, the work from previous stages of the process is integrated. Figure 3.4 explains the process of object design.

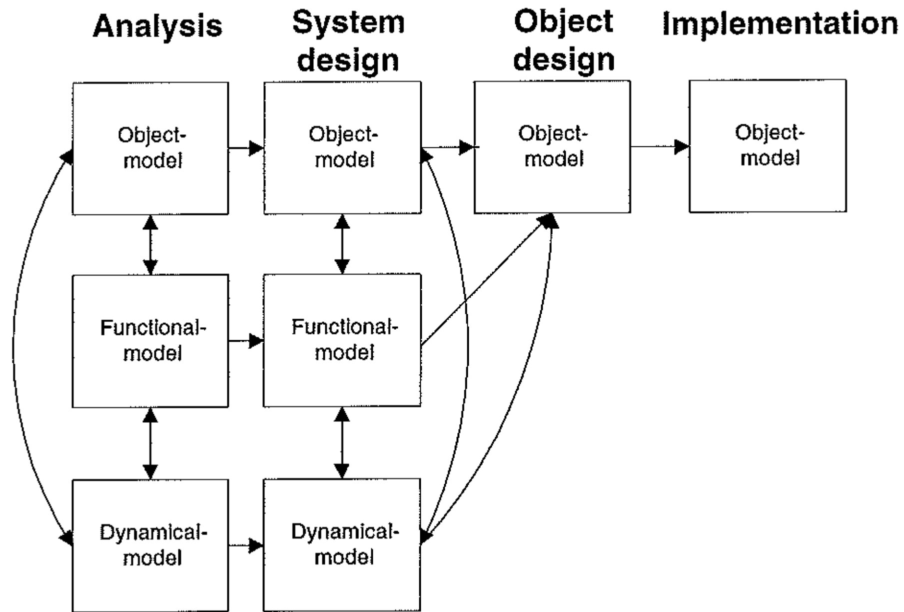


Figure 3.4, Combining the three models of the OMT-method(redrawn based on ENEA course material).

This integration process of the dynamic and functional models will add a number of operations to the object model.

3.5 Frameworks

A framework is a collection of cooperating classes that are put together as a reusable library. A framework is not a finished software product, rather it is a solid base to build applications upon. Frameworks focus mostly on a specific problem in software development. One framework may provide a base on which programmers build user interfaces, while another framework provides a basic structure for network operations. A framework for object-oriented programming can be customized for application specific purposes by subclassing the framework classes.

Most frameworks dictate a specific architecture to an application. Some frameworks even expect to be the only framework for an application. Sometimes there is a need to use two or more frameworks for a specific application. A framework can be inappropriate for solving a specific problem, if the dictated architecture is too restricting for an application. A close look into frameworks shows that they are full of patterns. These patterns are implemented as "ready to use" elements. A comparison of frameworks for similar environments (e.g. frameworks facilitating graphical user interfaces) shows that such frameworks mostly consist of similar patterns. The detailed implementation of a pattern, however, can differ from one implementation to another.

Frameworks are becoming more and more important. Especially large software projects use layers of frameworks. The code, written by using frameworks, is more "standardized" than independently implemented code. This is probably a result of that most application code is influenced by the frameworks it uses. Using frameworks to create applications, however, also requires that the programmers must be able to trust the frameworks they use. In other words, it is extremely important that framework implementations are robust. Nothing can be worse than changing a framework, when an application is about to get finished. Therefore, a careful evaluation of a framework for a specific problem can be one of the most important decisions within a software project.

4. The Java Language

4.1 Overview

In this chapter the Java language and its properties are discussed. The ability to run a Java program from inside an Internet browser as well as a standalone program is also discussed. Some aspects of the run-time environment are also covered. Finally available and coming API:s for the Java platform are presented. Most of the material presented herein builds on the Java whitepapers [W1].

4.2 Introduction

4.2.1 The Java Buzzwords

In the Java whitepaper [W2] the Java language is described with the buzzwords:

Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language.

Some of these buzzwords will be commented on in the sections below.

4.2.2 History

The Java language was developed at Sun Microsystems in 1991 as part of a research project to develop software for consumer electronics devices. The goal was to develop a small reliable, portable, distributed, real-time operating environment. When the project was started C++ was the language of choice. But there were several difficulties with using C++, so a new language called Java was developed.

4.3 Some Java Buzzwords Commented

4.3.1 Simple

The Java Language was designed to resemble C++. Some of the features of C++, that Sun saw as either poorly understood or confusing were removed in Java. These omitted features are primarily operator overloading (although method overloading is supported) and multiple inheritance. Automatic garbage collection is a feature that was added to Java, which simplifies memory management.

4.3.2 Network-Savvy

Java libraries have extensive routines to cope with TCP/IP protocols like HTTP and FTP. The goal is that it should be as simple to handle netaccess as handling access to the local filesystem.

4.3.3 Robust

Java is strongly typed and performs extensive compile-time checks, these checks are then repeated at link-time. Java also differs from C++, by having a pointer model that eliminates the possibility to overwrite memory and corrupt data. This is accomplished, by not using pointer arithmetic, but instead using so called true arrays.

4.3.4 Secure

Java is intended for use in networked/distributed environments. Sun says that Java will enable the construction of virus-free, tamper-free systems. This is supposed to be done by using authentication techniques based on public-key encryption. The use of the previously mentioned pointer model also adds to security.

4.3.5 Architecture Neutral

To allow the execution of a Java application anywhere on the network, the compiler generates an architecture neutral object file format. This fileformat can be executed on any platform (i.e. computer system and operating system) given the presence of the Java runtime system.

4.3.6 Portable

Architecture neutrality does not necessarily mean portability. Some computer languages (C and C++) contain aspects of their specifications, which can be implementation dependent. This is avoided in Java by specifying the sizes of primitive data types as well as the behavior of arithmetic on them (e.g. "int" always means a signed two's complement 32 bit integer etc.). The libraries that are part of the system also define portable interfaces. And the Java system itself is quite portable (based on POSIX) as the compiler is written in Java and the runtime system is written in ANSI C.

4.3.7 Interpreted

Java byte-codes are interpreted into native machine code instructions and not stored anywhere. Part of the byte-code contains some compile time information, that can be carried over to the runtime system, thus allowing type checks by the linker.

4.3.8 High Performance

To increase performance a JIT-compiler can be installed, which compiles the classes to native machine-code on the fly as they are dynamically loaded.

4.3.9 Multithreaded

Java support concurrency through threads as an integrated part of the language. Synchronization primitives are also supplied, these are based on the monitor condition variable paradigm introduced by C.A.R. Hoare [A1]. Multi-threading is a means of supplying real-time behavior. Unfortunately, this behavior is limited by the real-time responsiveness of the underlying system (e.g. Unix, Windows, Machintosh or Windows NT).

4.3.10 Dynamic

Java interconnects the different modules at runtime, this makes it possible to add new methods and instance variables to libraries, without having to recompile the whole system. Casts are also checked at run-time in Java.

4.4 Applet vs. Application

Java programs can run in two contexts - inside an Internet browser or as a standalone program. Programs written in Java that run inside HTML pages are called *applets*. They need a Java aware HTML browser, e.g. Netscape or Internet Explorer (or the Appletviewer) in order to run. Java programs can also run by themselves (these programs are called applications), like programs in C or C++.

Figure 4.1 shows a small standalone program.

```
class HelloWorld {
  public static void main( String args[] ) {
    System.out.println( "Hello world!" );
  }
}

Compiling the program:
C:\javac HelloWorld.java
Running the standalone program:
C:\java HelloWorld

Hello world!
```

Figure 4.1, A simple standalone Java program.

Creating applets is different from creating simple applications, because Java applets run and are displayed inside a Web browser with other page content. An example of a Java applet, together with html-code required is shown in Figure 4.2.

```
import java.awt.Graphics;

class HelloWorldApplet extends java.applet.Applet {
 public void paint( Graphics g ) {
                g.drawString( "HelloWorld !", 5, 25
);
        }
}
```

```
<HTML>
<HEAD>
<TITLE>Hello World Applet</TITLE>
</HEAD><BODY>
<P>Applet says:
<APPLET
CODE="HelloWorldApplet.class"
WIDTH=150
HEIGHT=25>
</BODY>
</HTML>
```

Figure 4.2, Java applet together with html page.

Usually a web-browser defines a certain security policy that limits what an applet can do. The most common restrictions are:

1. applets can never run any local executable program;

2. applets cannot communicate with any host other than the server from which they were downloaded;
3. applets cannot read or write to the local computer's filesystem (This is not part of the Java specification, but in Netscape things are done in this way and in the Microsoft Internet Explorer there is a user setting defining the security policy).
4. applets cannot find any information about the local computer, except for the Java version used; the name and version of the operating system; and the characters used to separate files, paths and lines.

The bytecode must be located in the same directory as that of the Web page or in any of its sub directories. When the Java aware browser encounters the <APPLET CODE> tag it transfers all of *.class files (i.e. the bytecode) to the clients side for execution. This can take a little time, depending on the bandwidth of the net, if there are many classes to transfer. Further version of Java will compress all class files into one major ZIP file before transferring and this will naturally decrease bandwidth utilization.

4.5 Java Compile and Runtime Environments

Figure 4.3 shows the workings of the Java language development environment, which includes run-time and compile-time environments. The Java Platform is represented by the run-time environment. A source file (*.java file) is written by the developer, this file then compiles into bytecodes (*.class file).

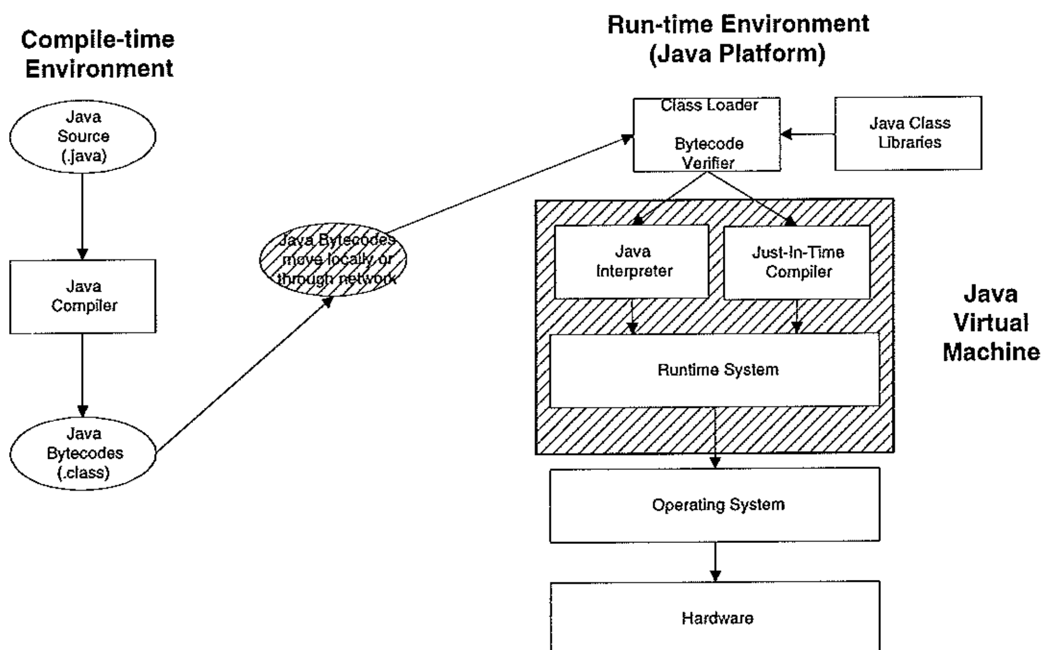


Figure 4.3, Source code is compiled to bytecodes, which are executed at runtime (from whitepaper The Java Platform p.24).

These bytecodes are instructions for the Java virtual machine. But before they can get into the run-time environment they must be transported over the network or loaded locally. Before the bytecodes are allowed to enter the Java virtual machine, they must

be checked for security reasons (it could be the case that they got compiled in a compiler that does not follow Java security rules).

The Java virtual machine interprets the bytecodes or compiles them to native machine code, with the assistance of the JIT-compiler (Just-In-Time). These operations are done in the context of the run-time system (threads, memory and other system resources).

Figure 4.3 also shows how the run-time system is implemented on different platforms. The Java Virtual machine is a program that can interpret or compile Java bytecodes. The virtual machine itself is implemented on top of the hardware and operating system of the platform.

4.6 Java API:s Available & Coming

This is a short presentation of available and planned API:s for Java. For the work, carried through in this thesis the JDK 1.0.2. is utilized (no other API:s were available at the time that the work was done). The information below is a short version of the information on the web-page http://www.javasoft.com/products/apiOverview.html. At the end of this section some words are also mentioned about the Java Process Automation API, where ABB Systems Control is involved as an industrial partner to Sun.

- The JDK 1.0.2 API is the API that ships with version 1.0.2 of the Java Development Kit (JDK). It includes the 8 basic packages: java.lang, java.io, java.util, java.net, java.awt, java.awt.image, java.awt.peer, and java.applet.
- The JDK 1.1 API is the next major release of the Java Development Kit. It is a superset of JDK 1.0.2 that will bring improvements in functionality, performance and quality to Java.
- Java Enterprise APIs support connectivity to enterprise databases and legacy applications. With these APIs, corporate developers are building distributed client/server applets and applications in Java that run on any OS or hardware platform in the enterprise. Java Enterprise currently encompasses three areas: JDBC™, Java IDL, and Java RMI. JDBC™ is Java Database Connectivity, a standard SQL database access interface, providing uniform access to a wide range of relational databases. Java IDL is developed in accordance to the OMG Interface Definition Language specification, as a language-neutral way to specify an interface between an object and its client on a different platform. Java RMI is remote method invocation between peers, or between client and server, when applications at both ends of the invocation are written in Java.
- Java Server API is an extensible framework that enables and eases the development of a whole spectrum of Java-powered Internet and intranet servers. The APIs provide uniform and consistent access to the server and administrative system resources required for developers to quickly develop their own Java 'servlets' - executable programs that users upload to run on networks or servers.
- The Java Security APIs are frameworks for developers to easily and securely include security functionality in their applets and applications. This functionality includes cryptography with digital signatures, encryption, and authentication.

- The Java Beans APIs define a portable, platform-neutral set of APIs for software components. Java Bean components will be able to plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's LiveConnect. End users will be able to compose together Java Beans components using application builders. For example, a button component could trigger a bar chart to be drawn in another component, or a live data feed component could be represented as a chart in another component. (Java Beans is currently an internal code name.)

- Java Commerce API will bring secure purchasing and financial management to the Web. JavaWallet is the initial component, which defines and implements a client-side framework for credit card, debit card, and electronic cash transactions.

- Java Management API provides a rich set of extensible Java objects and methods for building applets that can manage an enterprise network over Internets. It has been developed in collaboration with SunSoft and a broad range of industry leaders.

