CODEN: LUTFD2/(TFRT-5435)/1–37/(1991)

A Process Knowledge Base Browser

Marie Andersson

Department of Automatic Control
Lund Institute of Technology
March 1991

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

Document name
Master Thesis

Date of issue
March 1991

Document Number
CODEN: LUTFD2/(TFRT-5435)/1-37/(1991)

Author(s)
Marie Andersson

Supervisor
Nick Hoggard, ABB, Karl-Erik Årzén, LTH

Sponsoring organisation

Title and subtitle
A Process Knowledge Base Browser

Abstract

This report is the documentation of a project called "A Process Knowledge Base Browser", that has been performed as a Master Thesis in Computer Engineering at the Lund Institute of Technology.

The purpose of the project is to develop a graphical browser to an object-oriented data base containing various types of information about a process plant and its components. The structure and contents of the knowledge base were developed prior to the project by ABB.

The information in the knowledge base is divided into two large groups: classes and instances of these classes. Each of these groups is displayed in a separate window. The classes and the instances share the same basic structure. This made it possible to make the two windows almost similar, which in turn makes the browser easier to use. The class hierarchy graph is displayed in a third window on the screen. The completed graphical browser consists of three different windows, which together can show the user all the information currently in the knowledge base.

The browser is implemented in C++ using the object-oriented graphical user interface InterViews.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
37

Recipient's notes

Security classification

Contents

Chapter 1

Introduction

This report is a documentation of a project called *A process knowledge base browser*, which I am doing as a Master Thesis in Computer Engineering, at Lund Institute of Technology. The project has been done at ABB Corporate Research, Ideon, Lund, where I also have my first instructor Nick Hoggard. ABB cooperates with the Department of Automatic Control with projects similar to this one, hence I got my second instructor there, Karl-Erik Årzén.

Prior to this Master's Thesis project, a prototype knowledge base had been constructed by Nick Hoggard. The purpose of this knowledge base was to store all the information about a process plant, for example functional, topological, geographical information etc. Also defined were some procedures for searching through the knowledge base. My task was to show the information about the plant on the screen using the procedures mentioned above and the knowledge base. Displaying the information should be done in a way that tell the users as much as possible about the plant. Showing all the information simultaneously on the screen was out of the question for mainly two reasons. First a complete knowledge base is very, very large and second if everything is shown at the same time, it would be impossible to subtract any useful information from the "mess" on the screen.

In order to be useful the browser should be well-structured, logical and easy to use. All the information in the knowledge base was divided into two large groups: classes and instances of these classes. Each of these groups is displayed in a separate window. The classes and the instances share the same basic structure. This made it possible to make the two windows almost alike, which in turn makes the browser easier to use. Finally the class hierarchy graph is displayed in a third window on the screen. The completed graphical browser consists of three different windows, which together can show the user all the information currently in the knowledge base. The browser is implemented in C++ using the object-oriented graphical user interface called InterViews.

Chapter 2 shortly describes the whole Browser project. The concepts of the knowledge base are explained and described in Chapter 3, and after that, in Chapter 4, the implementation of the knowledge base is described. Chapter 5 is a thorough description of the graphical browser and its functions and Chapter 6 is about the implementation. Finally conclusions are found in Chapter 7.

Chapter 2

The Browser project

2.1 Introduction

The Browser project is being performed at ABB Corporate Research in Lund. The purpose of the project is to investigate the possibility of implementing an object-oriented knowledge base containing a variety of different information and knowledge about the plant. Different kinds of information might include *geographical* information, e.g. 3-D descriptions, *topological* descriptions of the plant, e.g. process and electric schematics, *functional* plant descriptions that describes the goals and abstract functions of the plant, component information, control logic, *textual* descriptions, e.g. installation descriptions and operator manuals, etc.

In a large knowledge base of this kind it is important to have structuring mechanisms that matches the plant and how information about the plant is stored. The most important of these mechanisms is the *object*, which can represent some physical entity in the plant. Objects representing larger parts of the plants are typically composite, i.e./ they are composed of other objects. It is often the case that a physical object needs to be described from multiple points of view at the same time. In order to handle this the multi-view object concept is used, which will be described later.

Plant knowledge bases have multiple applications. Ideally they should follow the life-cycle of the plant, i.e. they should be built-up during design and updated during production, operation and maintenance. Examples of knowledge base approaches to engineering are the Steerbear project at Kockums Computer Systems or the CALS (Computer Aided Logistics and Support) project initiated by the US Department of Defence. ABB cooperates with the Department of Automatic Control at Lund University of Technology in an IT4 project called "Knowledge-based Real-time Control Systems". In this project a multi-view knowledge base is specified that also includes the on-line computer control system.