- Java Media APIs allow developers and users to easily and flexibly take advantage of a wide range of rich, interactive media on the Web. The Media APIs encompass five areas: Java 2D, Java MediaFramework, Java Share, Java Animation, Java Telephony and Java 3D. Java 2D provides an abstract imaging model that extends the 1.0.2 AWT package, including line art, images, color, transforms and compositing. Java Media Framework has clocks for synchronizing, and media players for playing audio, video and MIDI. Clock and media players are on different schedules. Java Share provides for sharing of applications among multiple users, such as a shared white board. Java Animation provides for motion and transformations of 2D objects. It makes use of the Java Media Framework for synchronization, composition and timing. Java Telephony integrates telephones with computers. It provides basic functionality for a full range of telephone services including simple phone calls, teleconferencing, call transfer, caller ID and DTMF decode/encode. Java 3D provides an abstract, interactive imaging model for behavior and control of 3D objects.

- The Java Embedded APIs specify how the Java API may be subsetted for embedded devices that are incapable of supporting the full Java Core API. It includes a minimal embedded API based on java.lang, java.util and parts of java.io. It then defines a series of extensions for particular areas such as networking and GUIs.

A Java Chip family of microprocessors will also be realeased. These microprocessors will support the Java Virtual Machine and be designed to support the demands of Java, such as multithreading and garbage collection.

A Java OS will also be developed and implement the Java Base Platform for running Java-powered applets and applications. As such, it implements the Java Virtual Machine, Java Embedded API, and the underlying functionality for windowing, networking and file system. The operating system is designed for Network Computers, consumer devices, and network devices for embedded applications, such as printers, copiers and industrial controllers. These devices will have instant turn-on,

no installation setup, no system administration, and, when on a network, can be automatically upgraded. The Java OS can also run in RAM on JavaChip.

At ISA/96 in Chicago (Oct. 7), Sun announced the development of a Process Automation API in cooperation with industrial partners, such as ABB Systems Control, The Baan Company, SAP, Valmet Automation among others. The API is supposed to be ready in early Q2, 1997. The API will allow the development of real-time process control applications, that run over the internet or an intranet (More information is available at http://www.industry.net/isa96/sunapi.htm).

Today the JDK 1.0.2 API is defined as the Java Base API, and some of the above mentioned API:s will in the future migrate into this API (The Java 2D API is expected to do so, and that could be interesting for the prototype that is to be built in this thesis).

5. Methodology

5.1 Literature and Other Sources of Knowledge Used

The knowledge and competence to carry this project through has been gathered through several sources. My supervisor Mr. Göran Ekström, senior specialist software design at ABB Network Partner, has been a great support, with whom I have been able to discuss matters of design and implementation of the prototype. Other sources of information have been books on the subject as well as the Internet, which has the most recent information as well as a number of Java sites with FAQ:s (Frequently Asked Questions).

5.1.1 The Java Language

Much of the documentation as well as information and news on Java is available at Javasofts homepage [W4]. A good book on Java with many example applets is Core Java [2].

5.1.2 Design

The material on the OMT method comes mainly from a book called Object Oriented Modeling and Design [3]. Useful material on the OMT method can also be found in course material from ENEA AB [4].

5.2 The Prototype

5.2.1 The development process

Some literature suggests that the number of iterations that the development process has to go through, decreases with how much knowledge there initially exists about the problem. As the problem of this thesis is to evaluate the Java language through the implementation of a prototype, the initial knowledge of the problem is low, and thus one could suspect that the number of iterations may become many.

Some times it has been necessary to implement some code in Java to see if things can be done in that way and then return to the design of the OMT-method. This may seem awkward as the object-model in the analysis stage of the OMT-method, is supposed to be free from implementational considerations. This might be true ideally, but the pratical experience of this work, is that a number of iterations stretching all the way to the implementation stage of the development process have been necessary in order to construct a wise design.

5.2.2 Design

The prototype to be constructed is supposed to handle a number of primitive drawing elements (like lines, rectangles, circles), so it would be wise to find some kind of framework to handle such elements (Frameworks of this type, are normally available on other platforms, like Windows).

The framework supplied by the JDK facilitated through the awt-package (Abstract Window Toolkit) does not give sufficient support to handle primitive drawing

elements. So, there existed two alternatives: to develop such a Framework by myself (which would involve extensive work) or get it from another source. The web was searched for companies supplying such Frameworks (like Rouge Wawe). But, the solution was supplied by a colleague of mine at the department Mr. Christer Carlsson who was working on his bachelors' thesis. Some student colleagues of his had done a Java-project last semester, which involved developing such a framework as well as a primitive drawing application with basic functionality showing what the framework was capable of. The Framework called HiJC (Hilevel Java Canvas), was the solution to my problems and the drawing application supplied with it constituted a good ground to build some of the UI parts of the prototype on, naturally there was a need to extend these classes with much functionality. This framework, as well as its documentation is available on the web (cf. http://www-und.ida.liu.se/~pum17 [W3]).

For the design of the application, the OMT-method has been utilized. The OMT-method has not been used exhaustively, but used were applicable and has also been completed by scenarios to describe the functionality of the prototype.

5.3 System Configuration

The development has been done on a 100 Mhz Pentium PC, with 32 MB RAM and the Windows NT 3.51 Operating system. Netscape 3.0 Gold has been used as Internet browser in the project, so as to have a benchmark browser, as the behavior of the Java applet tends to differ between browsers. Symantec Café 1.50 beta_1 has been used as development tool. Symantec Café is a IDDE(Integrated Development and Debugging Environment). There are several IDDE:s on the market, but the Symantec product was chosen, because the ABB Department were the work was carried through had some prior experience with it.

6. Design

6.1 Overview

In this chapter the actual design of the prototype is presented. The OMT method has been used, but not to full extent. Where other techniques have been utilized, it is stated in the text. The chapter begins with a section describing the overall system followed by a section describing the analysis stage and the three different models of the OMT method. Then, the system design and object design stages of the prototype design are presented.

6.2 The System

6.2.1 System Overview

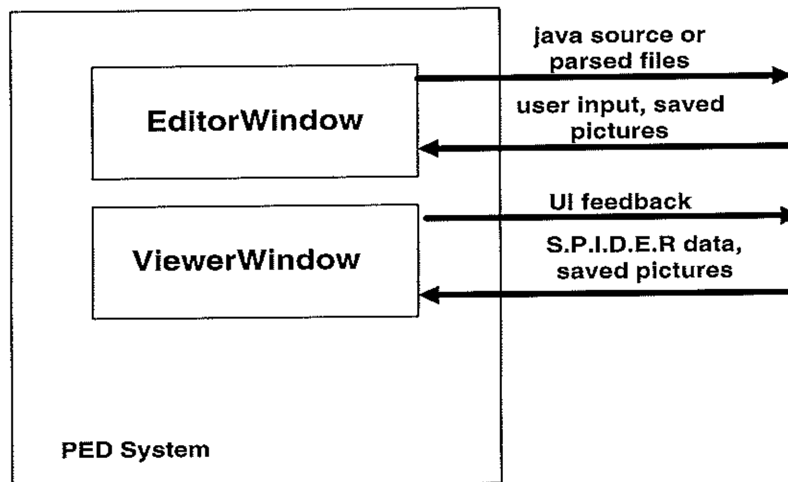


Figure 6.1, Overview of the System.

In Figure 6.1, an overview of the system is shown. There is support for multiple editor and viewer windows. The system inputs and outputs are also shown.

In the editor window the user can design his/her own control pictures. The user can also edit pictures that have already been saved. The data output from the editor window is a picture represented as java source code or a picture represented as a parsed file. This parsed file contains basic textual information describing the shapes constituting the picture. The java source code file then becomes compiled into a *.class file and then there is no distinction between the picture and the applet, i.e. the picture is part of the applet.

The viewer window allows the user to monitor the control pictures. The viewer window gets datainput from the S.P.I.D.E.R system. Based on this data input the state of the components in the UI changes, e.g. a switch can be turned on or off.

6.2.2 Operating the System

The system can be operated in either of two modes, as an applet in the context of a WWW browser or as a normal application that is started from the command line.

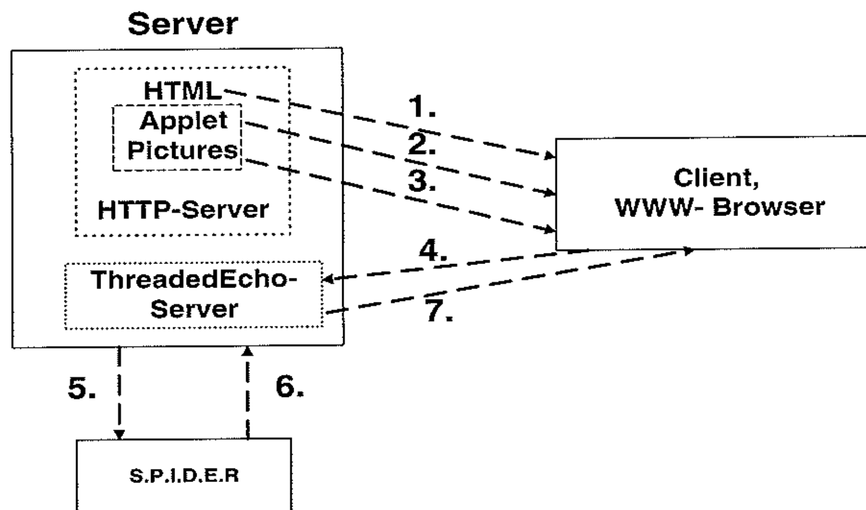


Figure 6.2, Running the system as an applet.

If the system is installed on a HTTP-server it will operate as an applet, which is the case in Figure 6.2. On the client side there must be a Java aware WWW-browser installed in order to run the system. The first thing the user on the client side must perform is to access the HTTP-server with its web address. The first thing that happens is that the web page is transferred to the client side (arrow 1), when the <APPLET> tag is found, the applet will be transferred from the server to the client side (arrow 2). If the user chooses to download a picture the picture-class is transferred over the net (arrow 3). Because pictures are compiled java classes there is no distinction between pictures and the actual applet. The pictures can also be parsed and then they are just datafiles and cannot be regarded as a part of the applet. In the current version of the prototype a picture can only be parsed when running in application mode (for this to work in applet mode requires extended server functionality). If the user has opened a viewer window, contact with the S.P.I.D.E.R system must be established in order for the monitoring to work. Due to applet security restriction the applet may only contact the server it was downloaded from. Often the HTTP-server and the S.P.I.D.E.R server are on different hosts. Therefore a Proxy server needs to be installed on the same host as the HTTP server. This proxy server establishes contact with the S.P.I.D.E.R system (arrows 5 and 6) and echoes bi-directional information between the applet and the S.P.I.D.E.R system (arrows 4 and 7).

If the program is operated in application mode it can be started from the command line and there is no need for a proxy server as the application is not subjected to applet security restrictions and can contact with the S.P.I.D.E.R system directly.

Figure 6.3 shows how an application is connected to the S.P.I.D.E.R system. In this mode files can be freely parsed and compiled and access to the local filesystem is granted.

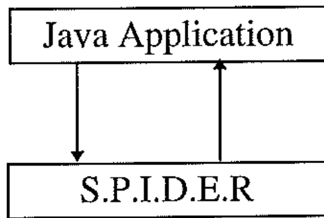


Figure 6.3, Running in application mode.

6.3 Analysis

6.3.1 Object Model

In Figure 6.4, an overview of the object model in the analysis stage of the prototype is shown (The OMT-notation used in preceding chapters is explained in appendix A).

This object model is the result of an iterative process involving all the stages of the development process. The goal has been to fulfill the requirements list of the objectives section in Chapter 1 to as large extent as possible (When referring to a requirement, the notation *Req. x* is used, where x is the requirement number). The abstract window toolkit (awt) framework as well as the framework supplied by the HiJC package have also influenced the model. This is the nature of frameworks to control the design, by putting limitations on the design possibilities through the structure supplied by the framework. This need not be negative. If the framework is soundly designed it can contribute to a good design. Below follows a more detailed description of the design together with motivations for the different choices.

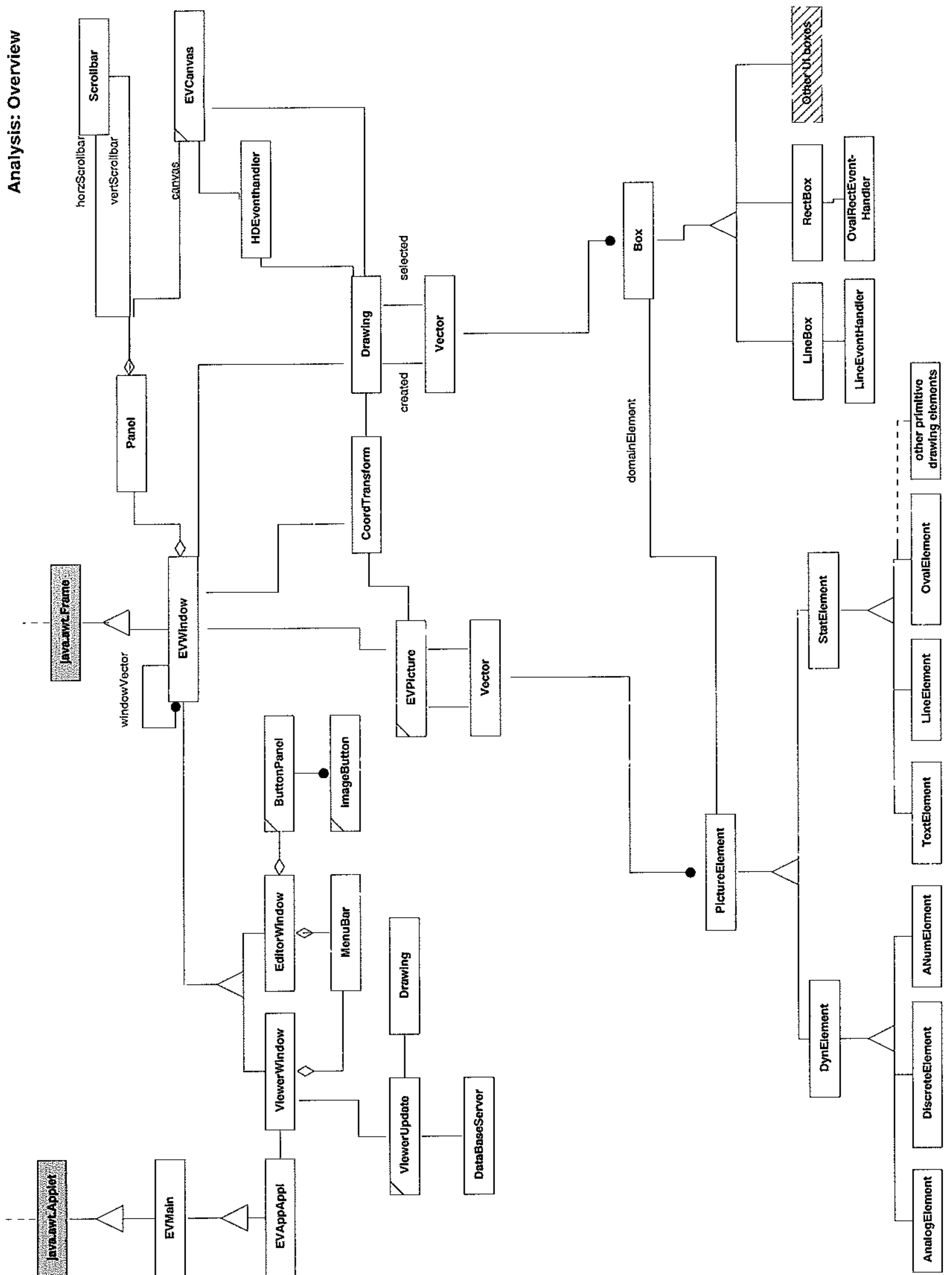


Figure 6.4, The object model resulting from the analysis.

Req. 1 suggests that there should be two different modes edit and view. Figure 6.5 shows the solution chosen for this requirement.

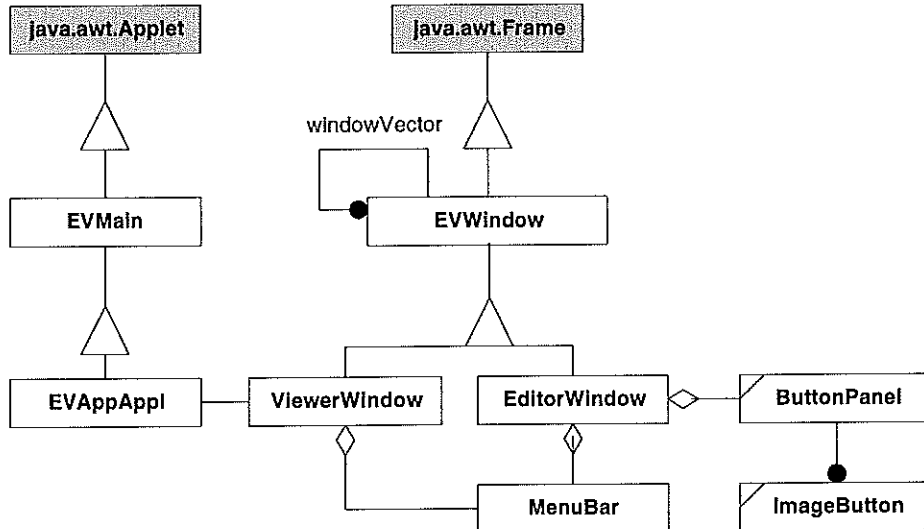


Figure 6.5, The prototypes' windows.

The object diagram suggests that there should be two different types of windows, a editor window and a viewer window specialized from the parent class EVWindow. The possibility to parallelly work with multiple documents is also covered by the diagram, through the zero or more windowVector association on EVWindow. The different components associated with the two types of windows are also shown in the model (dialogs are excluded). The left part of Figure 6.5 shows the class EVAppAppl, which has to do with *Req. 10*, that a picture object should at least be viewable from a Internet browser. The class allows the prototype to start in two modes: applet or application mode.

The requirements, *Req. 2* and *Req. 3*, are covered by Figure 6.6.

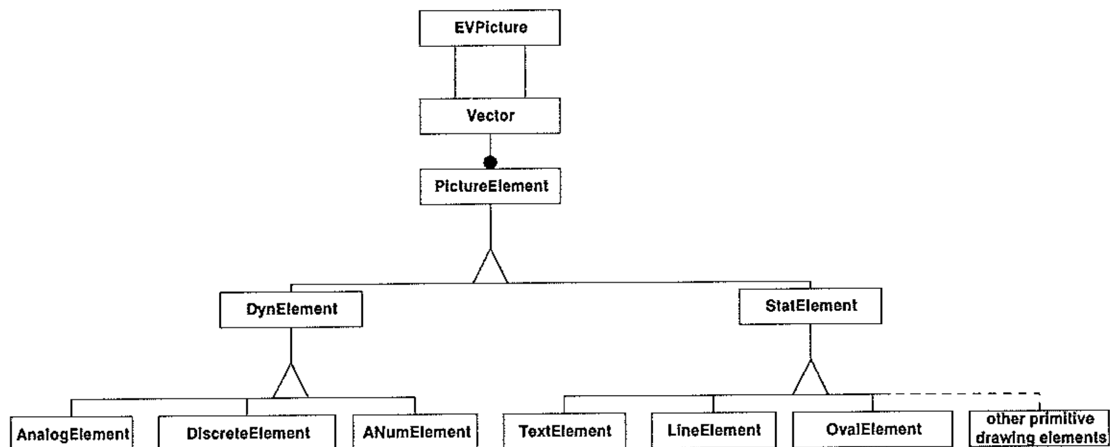


Figure 6.6, The domain model.

Figure 6.6 is referred to as the domain model, as this structure describes the picture object (PO) in world coordinates (*Req. 5* considered). *Req. 2*, states that a PO should be a collection of generic picture elements (PE). The above structure also considers *Req.3*, which states that a PE can be dynamic or static, by inheritance from the PictureElement class by the DynElement and StatElement classes.

Requirement 6 says that it should be possible to transfer world coordinates into pixel coordinates. This so that a PO can be presented in a window on the screen. Figure 6.7 shows the structure for how this is accomplished. The heart of the structure is the CoordTransform class which is responsible for the transformation between world coordinates and pixel coordinates as well as the opposite transformation.

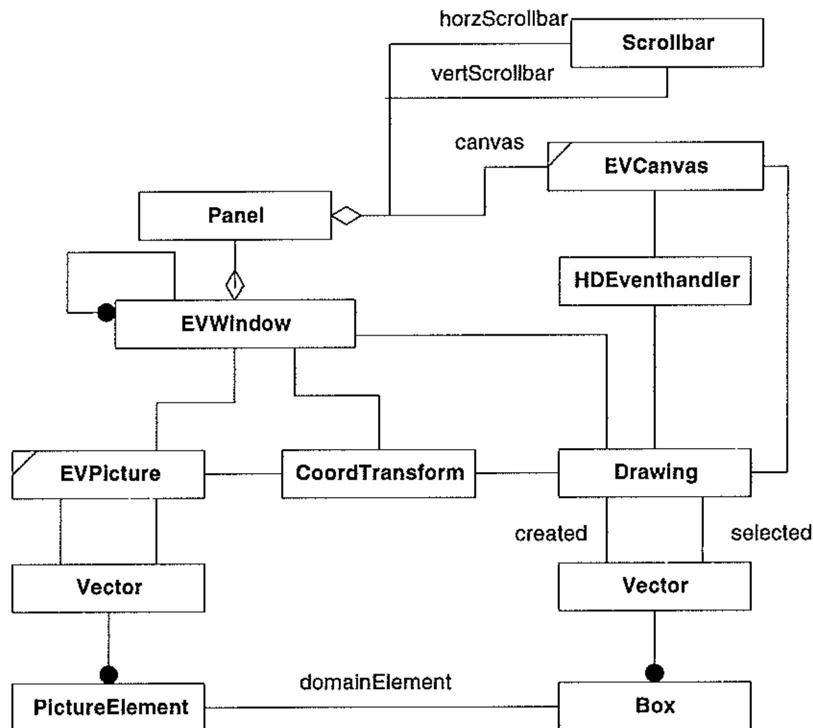


Figure 6.7, The two models: The domain model and the UI model and their interdependence.

The CoordTransform class can be seen as the bridge between the model of the PO expressed in world coordinates (referred to as the domain model) and the model of the PO expressed in pixel coordinates (referred to as the UI model). The domain model is held by the EVPicture class through the Vector class and the UI model is held by the Drawing class through the Vector class. The PictureElement is represented by the class Box in the UI model. The functionality stated by *Req. 11* and *Req. 12* is also covered by this structure, i.e. the possibility to zoom and pan. The panning is partly enabled by the Scrollbar and the CoordTransform classes. Coordtransform also enables the zooming of a PO. The portability of a PO (*Req. 17*) is promoted through the separation into two models, UI and domain model. The fact, that if there are only associations to the domain model and never from it, will make it possible to use the domain model independent of the other classes. This makes the domain model completely portable and thereby usable by other applications or applets.

Requirements 7 and 8, which has to do with a PE:s width and height and the preservation of the ratio between these two quantities, can also be fulfilled by the structure suggested by Figure 6.7. The structure also takes care of *Req. 13* which states the minimum size of the world coordinate system.

There are a number of specialized classes from Box, which represent the UI representation of the different PE:s in the domain model. These classes have names like LineBox, RectBox and so on, each corresponding to the classes LineElement, RectElement, etc., in the domain model. To every such class there is a connected eventhandler, which controls the behavior of the corresponding box object. The structure, called the UI-model, is shown in Figure 6.8.

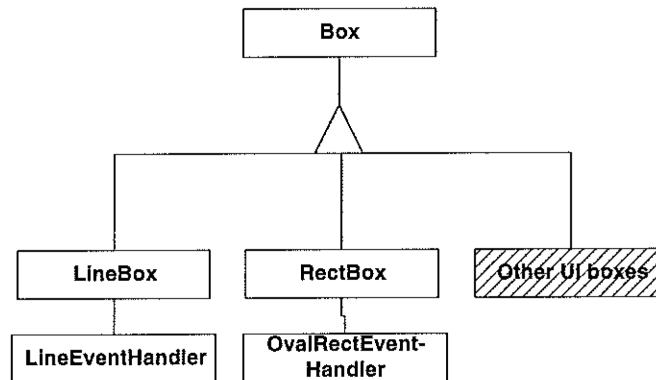


Figure 6.8, The UI-model.

The requirement 4, that the dynamic PE:s should be updated by a info server is considered by the structure of Figure 6.9.

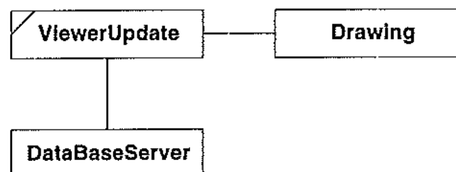


Figure 6.9, updating of dynamic PE:s.

The actual updating should be performed by the ViewerUpdate class. Associated with the ViewerUpdate class is a DataBaseServer class, which is supposed to handle the information gathering from the control system. There is also a association with the Drawing class, i.e. the UI model. An association with the domain model is not needed as the interesting thing is to view the updates, and the UI model is the viewable model. The ViewerUpdate class should be specialized from the Thread class (supplied by the java API framework). This because updating should take place concurrently with the other activities performed by the editor.

6.3.2 Functional Model

The functional model of the OMT-method, has not been used here. Instead, a number of scenarios have been defined. The scenarios deals with typical operations in the prototype, such as the drawing of a PE in the UI model, zooming and so on. The functional behavior of the prototype comes from the scenarios.

6.3.3 Dynamic Model

The dynamic model can present state diagrams, event sequence diagrams and event flow diagrams. In this design it was found sufficient to use event sequence diagrams.

Running the program as an applet

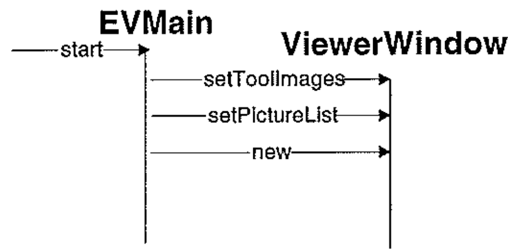


Figure 6.10, start of the program as an applet.

Figure 6.10 shows how the program is started, if it run as an applet, i.e. from the applet viewer or an Internet browser. To start the program as an applet a .html file containing the <applet> tag must be supplied. The EVMain class is specialized from the Applet class and therefore a start message is sent to it, when starting as an applet. After that some initializations are made, like loading the bitmaps for the toolimages and reading the parameters in the .html file listing the available pictures on the server.

Running the program as an application

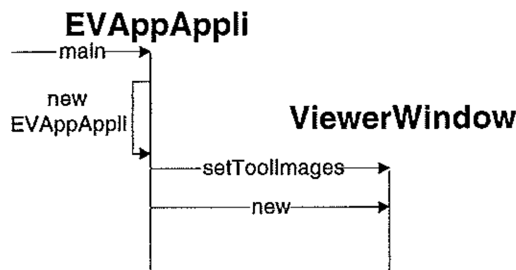


Figure 6.11, start of the program as an application.

The class EVAppAppl in Figure 6.11 is specialized from EVMain, but this time the start method is not called, but the main method, as the program is now run as an application. This architecture allows the program to sense in which mode it has been started and adapt itself accordingly.

Adjustment of the window size

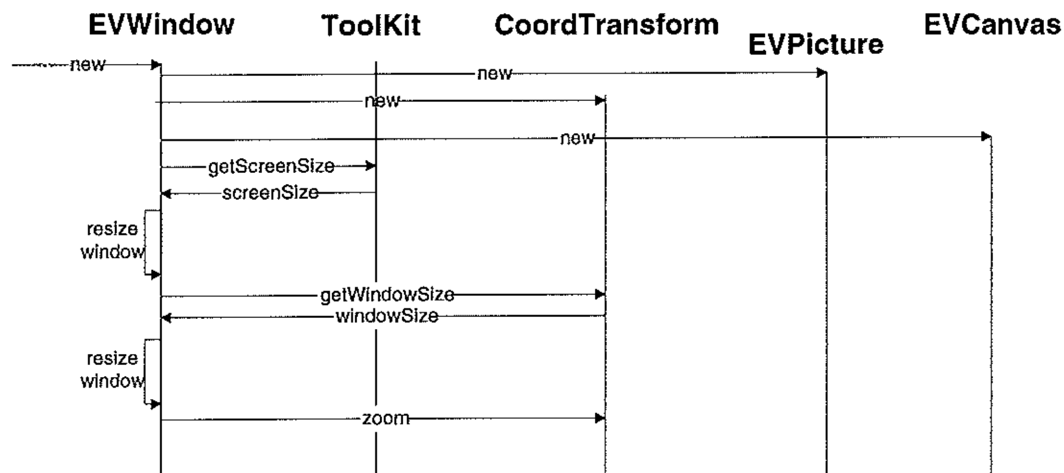


Figure 6.12, Adjustment of the window size.

As the program is started and a new picture is opened, the editor or viewer window must adjust its width to height ratio to be the same as that of the domain model. Or, to be correct, the width to height ratio of the canvas must be exactly the same as the corresponding ratio of the domain model, if a linear transformation is to be performed. Thus, the window must be resized to allow the canvas to get a preferred width to height ratio. This, requires some knowledge about the components' sizes in the window as well as what canvas size a certain choice of window size renders. Figure 6.12 shows one solution to this problem.