There are three larger parts in this project the knowledge base prototype, the load/save tool and the browser, which is the part that I am implementing. Connections between these parts are shown in Figure 2.1. The load/save routines are used to collect and save information in the knowledge base to/from a disk file.

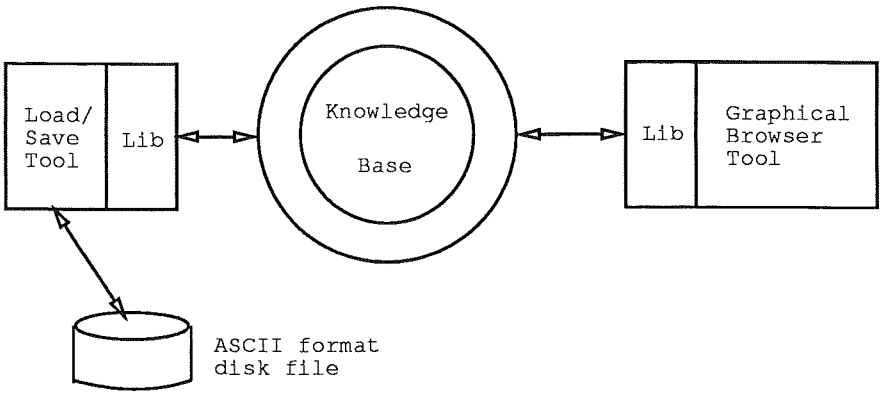


Figure 2.1: Connections between the three larger parts of the project, the knowledge base, the load/save toolkit and the browser.

2.2 Knowledge base

The knowledge base contains all information necessary to represent a process plant. This knowledge is divided up into *objects*. In a model of, e.g., a process plant these objects could correspond to process components such as pumps, valves, switches etc., process units or the entire plant.

One of the reasons for using this kind of knowledge base, where everything is divided into objects, is that the source code can be reduced with up to two thirds. The reduction is due to the fact that the data structure can look the same in the knowledge base as in the application. An object contains data that are logically connected. A user propably wants to read in a whole object at the same time, which can be done much faster in this kind of knowledge base than in a conventional data base.

The following characteristics are implemented in the knowledge base and described more detailed in Chapter 3 and 4:

- Classes and instances of objects
- Class to class inheritance
- Multi-view objects
- Composite objects
- Objects that are connected together using the concepts "connection points" and "connections"

2.3 Load/Save tool

Load/save tools are used for saving the information in the knowledge base to disk file and loading from disk file back to either files specified by the programmer or standard files. The

contents of the disk file is in *ASCII-format*, which means that the knowledge base does not have to be defined with C++ code, it can also be defined by editing the ASCII file.

2.4 Browser

Reading the ASCII representation of the knowledge base will not give much knowledge about the process plant, even if it contains all the information. The graphical browser is supposed to display the information, currently in the knowledge base, in such a way that it will be easier to understand compared to reading directly from the knowledge base.

Objects in the knowledge base have a lot of information connected to them. Each object usually contains information about what kind of object it is, the name of it, what makes this object special compared to other ones and so on. Displayed from the beginning in the browser are only the names of all objects. Using the mouse the user selects any object interesting enough to know more about. The browser then responds to that selection, finds out as much about it as possible, shows the result on the screen in a structured way and waits for new selections.

C++ and *InterViews*[1] have been used to implement the browser. For more information see [C++], [IV1], [IV2].

[1] InterViews is an object-oriented graphical user interface from Stanford University.

Chapter 3

Knowledge base concepts

3.1 Introduction

In this chapter all concepts concerning the knowledge base will be explained. Relationships between different concepts are also described here. To make this chapter somewhat easier to understand I will use the same example, Ex1, troughout the chapter.

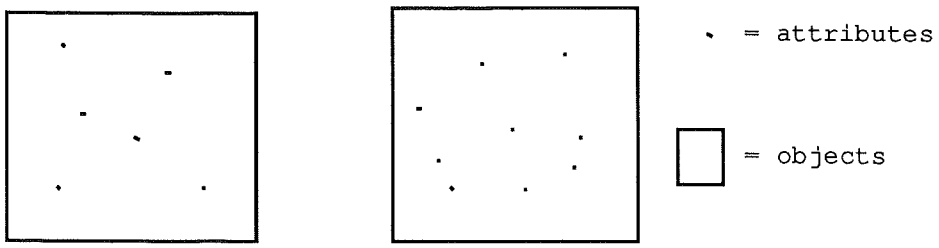
Here is Ex1:

```
-----------------------------------------------------------------------
OBJECT Steritherm
   LIST (consists-of relation) Electrical system
      STRING child entity = Electrical system
      STRING childview = topological view
   LIST (consists-of relation) Control system
      STRING child entity = Control system
      STRING childview = topological view
   LIST (consists-of relation) Steam system
      STRING child entity = Steam system
      STRING childview = topological view
   LIST (consists-of relation) Pneumatic system
      STRING child entity = Pneumatic system
      STRING childview = topological view
   LIST (consists-of relation) Warm water system
      STRING child entity = Warm water system
      STRING childview = topological view
   LIST (consists-of relation) Cold water system
      STRING child entity = Cold water system
      STRING childview = topological view
   LIST (consists-of relation) Main product system
      STRING child entity = Main product system
      STRING childview = topological view
   LIST (photograph class) Steritherm photo
      INT photograph number = 12344
```

```
LIST (view) topological view
   STRING Electrical system:topological view
   STRING Control system:topological view
   STRING Steam system:topological view
   STRING Pneumatic system:topological view
   STRING Warm water system:topological view
   STRING Cold water system:topological view
   STRING Main product system:topological view
LIST (view) geographical view
   STRING Steritherm photo
LIST (view) functional view
```

3.2 Objects, Attributes

The knowledge base consists of a list of objects, which in this case can be both classes and instances of classes. Each of the objects has a number of attributes. Attributes can be simple things like integers or strings, or more complex data structures.



In this knowledge base there are four different kinds of objects:

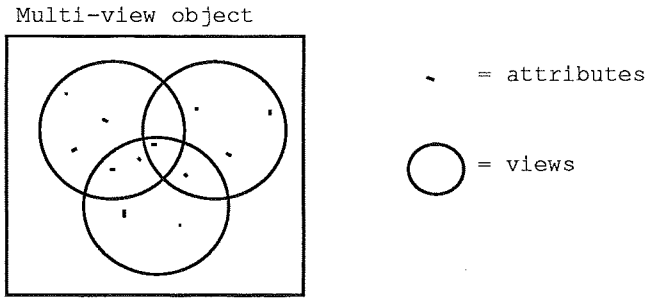
1. Multi-view objects
2. Connection objects
3. Class definition objects
4. Composite multi-view objects

3.2.1 Multi-view objects

Multi-view objects are used to describe objects that requires to be described in different contexts with different attributes and internal structure. At the same time, in the different contexts, though they need to be described as one unified object. To be able to divide the object, into pieces matching the special purposes, an attribute called "view" is used. The topological, the geographical and the functional view are the most general ones. Topological views shows, e.g., designs of parts of objects, circuit diagrams for electrical systems and so on. Geographical views can be photos, drawings etc. that show how something really looks, and finally the functional view, which explains how things work and what they are

supposed to do. In Ex1 at the beginning of this chapter the views are implemented as LIST-attributes. In the topological view there are diagrams of the electrical system, the control system, the steam system and so on. The geographical view is a photo of the Steritherm object. The functional view does not give any information at all at the moment.

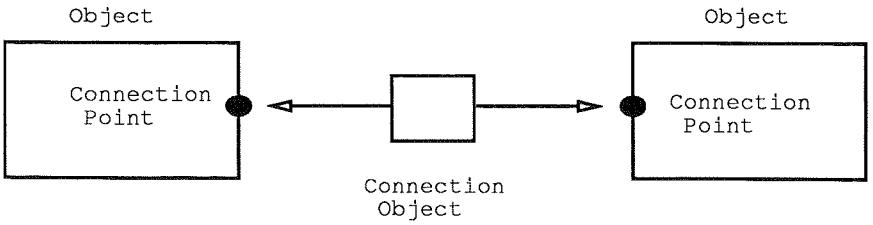
As shown in the figure below a multi-view object can have more than one view. Any attribute can also be included in more than one view.



If you look at the class definition for a "multi-view object", it is a *parent class.* And as with all other base classes new classes can be derived from them. "Pump class", for example, may be derived from a multi-view object. Derived classes inherit all attributes, including view attributes, from their "parent classes", who in turn may be derived classes with inherited attributes. The newly constructed classes then define and add their own characteristic attributes.

3.2.2 Connection objects

Connection objects are special objects that connects two other objects via "connection points".