The Zoom Procedure

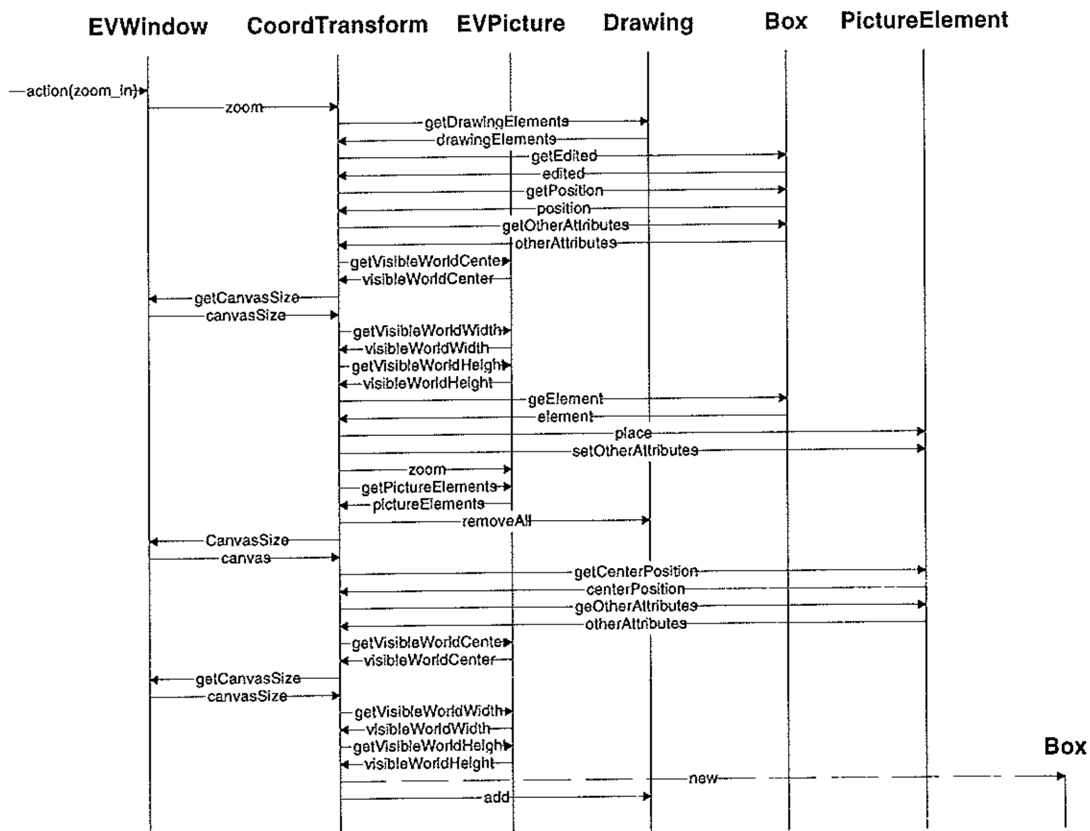


Figure 6.13, The zoom procedure.

Figure 6.13 shows the messages required to perform zooming in the program. The procedure begins with the user choosing Zoom In or Zoom Out from the View menu alternative. Then, the newly created or edited drawing elements on the canvas are found (i.e. the UI-model is verified) and the corresponding domain elements are updated. Then the domain model is zoomed (the message zoom from CoordTransform to EVPicture), i.e. the part of the domain model containing information about the area that is visible to the user is recalculated. After that, the domain model is read and all elements are removed from the UI-model. From the reading of the domain model, new resized elements are created in the UI-model. This require a number of messages concerned with recalculating distances and positions in the UI-model. Some of these self explanatory methods are accounted for in Figure 6.13.

The Pan Procedure

The pan procedure is identical with the zoom procedure, except for the message zoom between CoordTransform and EVPicture. Instead of increasing or decreasing the size of the visible area (as with Zoom Out or Zoom In) the size of the visible area is kept in the domain model, but the position of the visible area is moved, i.e. panned.

The Saving of a Picture and the Generation of Java Code

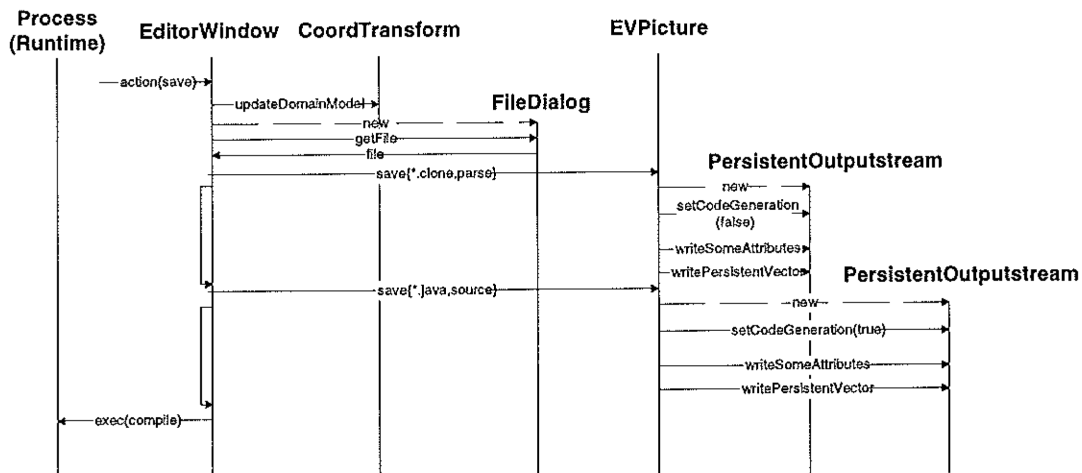


Figure 6.14, save and generate code procedure.

It is only possible to save a picture from the EditorWindow and it is only possible to launch an editor window, when in application mode (as access to the local filesystem is only granted in this mode). To save a picture the user chooses the save or save as menu alternative. First the domain model is updated, i.e. verification of the UI-model to see if some elements have been edited or created. If the save as menu alternative is chosen the user is presented with a standard filedialog. After that the EditorWindow knows the name of the picture to be saved. Then two versions of the picture are saved, a *.clone, which is a parsed version of the picture, and a *.java which also becomes compiled to a .class version.

It may seem a bit awkward to save two versions of the picture, a parsed one and a compiled one. The reasons for doing this are system consistency reasons. When the java runtime system encounters a class the first time it is registered to the java runtime system and after that it cannot be altered. By recompiling a picture class, the class is altered. But only the file version, *.class file, is altered as the java runtime system has already loaded the class previously. To overcome this problem, i.e. to open a newly saved picture without shutting down the system and making the runtime system unaware of the picture class, a *.clone version of the picture is always saved and then this version is parsed back into the system. The *.class version is used when opening a file from a ViewerWindow, when running as an applet.

The actual parsing and java code generation is handled by the interface Persistent, and the classes PersistentInputStream and PersistentOutputStream. The Persistent interface defines read and write methods for the classes that implement it. This allows objects to write information about themselves to file. And the method setCodeGeneration decides whether java code should be generated or not.

The Opening of a Picture, in Application Mode.

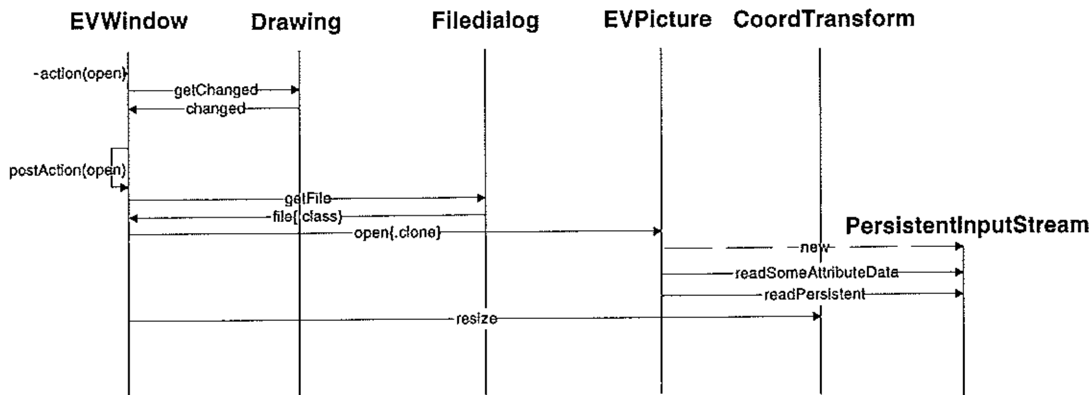


Figure 6.15, the opening of a picture in application mode.

When running the program in application mode access to the computer filesystem is granted. When the user chooses the Open menu alternative, the picture is checked to see if any changes has been made as compared to the saved version of the picture. If that is the case the user is prompted with a dialog informing him that the picture has changed and if he wishes to continue despite of this fact.

After this dialog has been shown, and the user wishes to continue, he is prompted with a file dialog box. The user chooses a file to open with the extension .class for a compiled version of the picture or an arbitrary extension for a parsed version of the picture.

If the user is not running the program from an internet browser (applet mode) the file will be parsed, if the user chooses a *.class file from the file dialog the corresponding *.clone file is chosen. The actual parsing takes place, with the help of the PersistentInputStream class and the readPersistent method.

The Opening of a Picture, in Applet Mode.

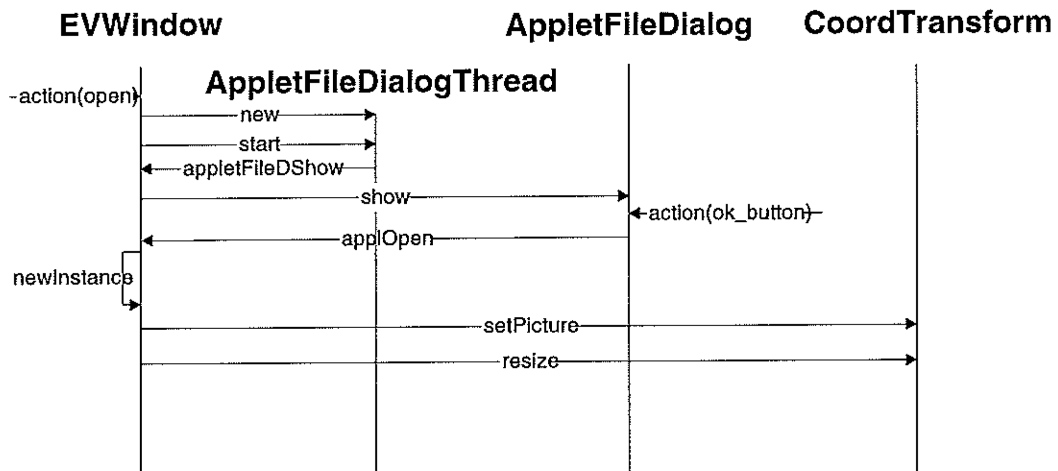


Figure 6.16, the opening of a picture in applet mode.

The event sequence diagram for opening a file from an internet browser is shown in Figure 6.16. First the user is prompted with a dialog, containing the available pictures. A list of available pictures is supplied by the *.html file from which the applet is started. The structure for launching the dialog has to do with the previously mentioned modal blocking bug.

When the picture is actually opened a new instance of the compiled picture class (the *.class file) is instantiated. The picture which the CoordTransform works with is set to this picture and the UI-model is verified by the resize method in the CoordTransform class.

The Creation of an Element on the Canvas

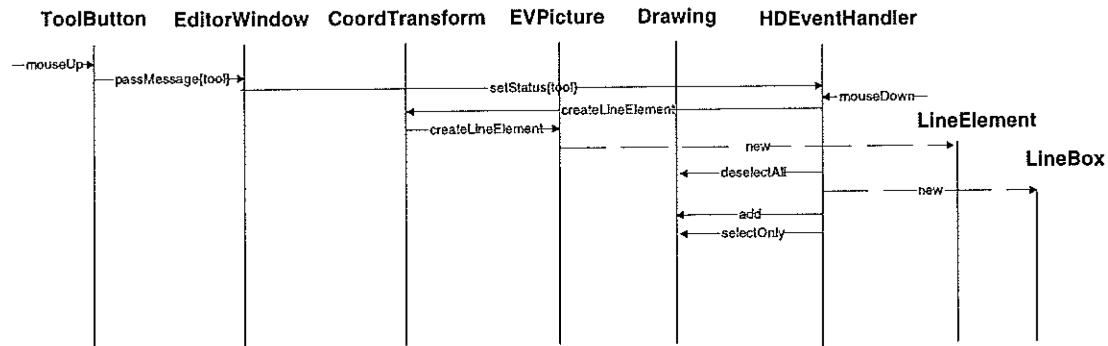


Figure 6.17, creating an element on the canvas.

To create an element on the canvas (assumes editor window) the user must push the appropriate tool button (in this example the line tool). The EditorWindow is informed of this, and it passes the choice on to the canvas eventhandler (HDEventHandler), where its state is set to the tool selection. Then, when a mouse down is received by HDEventHandler (i.e. the user pushes the mousebutton on the canvas), a request to Coordtransform of creating a new LineElement (i.e. a domain element) is sent. This request is passed on to EVPicture and a LineElement is created. All elements on the canvas are deselected and a LineBox (i.e. a UI element) is created. This LineBox is added to the Drawing, which holds all the elements in the UI model and the LineBox then becomes selected.

Updating of Dynamic Elements in Viewer Mode

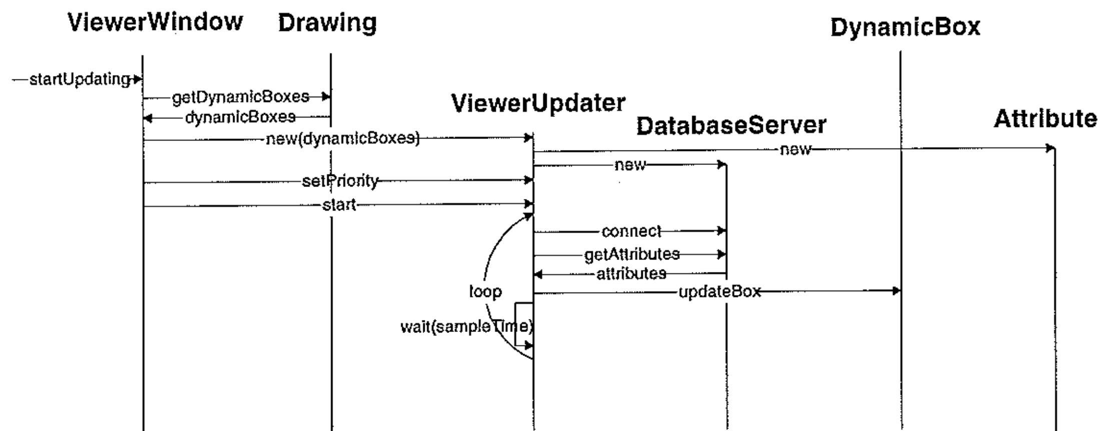


Figure 6.18, updating dynamic elements on the canvas.