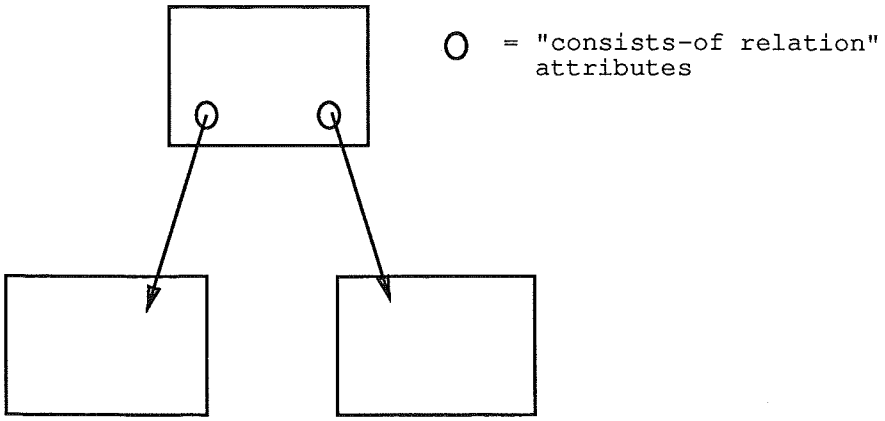
3.2.3 Class definition objects

All objects that have the same attributes and behaviour can be grouped together. A "class definition object" is an object that defines the structure of the group of objects. Class definition objects are further described in the following section.

3.2.4 Composite multi-view object

A composite multi-view object is an object that has an internal structure consisting of interconnected objects representing subparts of the superior object. A subpart of a com-

posite object does not have to be another whole object, it could also be one or more views of an object. Ex1 contains LIST attributes called "consists-of relations". These attributes show the subparts and also what view/views of those subparts they concern.



3.3 Classes, instances and inheritance

A class is a group of objects that share the same attributes and behaviour. As there are different kinds of objects, there are also different kinds of classes:

- classes describing an object
- classes describing a composite object
- classes describing an attribute

Objects describing a unique member of some object class are called instances.

A class definition object is an object describing the structure of a class. If a new class definition is created from any already existing class definition, a process known as "inheritance", the new class will consist of all attributes included in the existing class. In the new class it is forbidden to remove any of these inherited attributes but it is allowed to add new ones.

Created classes are called child classes and those they were created from are parent classes. Multiple inheritance means the ability to inherit attributes from more than one parent class. This is allowed here.

Chapter 4

The Knowledge base implementation

4.1 Introduction

In this chapter the contents and structure of the knowledge base will be described. It will also be described how all the concepts, explained in the last chapter, are implemented in the knowledge base. In Section 4.2 the structures of objects and attributes are described and in Section 4.3 the structures of different classes are explained. The Steritherm-example I used in Chapter 3, Ex1, is sometimes used here too.

4.2 Objects, Attributes

The knowledge base consists of a list of objects.

```
-----------------------------------------------------
(OBJECT
OBJECT
OBJECT
...)
-----------------------------------------------------
```

Objects consists of lists of attributes.

```
-----------------------------------------------------
(OBJECT
  ( INT
   STRING)
OBJECT
OBJECT
   (STRING
   LIST
      STRING
      INT)
OBJECT
...)
-----------------------------------------------------
```

An *object* has a list of what is called *attributes*. There can be three different kinds of attributes in an object, INT, STRING and LIST. The third type, LIST, makes it possible to construct composite attributes, which are defined in classes for attributes. As mentioned before this knowledge base contains four kinds of objects. Their implementation will be described in the following sections.

4.2.1 Multi-view objects

Multi-view objects have one or more attributes called "views". The view-attribute contains a LIST of the attributes that belongs to a particular view. And as mentioned before one attribute can appear in more than one view. Ex1 shows the most common views, the topological, the geographical and the functional view.

4.2.2 Connection objects

An object may contain attributes named "connection points", which are defined by the class definition "connection point class". As can be heard by the name of them, "connection points" are used to connect two multi-view objects together.

Four attributes are used in each connection object and, in order, they are:

- Name of first object
- Name of connection point on first object
- Name of second object
- Name of connection point on second object

Ex 2 below shows how one of the connections between the switch K2 and the pump M2. This connection is connected to a connection point called 6 on the switch and a connection point called C on the pump. There are propably more connections than this one between switch K2 and pump M2, but one connection object only defines ONE connection.

Ex2:

```
-----------------------------------------------------------------
OBJECT (connection class for pump-switch group, 3) K2:6,M2:C
   STRING object 1 name = Switch K2
   STRING connection point 1 name = 6
   STRING object 2 name = Pump M2
   STRING connection point 2 name = C
-----------------------------------------------------------------
```

The class "connection class" defines how a connection point is supposed to be implemented.

4.2.3 Class definition objects

All information about a class is gathered together in its class definition, which is an object. The contents of it is shown and described as follows:

- a LIST of ancestors: this is a list of parent classes that this class has been **directly** derived from.
- a LIST of attributes: when an instance of this class is created, these attributes and inherited attributes should be included.
- a LIST of class component entities: if an instance of this class is made, all the objects in this list are automatically created. For example, if an instance of a pump-switch group class is created, then one object of the type pump class is created, one of the type switch class and three of the type connection class for pump-switch group. Furthermore "consists-of relation" attributes are added, one for each of the above objects, to the new composite object.

Ex3:

```
-----------------------------------------------------------------
OBJECT (class definition) pump-switch group class
   LIST (class ancestors)
      STRING multi-view object
   LIST (class attributes)
   LIST (class component entities)
      STRING pumpname = pump class
      STRING switchname = switch class
      STRING connect1name = connection class for p-s group, 1
      STRING connect2name = connection class for p-s group, 2
      STRING connect3name = connection class for p-s group, 3
-----------------------------------------------------------------
```

The list of class attributes shows that instances created from this class will have no attributes inherited. Next list, the list of component entities, shows that instances created from this class will be built up by one pump, one switch and three connections between them. It should be mentioned that if this list is empty then the created instance of the class is not a composite object. The example below shows an instance of the pump-switch group class.

```
-----------------------------------------------------------------
OBJECT (pump-switch group class) PSG 2
   LIST (consists-of relation) Pump M3
      STRING child entity = Pump M3
      STRING childview =
   LIST (consists-of relation) Switch K3
      STRING child entity = Switch K3
      STRING childview =
   LIST (consists-of relation) K3:2,M3:A
      STRING child entity = K3:2,M3:A
      STRING childview =
   LIST (consists-of relation) K3:4,M3:B
      STRING child entity = K3:4,M3:B
      STRING childview =
   LIST (consists-of relation) K3:6,M3:C
      STRING child entity = K3:6,M3:C
      STRING childview =
-----------------------------------------------------------------
```

As mentioned one "consists-of relation"-attribute is created for each component in the component entities list, which is in the class definition for a pump-switch group. The childviews can be named but in this example they are not.

4.2.4 Composite multi-view objects

A composite multi-view object is created from a class with a component entities-list that is not empty. These kinds of objects have a characteristic attribute called "consists-of relation". Composite objects have one or more components, all of which are refered to by the "consists-of relation" attributes, which consists of two attributes, the name of the component and what view of that component it refers to.

```
-----------------------------------------------------------------
OBJECT (multi-view object) Main product system
   LIST (view) topologocal view
   LIST (view) geographical view
   LIST (view) functional view
   LIST (consists-of relation) Pump M2:topological view
      STRING child entity = Pump M2
      STRING childview = topological view
   LIST (consists-of relation) Pump M2:geographical view
      STRING child entity = Pump M2
      STRING childview = geographical view
-----------------------------------------------------------------
```

4.3 Classes

As mentioned before there are different kinds of classes. These will be described thoroughly in the following sections.

4.3.1 Classes for objects

The "multi-view object"-class and the "pump-class" for example, are two classes describing objects.

```
-----------------------------------------------------------------
OBJECT multi-view object
   class ancestors
   class attributes
   class component entities
-----------------------------------------------------------------

-----------------------------------------------------------------
OBJECT pump class
   class ancestors
      multi-view object
   class attributes
      ATTRIBUTE
      ATTRIBUTE
      ATTRIBUTE
      ...
   class component entities
-----------------------------------------------------------------
```

Their most important similarity is that in both classes the list of component entities is empty. As can be seen in the above picture, the list of ancestors is allowed to be empty, and so is the list of attributes. They can also have one or more attributes, like in the pump class.

4.3.2 Classes for composite objects

In this kind of class the list of component entities is not empty. Composite objects are constructed from other objects, called *components*. The list of component entities consists of name references to other object classes, which put together make the composite object.

An object created from the pump-switch group class, see Ex3, are built up by five components, a switch, a pump and three connections for pump-switch groups. The three connection objects describes how and where the pump and the switch are connected to each other. See Ex2.

4.3.3 Classes for attributes

This type of class defines how a certain kind of attribute should be implemented. As an example I chose "electrical connection point class" :

```
-----------------------------------------------------------------
OBJECT electrical connection point class
   class ancestors
      connection point class
   class attributes
      connection type = electrical power phase
      phase = put A,B,C or ground here
   class component entities
-----------------------------------------------------------------
```

The list of attributes in this class shows that an instance of electrical connection point class will have one attribute defining what kind of connection type it is and another attribute defining the phase.