When a picture is opened in a viewer window the startUpdating method is called. The first thing that happens is that the dynamic boxes are looked up and passed on to the constructor of a ViewerUpdater object. The ViewerUpdater object is launched in a thread. The ViewerUpdater class holds an Attribute class, which holds the database references to be fetched from the database, the references are supplied by the dynamic boxes. The ViewerUpdater class also holds a DatabaseServer class. When the Thread is put to a running state by calling the start method from the ViewerWindow, a connection to the database is established. Then the attributes are fetched from the database and after that the dynamic boxes are updated by use of polymorphism. Then the Thread waits a second followed by a new cycle of the previous procedure, starting with establishing a connection to the database.

6.4 System Design

Normally, the system design part of the OMT-method describes the different subsystems, the Graphical User Interface (GUI), the division into processes and the data management of the system.

As much of the editor/viewer prototype's functionality lies within the GUI, this part has already been considered during analysis. The only part relevant to the prototype is the data management part.

6.4.1 Data Management

The data management of the editor/viewer prototype must consider requirement 16 of the requirements list. *Req. 16* states that Java source code should be generated for the PO.

A solution to requirement 16 is to let each subclass of the baseclass PictureElement in the domain model hold the knowledge of how to generate its own Java-code. The generated Java source code for the PO should ideally be a subclass of EVPicture, the class holding the domain model.

The basic structure of the Java source code for a PO could look like in Figure 6.19.

```
// Editor-Viewer selfgenerated java-picture
import java.io.*;
import SE.abb.senet.EVDomainElements.*;
import SE.abb.senet.EVDomainUtilities.*;

public class bild40 extends EVPicture {
  bild40() {
     super();
     created.removeAllElements();
     domainSize[0] = 32768;
     domainSize[1] = 16384;
     visibleDomainSize[0] = 32768;
     visibleDomainSize[1] = 16384;
     visibleAreaCenter[0] = 0;
     visibleAreaCenter[1] = 0;
     created.addElement(new SE.abb.senet.EVDomainElements.LineElement(0, 0, 0, 0, 0, 0, 1,
 -5856, 2112, -13216, -3968));
     created.addElement(new SE.abb.senet.EVDomainElements.LineElement(0, 0, 0, 0, 0, 0, 1,
4576, -864, -6016, -1600));
     created.addElement(new SE.abb.senet.EVDomainElements.LineElement(0, 0, 0, 0, 0, 0, 1,
9856, 6784, 4448, 1984));
     created.addElement(new SE.abb.senet.EVDomainElements.LineElement(0, 0, 0, 0, 0, 0, 1,
 9504, -2368, 9984, 3808));
     created.addElement(new SE.abb.senet.EVDomainElements.LineElement(0, 0, 0, 0, 0, 0, 1,
-3840, -4320, 5376, -4288));
  }
} //class bild40.java
```

Figure 6.19, Possible structure of the Java source code for a picture object.

The Java source code can then be compiled and then it will behave like any other class in the Java system.

To achieve that each subclass of PictureElement holds the knowledge of selfgeneration through Java source code a number of techniques could be deployed. In some way each object must create itself on file. In C+ +, the concept of stream operators can be utilized to achieve this. In the next release of the Java API the concept of Object Serialization will be supported(cf. JDK 1.1, http://www.javasoft.com/products/JDK/1.1/designspecs/index.html [W5]).

Meanwhile, the problem can be solved with the solution suggested in the book CoreJava [2], by the notion of persistent storage (cf. CoreJava pp.482-498). By letting the PictureElement implement the Persistent interface, each subclass of PictureElement must define read and write procedures, that allows the class to create itself on file and recreate itself from file. By doing this and utilizing the

PersistantInputStream and PersistantOutputstream classes, found on pp. 492 - 498, and slightly modifying these classes the problem is solved. The modification is to add some java-code to the values written to file by these classes. Then, a parsed as well as a Java source code version of the domain model could be written to file.

Now by compiling this Java file describing the PO, the editor can create a class that is part of the system. The next time you need to look at the class or edit the class an Object of the PO class is created. This solution has some problems and advantages associated with it.

The advantages of a picture being a class, are when running in applet mode, there is no difference between the applet code loaded from the server and the data (the pictures) loaded from the server as both are classes. This also contributes to the speed of picture creation as a picture is a compiled class, from which it is possible to create a new instance.

But, there are also some problems associated with this solution. One problem is if you edit a picture and want to save it under the same name there will be problems. As the picturename is the class name and if you redefine a class during runtime there will be inconsistencies in the system (cf. article in JavaWorld [A2]). The solution to this problem would be to parse the picturefile at runtime. The other type of problems that occur are due to applet security restrictions and are covered in the section below.

6.4.2 Data Managemnet Limitations

When running an editor the user might want to save the picture he has created to the local filesystem. This is not possible, when running the system in applet mode, as the applet security restrictions denies access to the local filesystem. It is also not possible to run a system call, i.e. run the Java compiler javac, when running in applet mode.

To overcome this a server could be written to handle the saving of pictures as well as compilation of pictures. This functionality must then lie on the server side. But, on the other hand it would make it possible to run not only the viewer, but also an editor from a internet browser.

6.4.3 Network Communication Limitations

Applet Security Restrictions also say that an applet may only contact the server it was downloaded from. This calls for some server functionality regarding the DbServer Class that contacts the S.P.I.D.E.R system to get data for dynamic PE:s. The DbServer class can not directly contact the database which is located at some IP address. It must go through the server from which the applet was downloaded.

6.5 Object Design

6.5.1 Overview

In the object design phase, the three models that have been created during the analysis stage and the system design stage of development are put together. The result will be a new object model with messages as well as attributes added as a result of the analysis of functional and dynamic models. The different parts of the systems' object design will be presented below.

6.5.2 The Heart of the Design

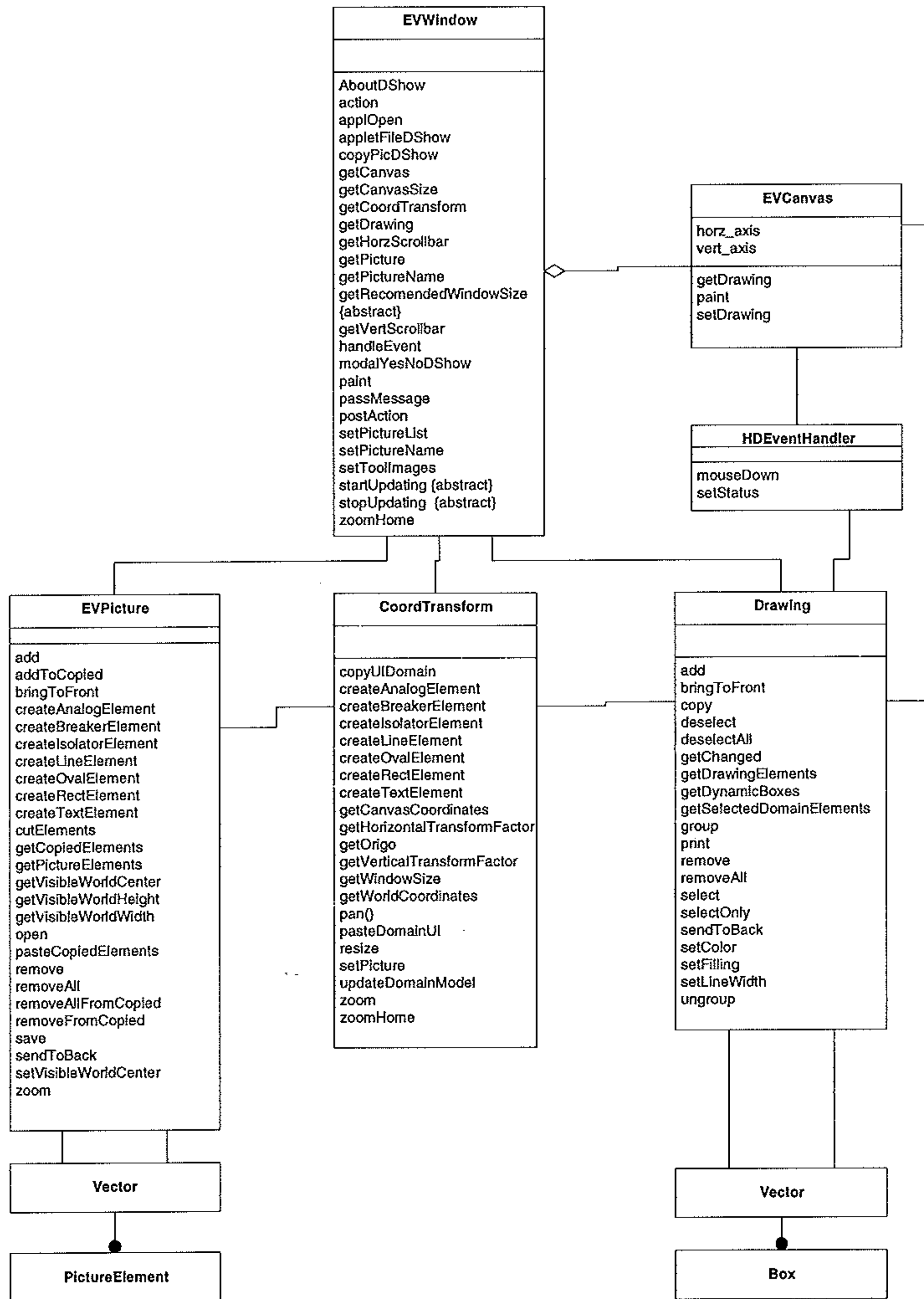


Figure 6.20, The heart of the design.

The heart of the editor/viewer prototype is shown by Figure 6.20. An exhaustive presentation of all the methods in the figure will not be made, only the most important will be presented.

Methods In EVWindow:

- **action:** This method takes care of all action events in the window, mainly menu selections. Sometimes a menu selection causes a dialog to be posted. Depending on the dialog input, the postAction method is then called.
- **handleEvent:** This method takes care of all sorts of events in the window, e.g. it senses when the scrollbars are moved.
- **applOpen:** This is the open method that is called when the program is run as an applet. It merely instantiates a subclass of EVPicture (i.e. a picture generated by the java-code generating editor).
- **getRecommendedWindowSize:** abstract method, that must be defined in the subclasses of EVWindow (i.e. EditorWindow and ViewerWindow). The input to the method is the canvas size and the output is which window size this canvas size yields.
- **paint:** This method contains information of how to create the contents of the window, as well as how to catch a resize of the window (cf. Implementation chapter).
- **passMessage:** This is a callback method, a push on one of the toolbuttons causes a callback and so on.
- **postAction:** This is also a callback function from some of the modal dialogs.

Methods in EVPicture:

- **open:** Causes a picture to be parsed and loaded into the system through the PersistentInputStreamClass and the readPersistentVector method.
- **save:** Causes a picture to be saved through parsing or through the generation of its Java-code. This is done through the PersistentOutputStream class and the writePersistentVector method.
- **setVisibleWorldCenter:** Sets the centerpoint of the domain model in world coordinates, corresponding to the centerpoint of the canvas (i.e. the visible part of the UI model).
- **zoom:** zooms the domain model that is the visible extent in world coordinates is recalculated.

Methods in CoordTransform:

- **copyUIDomain:** The selected and copied elements in the UI model are transferred and stored as their domain model equivalents. This makes correct copying between different zoomlevels possible.
- **getCanvasCoordinates:** Calculates the corresponding coordinates on the canvas, for a given set of world coordinates.
- **getHorizontalTransformFactor:** Calculates the horizontal transformation factor between the domain model and UI model at the current zoom level.

- **getOrigo:** In the domain model origo is at the center of the coordinate system. On the canvas origo is at the upperleft corner. This method calculates the center coordinates of the canvas.

- **getVerticalTransformFactor:** calculates the vertical transformation factor in between the domain model and UI model at the current zoom level.

- **getWindowSize:** As the window contains a canvas which must have the same width to height ratio as the domain model if the transformation is to be linear. The width to height ratio of the domain model must be taken into consideration, when calculating the window size. This is done by taking, as input, a proposed window size which is based on the screensize and then making the best under the given constraints to meet the proposal.

- **getWorldCoordinates:** Calculates the corresponding coordinates in the domain model, for a given set of coordinates on the canvas.

- **pan:** Updates the domain model, zooms the domain model (reversed to the zoom method below, i.e. enlarges the visible extent) and then calls the resize method to recreate the UI model.

- **pasteDomainUI:** Takes the elements copied to the domain model and recreates them in the UI model, at an arbitrary zoom level.

- **resize:** Traverses the domain model and creates the elements in the UI-model at an arbitrary zoom level.

- **updateDomainModel:** Traverses the UI model and creates or alters the elements that have been edited in the domain model.

- **zoom:** Updates the domain model, zoomes the domain model and then calls the resize method to recreate the UI model.

Methods in Drawing:

- **getChanged:** As the drawing holds all the elements in the UI model, this method just finds out if an element in the model has been changed. This can be useful, if the user is to be notified of this fact when exiting the program or opening a new picture.

- **getDynamicBoxes:** Selects the dynamic boxes to a vector. This is useful for the ViewerUpdater class, which wants to update these boxes.

The EVCanvas class is a subclass of the Canvas class in the HiJC package and on this canvas all the elements of the UI model are drawn. The HDEventHandler class is an EventHandler for the canvas which takes care of events on the canvas and this class holds a status attribute, that modifies the behavior of the EventHandler depending on what tool has been selected.