Chapter 5

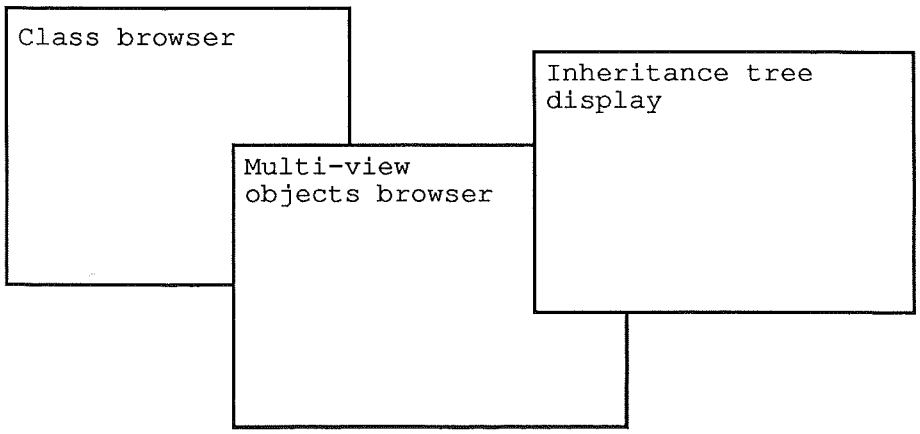
The Browser

5.1 Introduction

The browser shows the contents of the knowledge base to the user in a graphical way. As the knowledge base can be very large it is impossible to show everything at the same time. Therefore it was decided to show only the names of the classes and objects. If the user then wants to know more about an object for example, he or she selects that object with the mouse. The browser gets the message, searches through the knowledge base for all information about the selection and shows it on the screen.

The browser is partitioned into three parts each having its own window on the screen. These parts are:

- Class browser
- Multi-view object browser
- Inheritance tree display



Two of them, the class browser and the multi-view object browser, interact. Hence, if the user makes certain steps in one of the windows, the other one will respond.

Classes are displayed in the class browser and multi-view objects in the multi-view object browser. Connection objects are not shown explicitly in any window. Connections are instead shown in the "connection points" subwindow. In all of the windows a mouse is used for all kinds of operations. Most of the classes in the knowledge base are child classes, this fact made it interesting to draw the whole class inheritance tree once and for all in a separate window, the Inheritance tree display. In this window the mouse is only used for scrolling and zooming.

5.2 Multi-view object browser

5.2.1 Introduction

All objects that are created from multi-view object class or any of its child classes are to be shown in this window called *Multi-view object browser*. What the final result ought to look like, and precisely what was to be displayed I did not know from the beginning. This window was created bit by bit, and has been changed a number of times since the start of the project.

To write the program InterViews was used. Problem number one that arised was the question of which of the InterViews classes were to be chosen to implement the smaller subwindows inside the multi-view object browser. Could an already existing class be used, or did it have to be child classes of such. Final decisions are discussed in a separate section at the end.

Figure 5.1 shows the multi-view objects browser as it looked when it finally was completed. Dividing this big window into smaller subwindows makes it more easy to describe and understand. The following sections each take care of the description of a subwindow.

5.2.2 Multi-view objects subwindow

The purpose of this subwindow is to show the user all multi-view objects currently in the knowledge base, so that users by themselves may choose what object is interesting. Only the names of the objects are displayed. When starting up the program this subwindow will be the only one in multi-view objects browser containing anything at all. See Figure 5.2.

5.2.3 Selected multi-view object subwindow

Selections of multi-view objects can be carried out in a number of different ways. Hence chosen objects may not be visible in the Multi-view objects subwindow. As it often is desirable to know the name of the last choice a "Selected multi-view object" subwindow was added to the Multi-view object Browser. In this subwindow the user can always see what the last selection was. An example is shown in Figure 5.3.

5.2.4 Class subwindow

All multi-view objects are instances derived from some class in the class browser. Therefore it could be of interest to show to the user what class the chosen object has been derived from. This is done in class subwindow, and the class that is displayed is the one that the object is an instance of. Figure 5.4 shows an example.

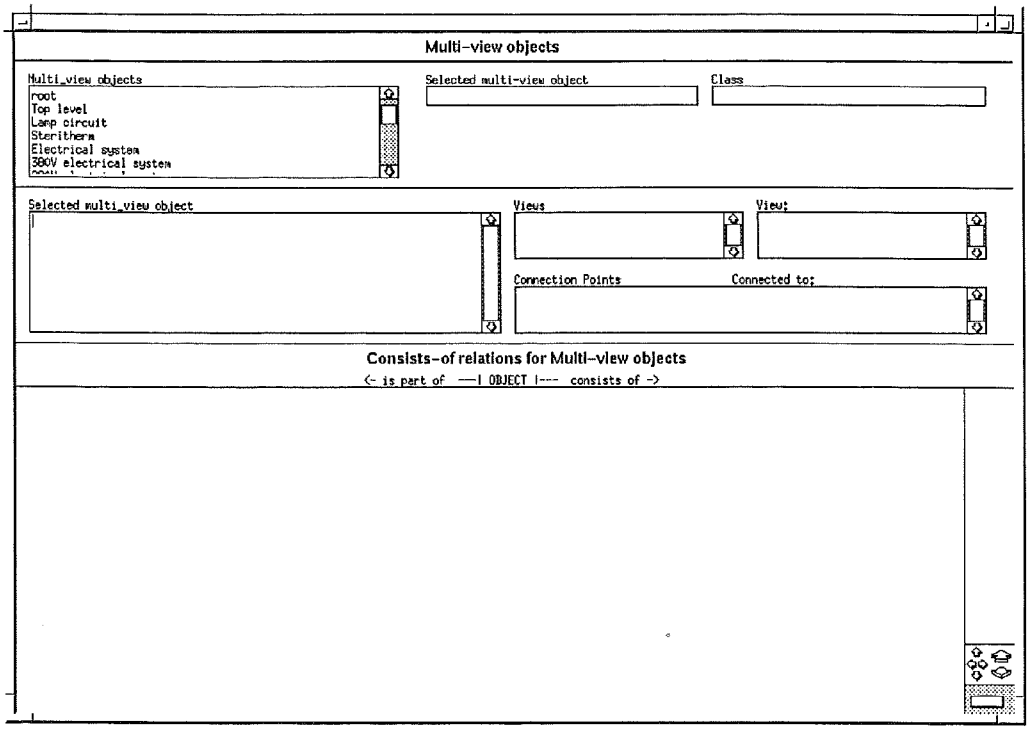


Figure 5.1: Multi-view objects browsers final look.

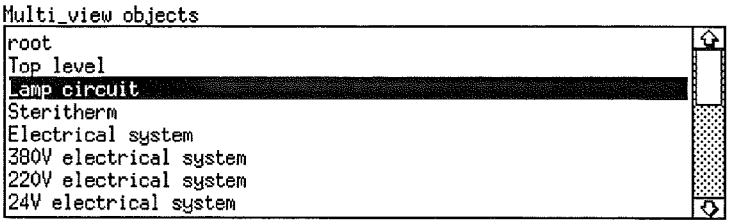


Figure 5.2: Multi-view objects subwindow when multi-view object "Lamp circuit" has been chosen

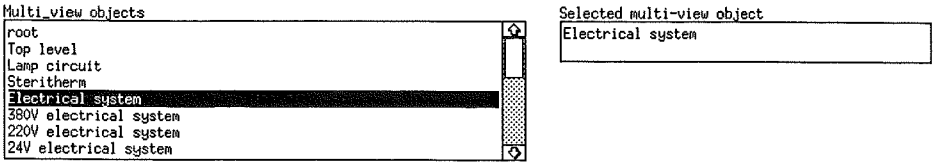


Figure 5.3: The purpose of selected multi-view object subwindow.

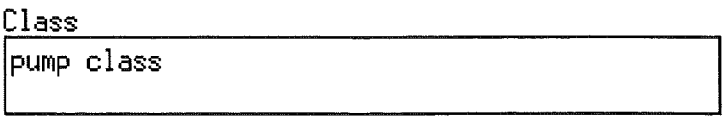


Figure 5.4: Class subwindow when object "Pump M2" has been chosen.

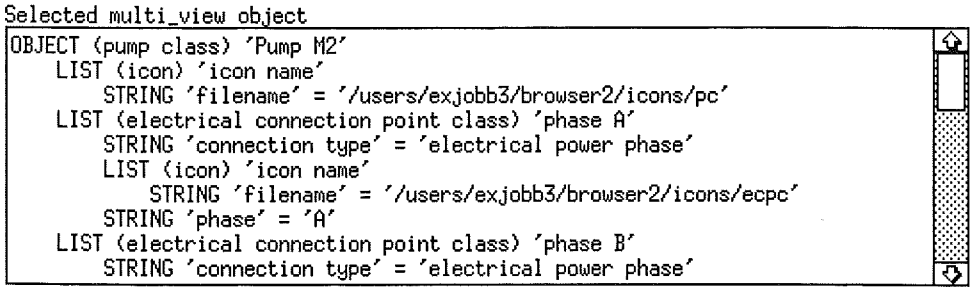


Figure 5.5: This selected multi-view object subwindow shows all information about the chosen object.

5.2.5 Selected multi-view object subwindow II

Selected multi-view object subwindow II shows, as well as the selected multi-view object subwindow we earlier discussed, the multi-view object last chosen. The difference between these subwindows is that in subwindow II, all information about it is shown as it is represented in the knowledge base (although formatted a bit to make it more readable). See Figure 5.5.

5.2.6 Views and View: subwindows

In an earlier chapter in this report views are described as parts of an object. As can be heard from the word *multi-view object* one object may have more than one view. If a selected object has one or more views their names are displayed in Views subwindow as shown in Figure 5.6. Select one of the views in View subwindow and all parts of the object the chosen view contains is shown in Views: subwindow, see Figure 5.6. Just as before all selections are marked using reversed colours.