6.5.3 The Editor and the Viewer Window

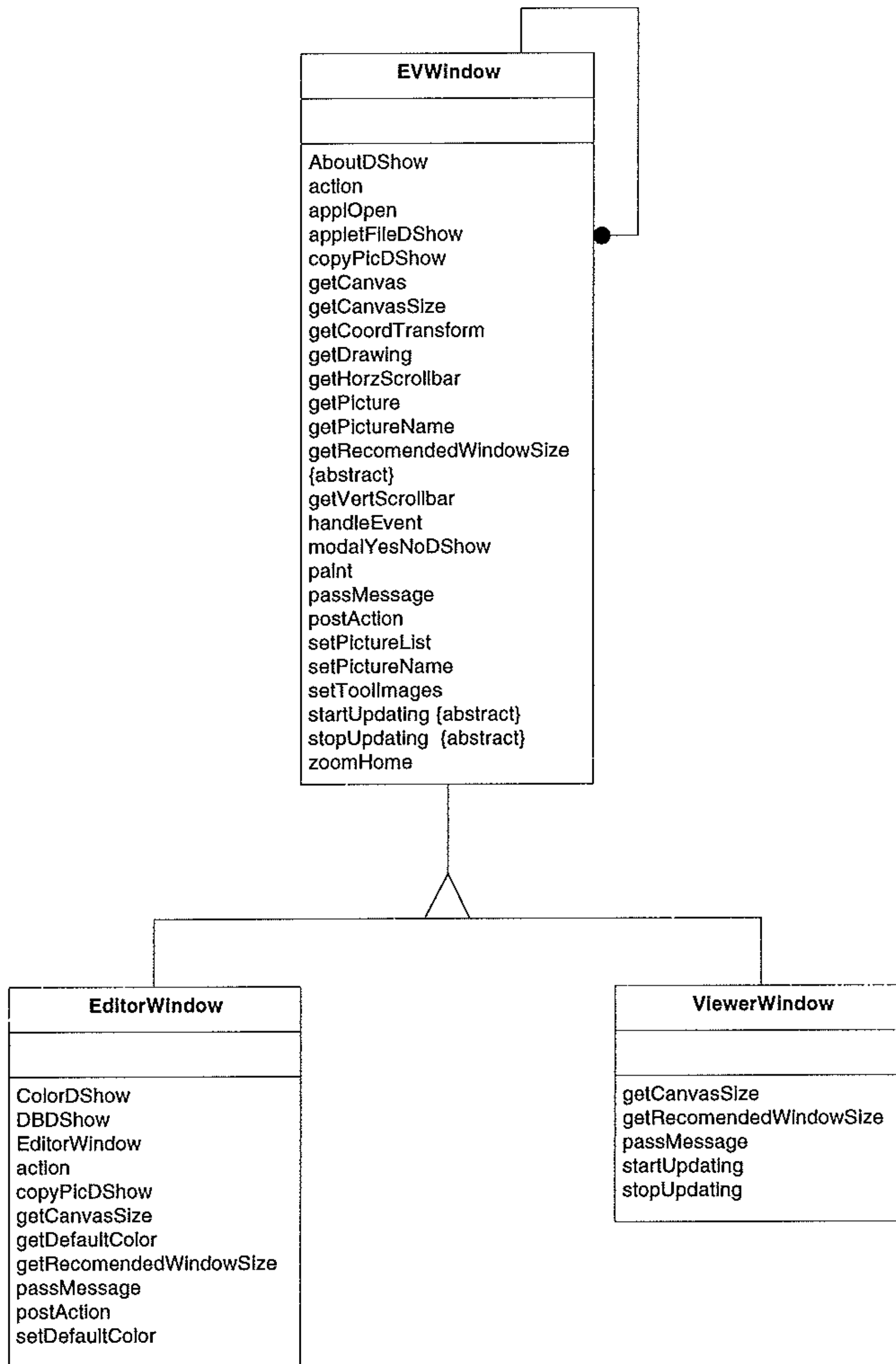


Figure 6.21, The editor- and viewerwindow structure.

The most important thing to mention about Figure 6.21 are the methods startUpdating and stopUpdating. The startUpdating method launches a ViewerUpdater thread, which updates the dynamic boxes. And the stopUpdating method stops the ViewerUpdater thread. This is applicable to the ViewerWindow class, in the EVWindow class they are just empty methods.

6.5.4 Design Considerations, due to Applet Security Restrictions: Program Start.

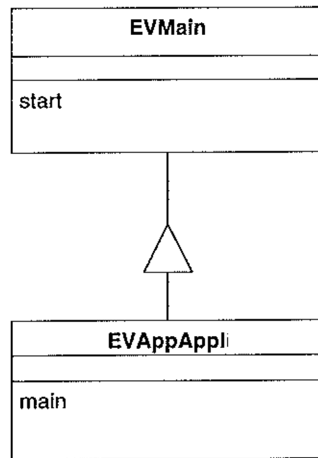


Figure 6.22, The different start modes of the program.

Depending on if the program is started as an application (i.e. java *programname*) or it is started as an applet (i.e. from an Internet browser) the main method is called or the start method is called.

6.5.5 The UI model

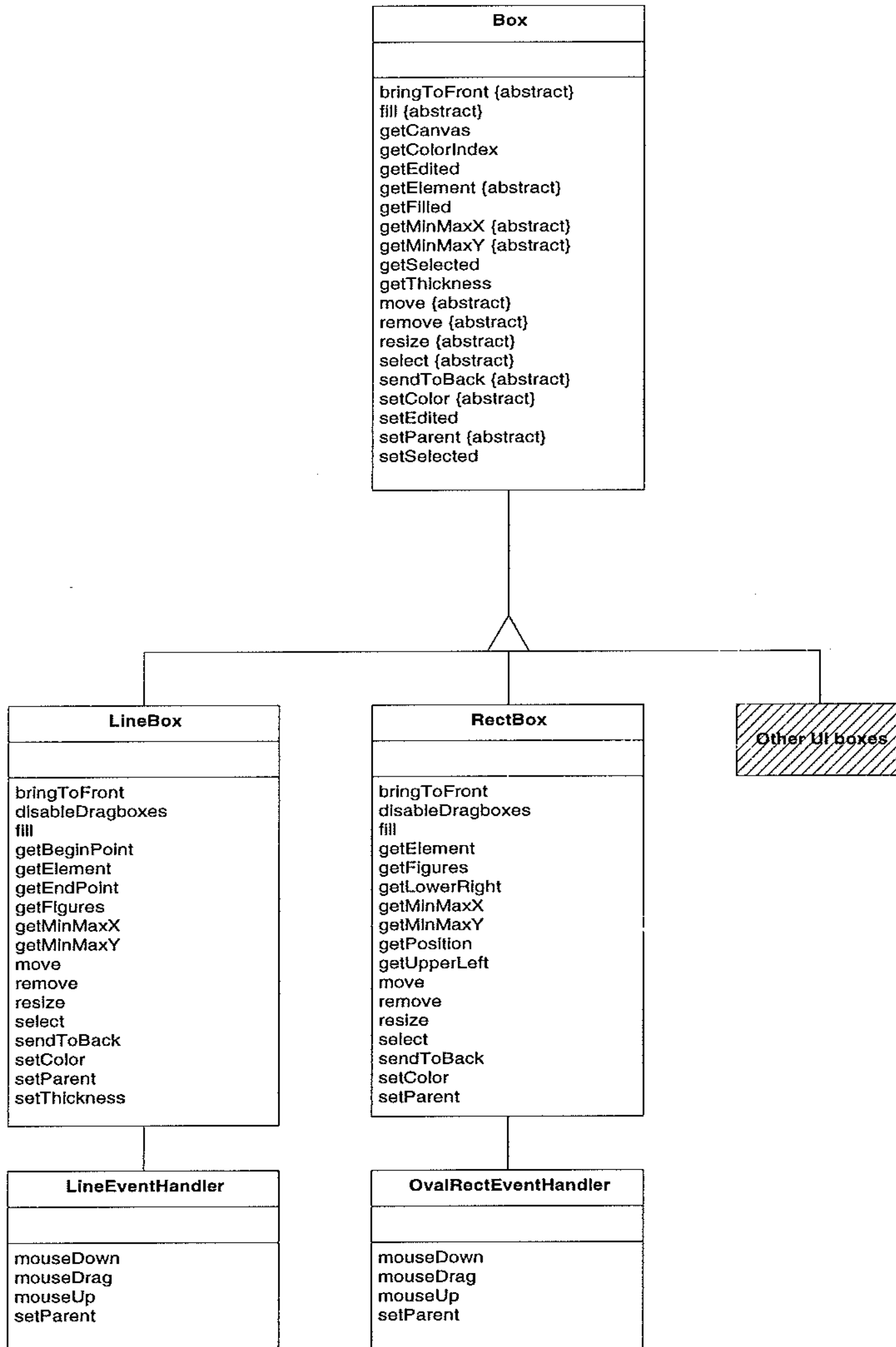


Figure 6.23, The UI model.

Figure 6.23 shows the UI model.

6.5.6 The Domain model

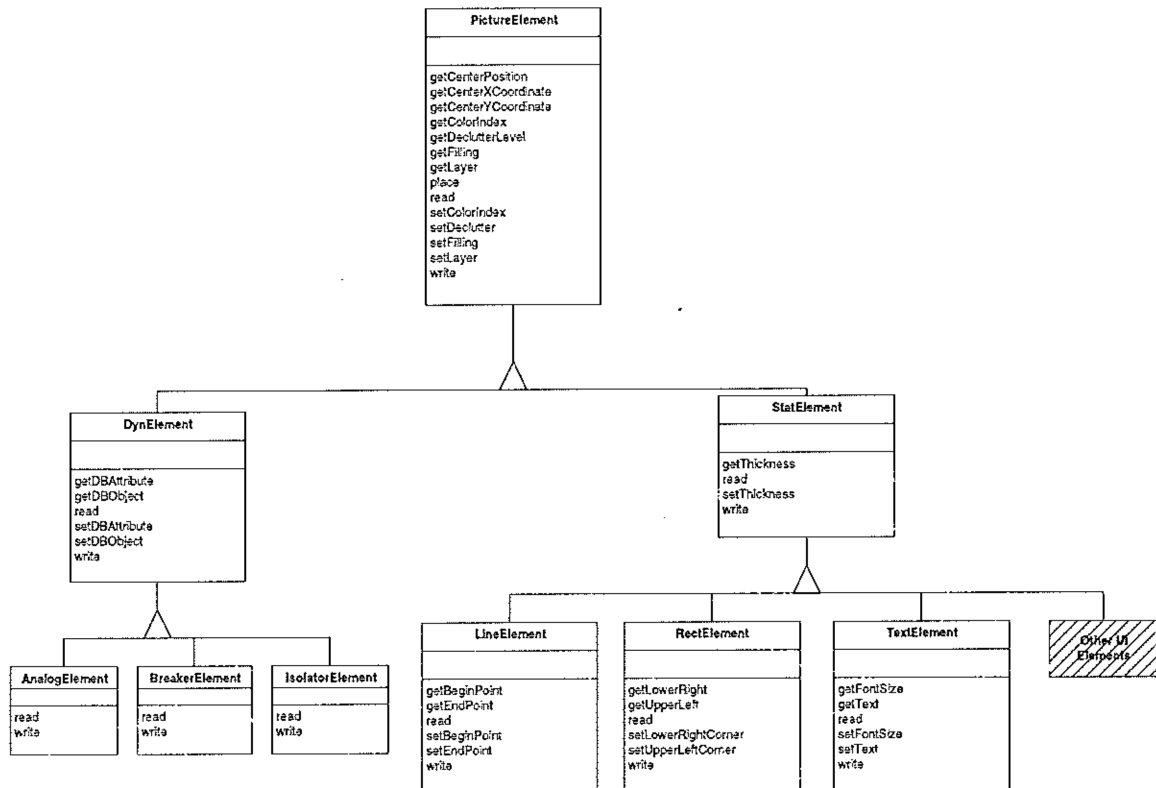


Figure 6.24, The domain model.

Figure 6.24 shows the domain model. The setDeclutter and getDeclutter methods of the PictureElement class are there in order to support future extensions with information zoom. This design allows information zoom to be implemented easily just by considering the declutter level (which can be linked to a zoom level) when creating the UI model in the CoordTransform class.

6.5.7 Updating of the ViewerWindow

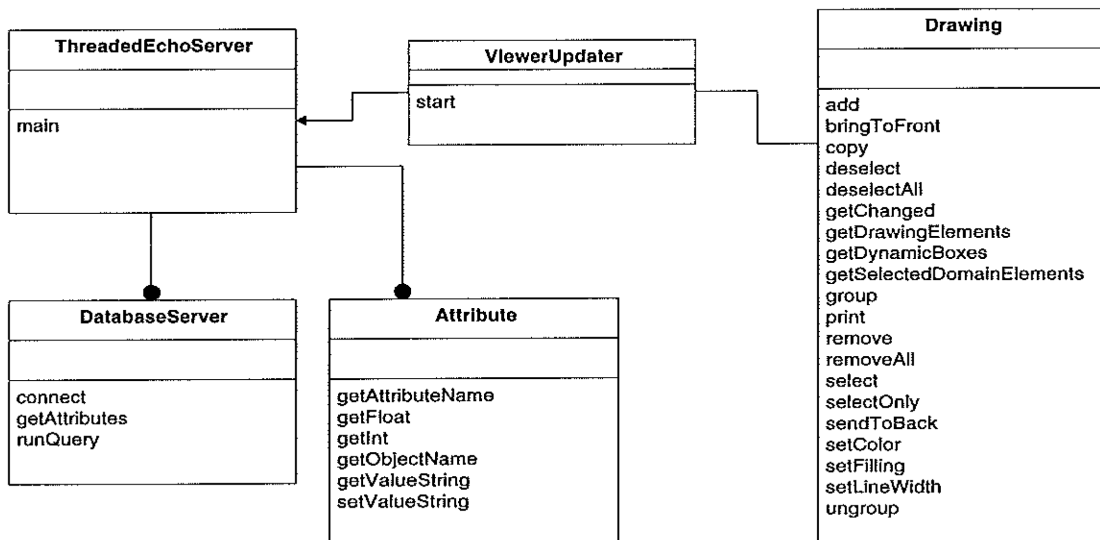


Figure 6.25, The updating of dynamic boxes in the ViewerWindow.

Figure 6.25 shows how the updating of dynamic elements is handled. The ViewerUpdater class, which is launched in a thread, takes the dynamic elements from the Drawing class and then makes a socket connection to the ThreadedEchoServer, which must be installed on the same computer as the HTTP-server (when in applet mode). The ThreadedEchoServer can handle multiple clients.

When the start method of the ViewerUpdater is called it goes into a loop where it communicates with the ThreadedEchoServer (which communicates with the S.P.I.D.E.R database) and the elements in the UI model are updated once per second.

7. Implementation

7.1 Overview

This chapter will cover some implementational aspects of the prototype. The chapter is divided into three sections, two of them covering the awt (Abstract Window Toolkit). In the first section extensions to the awt are covered, this may not be so important, as the API for GUI development is expected to be extended in future releases of the JDK (cf. Java 2D located at [W4]). The second section discusses some usual problems that are encountered when working with the awt. The last section, discusses the inner workings of the prototype. The workings of the HiJC are not discussed, as this is well documented on the web.

7.2 Extended AWT Functionality

7.2.1 The ToolButton Class

The ToolButton class is specialized from the ImageButton class, which contains the two methods setImage and setSelectedImage. The ToolButton is essentially a canvas onto which to images are drawn depending on if it is selected or not. The awt normally only supports TextButtons. The ToolButton also utilizes the Callback functionality (see below).

7.2.2 The PolyThickLine Class

The PolyThickLine class tries to emulate lines of arbitrary thickness on the canvas. The first attempt was to draw several lines of thickness one beside one another. This works fine for straight horizontal lines, but the thickness of the line will vary with the angle in which the line is drawn. So, the thick line, was instead implemented as a filled polygon, which works good. But, it seems as polygons must at least be two pixel high or two pixels wide, so there is a problem with line thickness one. Another problem that has not been entirely implemented is the BoundingBox definition of this thick line. These are all problems, which with some effort could be coped with. But as the purpose is to implement a prototype for a technology evaluation there is no need, and hopefully will the new release of the JDK support thick lines.