5.2.7 Connection points subwindow

When an object is chosen the browser checks if the object has any connection points. If so, the names of all connection points are written out in Connection points subwindow. It should be mentioned though that all connection points are shown, not depending of what view it belongs to. If the connections of the selected objects are connected to anything then the name of that object will be written out, see Figure 5.7. In Connection points

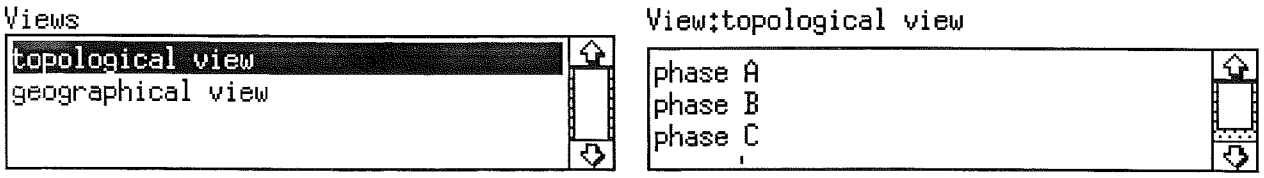


Figure 5.6: Views and View: subwindows.

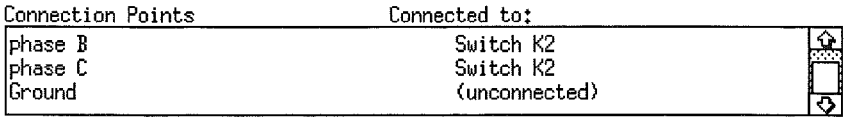


Figure 5.7: Connection points subwindow.

subwindow it is possible to make selections. Not of the connection points themselves but of whatever object it is connected to. The Multi-view object Browser then shows the selected object. Reversed colours are used to point out what choice was just made.

Connection points is something that could be shown graphically, but in this implementation it is not.

5.2.8 Consists-of relations for multi-view objects subwindow

Some objects may be put together to make a new object, a composite object. A great deal of all objects in Multi-view objects browser are either composite objects or parts of composite objects. When an object is chosen in the Multi-view objects browser, the "Consists-of relations for multi-view objects" subwindow will show if the selection is a composite object or is a part of another composite object. Nothing says that it cannot be both.

One great difference between this subwindow and other subwindows is that lines, rectangles, labels etc. are used to display information. In Figure 5.8 the object Steritherm has been selected. As can be seen in 5.8 the selected object is the one in the middle. If the chosen object is a component of other objects, these are drawn to the left (in this case Top level). On the other hand if the selection has got components these are drawn to the right (for example steam system). In this window all components are shown independently of what view they belong to.

All selected objects in this subwindow are surrounded by a light-grey rectangle. It is also possible to select multi-view objects by pressing a button on the mouse when pointing at the object's label. When that has been done all subwindows will be cleared in multi-view objects browser and the newly selected object is displayed instead. In "the consists-of relations for multi-view objects" subwindow, the chosen object will be marked and centered.

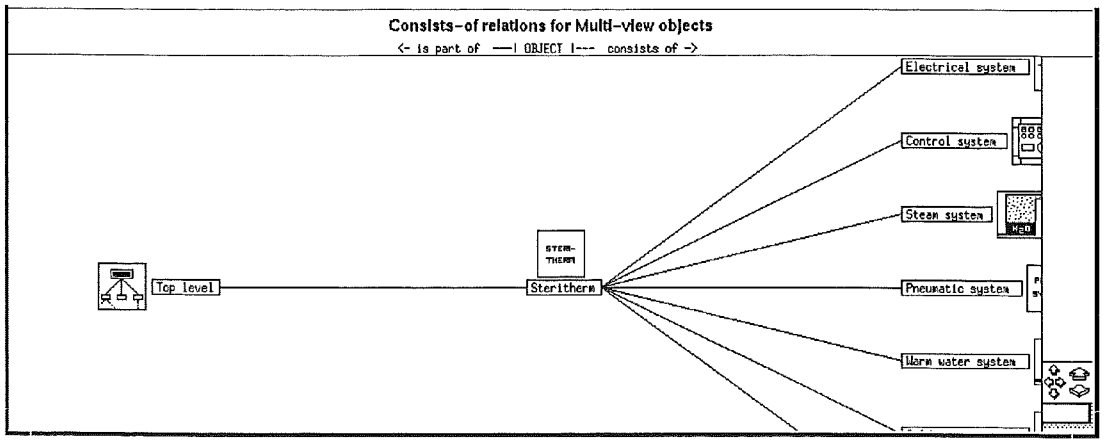


Figure 5.8: Consists-of relations for multi-view objects subwindow.

5.3 Class browser

5.3.1 Introduction

The Class browser window shows the classes defined in the knowledge base. To make the whole browser logical and well-structured this window ought to have the same layout as the Multi-view objects browser. Because of