7.3 AWT Specific Properties

7.3.1 To Catch Resize

In the prototype it is important to know when the user resizes the window, as the width to height ratio of the canvas must be the same as that of the domain model. It seemed like a problem to catch the resize, since to my knowledge no event gets posted. But, the solution was found on the web at a the Java Developer site, in the "How do I..." section (cf. http://www.digitalfocus.com/digitalfocus/faq/howdoi.html [W6]).

How do I tell when a window has been resized?

"I read somewhere that a WINDOW_MOVED event will get posted during window resizes, but it doesn't seemed work for me."

(Submitted by the editor)

One thing that WILL happen is paint() will be called. Thus, in paint() you could check the window's size against the previous size:

```
// Check if window was just resized
public void paint(Graphics g)
{
    Dimension d = size();
    if ((d.width != curDim.width) || (d.height != curDim.height))
    {
        curDim = d;
        // do whatever...
    }
}
```

This is the solution that has been deployed in the windows' paint routine in the prototype.

7.3.2 The Updating of a Window

Another good point that is made in the "How do I..." section of the Java Developer [W6] is the mechanism by which windows are redrawn.

What exactly is the interaction between repaint() and update()/paint()? How do I indicate that only certain parts of my display need to be updated?

(Submitted by Eirc McCall)

Notice that paint() is called in two cases:

when the window is resized or de-iconified, or needs to be redrawn for obscure reasons (pun intended), and
when you haven't overridden the default update() method.

repaint() is the method you use to have the system call update() with the current graphics context. By default, update() calls paint() after clearing the display, and so the applet is painted again.

The problem occurs when you need to display things moving around, but also need some static background. Calling repaint() and having update() perform your simple animation is the right solution, but the animation drawing should be separated from the background drawing. Putting the animation into update() (or a separate function called by update(), or double-buffering, ...), putting the background drawing routines into paint(), and calling repaint() is a good solution.

7.3.3 How to Implement Scollbars

In order to make a scrollable canvas, you will often want to have scrollbars. First, the scrollbars are created and added to the Panel, where the canvas resides. Then, the second problem is to sense when the user moves the scrollbars. There is a callback to handleEvent when this happens. When program senses that a scrollbar has been moved the programmer must implement something wise to do, like translating the

bitmap on the canvas or recreate the objects on the canvas and so on. Below a scaled down example of the handleEvent routine of EVWindow is given.

```
public boolean handleEvent(Event event) {

    if (event.target == vertScrollbar) {
      if (vertScrollbar.getValue() != vertValue) {
         // do something.....
      }

      return(true);
    }

   if (event.target == horzScrollbar) {
      if (horzScrollbar.getValue() != horzValue) {
         // do something....
      }

      return(true);
    }
```

7.3.4 Bugs to be Patched in the JDK

There is a very annoying bug in JDK 1.0.2 concerning the posting of modal dialogs on the windows platform. If a modal dialog is posted the whole system becomes blocked. There is a patch to this bug as well as a thorough problem description(see [W4], the modal blocking bug). The patch is essentially to launch a Thread from which show on the dialog is made. But, this patch has shown some problems when using it with Internet browsers (Netscape 3.0 Gold and Microsoft Explorer 3.0), where it causes a "security exception: thread". This bug is supposed to be fixed for JDK 1.0.3, but in the meantime the picture selection dialog has been replaced with a Frame and this works fine.

7.4 Points to be Highlighted in the Prototype

7.4.1 The ColorDataBase and the MessageDataBase

Two static classes handling color and message passing in the system have been implemented: ColorDb and MessageDb. These two classes defines valid colors and valid messages to be passed in the program. As an example the message TOOL_LINE is passed to the window, when the user selects the line tool. This done through the interface CallBack and the method passMessage (see below).

7.4.2 The Implementation of Callbacks

Callbacks are implemented through the notion of interfaces. Interfaces can also be used to get some of the functionality of multiple inheritance, which is not supported by Java. In the prototype callbacks are implemented through the interface CallBack How to implement interfaces and callbacks is thoroughly described in CoreJava pp. 154 -161. The CallBack interface defines a method passMessage, the CallBack interface is then implemented by the EditorWindow class through the clause: ... EditorWindow implements CallBack, in the constructor. This forces the programmer to implement the passMessage method in EVWindow, and in that way messages can be passed to the window.

7.4.3 To Increase Performance of the Picture Drawing

The resize method of the CoordTransform class, basically handles all the drawing and resizing of UI components (i.e. the picture drawing). To tune the drawing performance, the hide method of the canvas should be called, as well as the delayRepaint(true) method of the canvas. Then, the UI components can be drawn. After the draw procedure is completed the delayRepaint(false) and the show() methods should be called. The call to the hide() method increases drawing performance drastically.

8. Results

8.1 Overview

In this chapter the implemented prototype and its properties are discussed. The prototype is also discussed in an Intranet context, and how it could be used if it was to be marketed as a product. Some personal reflections on using the Java language and the JDK, as well as the Symantec Café IDDE are made.

8.2 The Prototype

8.2.1 Description

The result of the design and implementation effort is a prototype that allows the user to edit and view pictures, which get input from the S.P.I.D.E.R system. The prototype can execute in two modes: as an application or as an applet from within an Internet browser. In application mode it is both possible to edit and to view pictures. In applet mode it is only possible to view pictures. In both modes multiple documents are supported (i.e. arbitrary number of viewer windows of the same picture, but only one editor window).

The reason why it is only possible to view pictures in applet mode, is the security restrictions of most Internet browsers, which prevents you from accessing the clients' filesystem. This problem could be overcome by adding functionality on the server side, so that it would be possible to save pictures on the server, thus making it possible to edit pictures online as well.

The performance of the prototype is a bit poor, about 4 seconds to draw and recalculate a picture for zooming, containing some 300 picture elements, on a 100 Mhz Pentium with JIT compilation enabled. Probably could this performance be improved a bit by optimizing the drawing algorithms and sorting the elements and using the tricks of the trade.

The implementation of the prototype contains 62 Java classes (the frameworks used are excluded). The average class is about 60 lines of code. There are about 6-7 larger classes. The largest two, EVWindow and CoordTransform, are about 880 lines of code each.

8.2.2 Application Appearance

Figure 8.1 an editor window is shown. At the top of the window, on the left hand side, there are some buttons, that allows the user to select a tool: such as a line or a breaker or an isolator. The attributes of the created elements can be edited by the buttons at the top of the window, to the right, as well as through menu selections. It is also possible to alter the color of the elements by launching a palette window from the menubar. Cut, Copy and Paste within and between documents is supported. It is also possible to zoom and pan a picture. The design also allows information zoom, through different declutter levels, but this option is not enabled in the prototype.

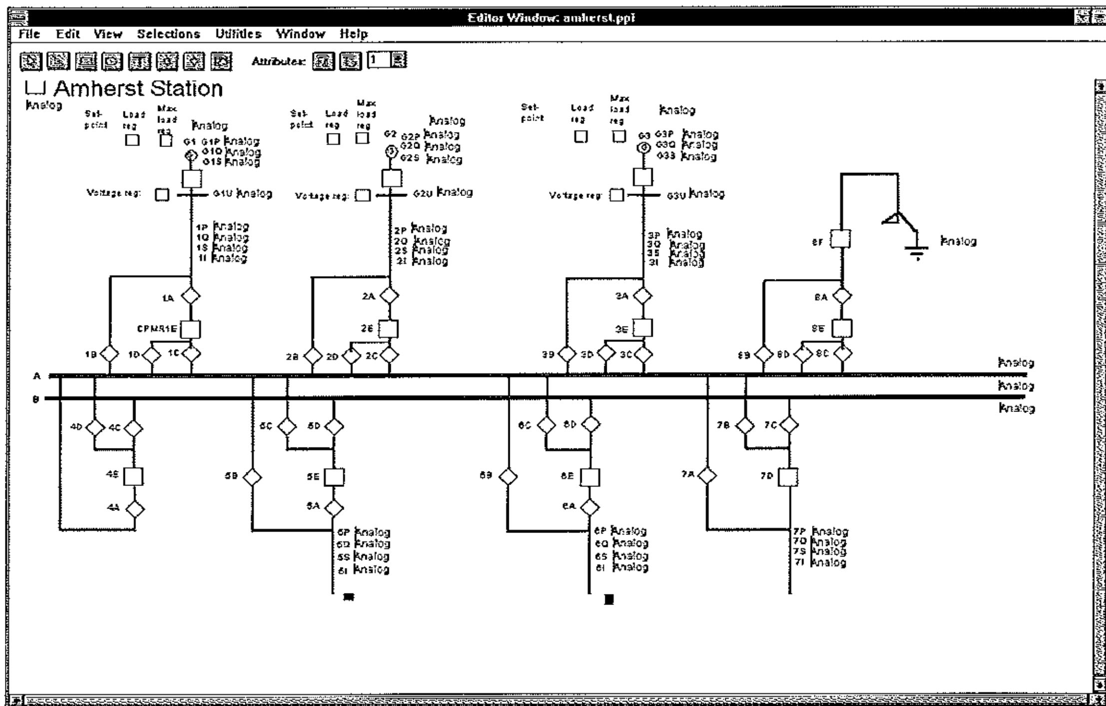


Figure 8.1, The appearance of the editor window in application mode.

To each of the dynamic elements (i.e. isolator-, breaker and analog elements) there is a connected database dialog, where the element is connected to a certain point and type of value in the S.P.I.D.E.R system. Figure 8.1 contains a station in the distribution net called Amherst, this station has been built with the editor (please note the analog elements in the picture, the text Analog will change to the database value of the point when the picture is viewed from a viewer window, see also the next section).

Figure 8.2 gives another example of how pictures are created in the editor window.

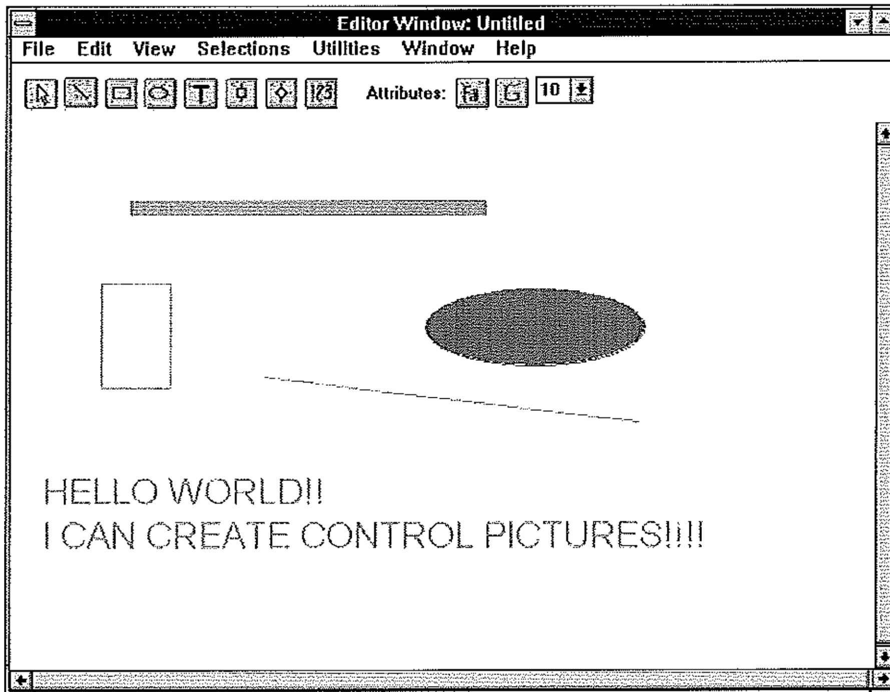


Figure 8.2, The editing of a picture.

The design of the prototype also makes it very easy to extend it with new types of elements as well as adding attributes and functionality to existing elements.

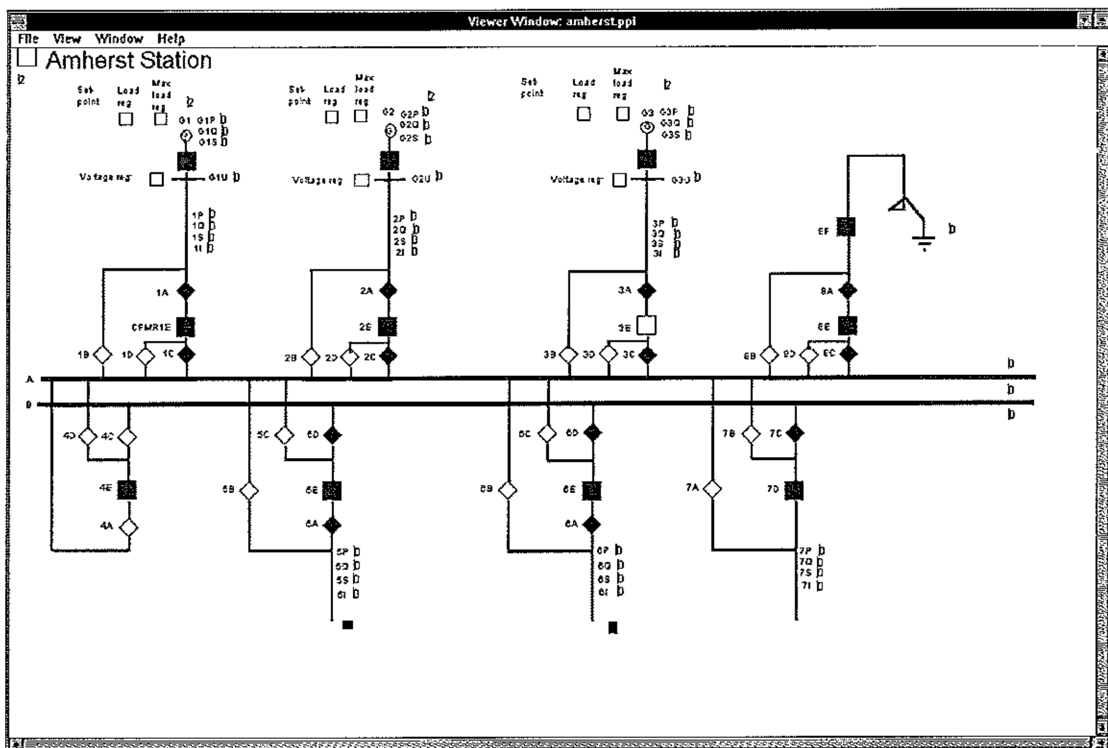


Figure 8.3, The appearance of the viewer window in application mode.

Figure 8.3 shows the viewer window in application mode. A viewer window is basically an editor window with scaled off functionality and with updating from the

S.P.I.D.E.R database. In viewer mode it is only possible to open and close pictures, as well as zooming and panning them.

8.2.3 Applet Appearance

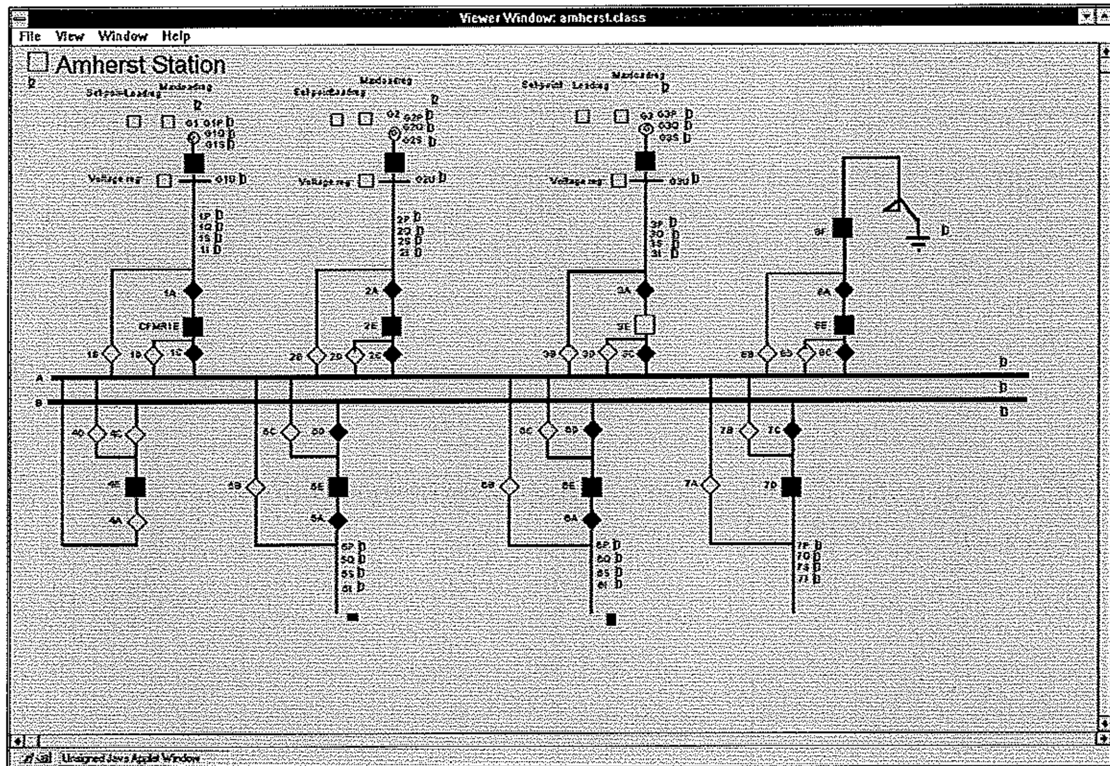


Figure 8.4, The viewer window as seen from the Netscape 3.0 Gold browser.

If the prototype is installed on a HTTP-server the pictures created by the editor can be viewed through a Internet browser, as seen in Figure 8.4. This could be useful for some power companies, giving them the possibility to view the status in their stations from any platform that is connected to the corporate computer network. Usually today within a company there exists many different types of computers as well as operating system (i.e. platforms). Given the possibility to build an Intranet utilizing web-techniques, the cost of having to install and develop and update new versions of the client for each computer type and operating system will vanish. The only program that needs to be installed on each computer is a web-browser, and those are cheap and available for many platforms.

It is probably also a relatively simple matter to convert pictures constructed by other editors to be viewable by the viewer applet. This conversion might also be an important factor, as there is no need for reinventing the wheel.

8.2.4 The Generation of Java Code and Parsing

```
// Editor-Viewer selfgenerated java-picture

import java.awt.Color;
import java.util.Vector;
import java.util.Stack;
import java.io.*;
import SE.abb.senet.EVDomainElements.*;
import SE.abb.senet.EVDomainUtilities.*;

public class SamplePic extends EVPicture {
  SamplePic() {
    super();
    created.removeAllElements();
    domainSize[0] = 25919;
    domainSize[1] = 16384;
    visibleDomainSize[0] = 25919;
    visibleDomainSize[1] = 16384;
    visibleAreaCenter[0] = 0;
    visibleAreaCenter[1] = 0;
    created.addElement(new SE.abb.senet.EVDomainElements.LineElement(0, 0, 0, 0, 0, 0, 1, -264, -902, -
9108, -4796));
    created.addElement(new SE.abb.senet.EVDomainElements.RectElement(-3652, -5390, 0, 0, 0, 0, 1, -
3652, -5390, 572, -2926));
    created.addElement(new SE.abb.senet.EVDomainElements.TextElement(-9658, -2904, 0, 0, 0, 0, 1, 10,
""));
    created.addElement(new SE.abb.senet.EVDomainElements.BreakerElement(4048, -1738, 0, 0, 0, 0,
"object", "attribute"));
  }
} //class SamplePic.java
```

Figure 8.5, Java source code for a picture generated by the editor.

Figure 8.5 shows the Java source code generated by the editor for a simple picture. The editor automatically generates the source code and compiles it, if the picture is saved with the .java-suffix. If the picture is saved with any other suffix the information will be parsed as in Figure 8.6.

```
domainSize[0]=25919,domainSize[1]=16384,visibleDomainSize[0]=25919,visibleDomainSize[1]=16384,
visibleAreaCenter[0]=0,visibleAreaCenter[1]=0,created=Vector[4,
SE.abb.senet.EVDomainElements.LineElement[0, 0, 0, 0, 0, 0, 1, -264, -902, -9108, -4796]1,
SE.abb.senet.EVDomainElements.RectElement[-3652, -5390, 0, 0, 0, 0, 1, -3652, -5390, 572, -2926]2,
SE.abb.senet.EVDomainElements.TextElement[-9658, -2904, 0, 0, 0, 0, 1, 10, ""]3,
SE.abb.senet.EVDomainElements.BreakerElement[4048, -1738, 0, 0, 0, 0, "object", "attribute"]4]
```

Figure 8.6, A parsed picture generated by the editor.

The parsing as well as the Java-code generation is handled by the objects in the domain model (cf. the design chapter). This functionality could easily be extended, other code than Java code could be generated and for other purposes than describing the appearance of the picture, or each element could be associated with links to information about the element (e.g. A script could be started that gets information from a relational dB about the element, such as service status, type, installation year, manufacturer, or why not some multimedial facilities showing a live video of the element and so on).

In the prototype, the pictures that are opened are created from compiled classes representing the picture, that are installed on the server. Practical experience with this system has shown, that it might be better to use parsed versions of the picture instead. This can be enabled through small server and client modifications.

The performance seems to be the same when loading from HD. This might depend on the need for class verification before running in the Java Virtual Machine. Another thing worth noticing is that the size of the .class file (14 kb) is about half of that the parsed version for the Amherst station. This might be important if the bandwidth of the network is low. But, the parsed version has many advantages, there is not the problem of system consistency due to many definitions of the same class in the runtime system. It is probably easier to build mechanisms for picture navigation (i.e. a picture directory system) with parsed files, because a .class file is not really loaded, it is rather seen as part of the program, and you just make a new instance of the class in order to view the picture.

8.3 Using Java For Development

Using the Java language was a really positive experience. Java resembles C++ a lot, and that made me feel comfortable as I have worked with C++ before. Some of the pitfalls that are common in C++, have been removed in Java. It was a relief not having to deal with pointers (based on the arithmetic pointer model) and destructors and the like. This also promoted object oriented programming practices. On the whole I found the language simple to learn and use.

But, it is also evident that Java is a new product. The JDK 1.0.2. and the libraries supplied with it do not suffice to develop a mature software product. This is evident from the section above, where a need persists to extend the Java AWT API. There were also some irritating bugs that were associated with the API (i.e. the dialog blocking bug). In order to build a complete application out of the prototype a more extensive library supporting GUI components is needed. Hopefully, the new release of the JDK 1.1 and the Java 2D API will overcome this problem. The JDBC API might also be useful for enhancing database communications.

The different behavior of applets in different browsers as well as in the applet viewer, is also somewhat irritating. The grounds for this might be the fact that the underlying platform and operating systems are not identical. However, In my opinion it is Java´s task to overcome these problems. Hopefully Java will mature and the behavior demonstrated on one platform will be the same for all platforms (i.e. it is good to be sure of that your applet will not crash on a specific platform or in a specific browser. It should suffice to test the applet on one platform and on one browser.)

8.4 The Development Tool - Symantec Café

At the time of writing there are many IDDE:s on the market: Microsoft has released J++, Symantec has released a new product called Visual Café, Jfactory from Rouge Wave and so on. In this section the IDDE that the author has used, Symantec Café, will be briefly described, together with some personal reflections (cf. Dr. Dobbs Journal August 1996 [A3] for a more extensive review of Symantec Café).

8.4.1 Description Of The Symantec IDDE

Café supplies most of the features that are common in today's IDE:s (Integrated Development Environment): project management, wizards, compilers, class browsers, graphical debuggers, color-coded source code etc. A resource editor called Café Studio is also included, which is a visual tool that lets you create Java forms by using drag-and-drop on visual controls (i.e. buttons, choice lists, scrollbars and so on). It is also possible to import existing Windows resource scripts into Studio and convert them into Java source code.

Project Management

The project management feature allows the management of a set of source files. A project can be built into a program. The project manager lets you choose if it is an applet or application is to be built. What compiler, that is to be used, the Sun Java Compiler or the Symantec one (yes, they supply their own that is faster than Sun's), can also be chosen together with some other features.

Views and Workspaces

The Views and Workspaces allows you to look at your program from different perspectives. A view is essentially a window, whereas a Workspace is a collection of views. Altogether, there are nine different views: Source Editing, Project, Output, Call Chain, Data/Object, Breakpoints, Thread Debugging, Class Editor, and Hierarchy Editor.

Most of these views are self explanatory. The Class Editor is a class browser that lets you look at the classes of a project. The members of a class as well as the source code for that member can also be viewed. The window is divided into three panes. The Hierarchy editor provides the same functionality, but instead the graphical representation of the class relations is given. The classes and members can be edited, new classes and methods can be added and so on.

Café allows up to five workspaces to be defined. Initially, four predefined workspaces are provided Editing, Browsing, Debugging and Output. Naturally, the views contained in each workspace can be defined and edited.

8.4.2 Personal Reflections On Using Symantec Café

I found Café simple to work with, almost everything is self-explanatory. It takes a little time before you get a grip of things, how projects work and so on, but after that it just to get on with it. I especially appreciated the class editor, when my project grew in size. The class editor made it ease to browse through the system and get an overview of it.

There were also some negative things, among them an irritating bug that occurred once in a while having to do with error messages. When a compile error is detected it is shown in the output window, and then the line containing the error can be double clicked and you come directly to the incorrect line in the source code. This worked fine most of the time, but after error correction it was sometimes impossible to save the source file and the whole IDDE had to be restarted. Also, I did not find the Studio very helpful. The Studio is supposed to generate the Java source code for the forms

that you graphically create, but if you did something wrong there was no Cancel button and you had to manually remove the source code files.

To sum things up, on the whole I liked working with Symantec Café.

9. Conclusions

This thesis has shown that it is possible to develop a prototype resembling a mature application, with the use of state-of-the-art object oriented design techniques. The prototype that has been the result of the work enables control pictures to be viewed over the corporate intranet. Control pictures can also be designed in the editor window of the prototype. The created control pictures generate their own Java code or they can be parsed. The control pictures get updates from a real-time database, containing information about the controlled system.

It is also possible and easy to extend the prototype to contain new types of picture elements. One possible extension might be a picture element that is a picture, i.e. the mapping of a picture onto the coordinate space of another picture.

On the whole the Java language and the development environments used have lived up to what they promised. But, it has also been shown that the Java language and the available frameworks do not suffice to write a full-blown professional software product. But, this could soon change, as many new API:s are being released or is soon going to be released at the time of writing. If the cooperate strategy is to be on the development edge, it might be a good idea, to start planning those full-blown software projects in Java today.

The whole idea of supplying advanced applications over an Intranet, that seems plausible to do bearing the example of the developed prototype in mind, can change the view of how program maintenance and distribution is to be done. The whole concept of an intranet will fundamentally change many values in this business.

References and Bibliography

BOOKS

[1]

J. Gosling and H. McGilton (1995) *The Java Language: A White Paper and The Java Language Environment: A White Paper*, Sun Microsystems Computer Company, Mountain View, CA, Edition May 1995 (available at http://java.sun.com/whitePapers/)

[2]

G. Cornell and C. S. Hostmann (1996) Core Java, SunSoft, Sun Microsystems Computer Company, Mountain View, CA.

[3]

Rumbaugh et al(1991) Object Oriented Modeling and Design. Prentice-Hall International 1991.

[4]

ENEA AB, swedish computer consulting firm, Course material on the OMT-method.

ARTICLES

[A1]

1974. Hoare, C.A.R. Monitors: An Operating System Structuring Concept, Comm. ACM 17, 10:549-557 October.

[A2]

JavaWorld(October 1996), available at [W4].

[A3]

Dr. Dobbs Journal (August 1996).

WEB

[W1]

http://www.javasoft.com:80/doc/white_papers.html

[W2]

http://www.javasoft.com:80/doc/Overviews/java/

[W3]

http://www-und.ida.liu.se/~pum17

[W4]
http://www.javasoft.com

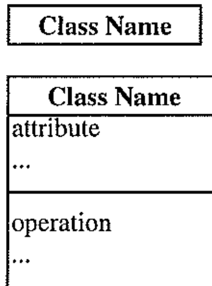
[W5]
http://www.javasoft.com/products/JDK/1.1/designspecs/index.html

[W6]
http://www.digitalfocus.com/digitalfocus/faq/howdoi.html

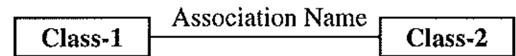
Appendix A

OMT-notation used in this paper:

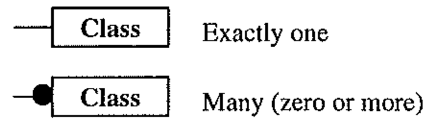
Class:



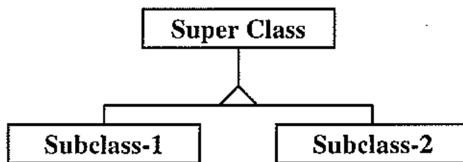
Association:



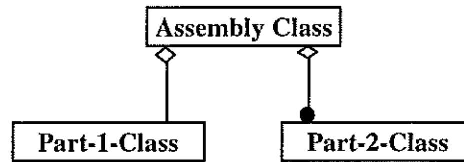
Multiplicity of Associations:



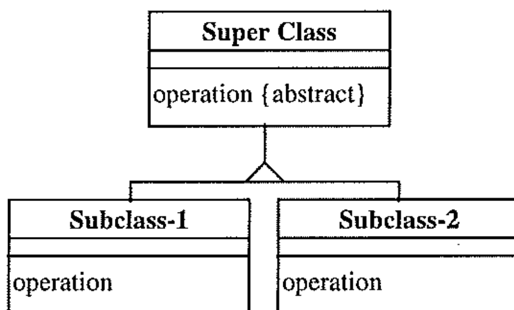
Generalization (Inheritence):



Aggregation:



Abstract Operation:



Derived Class: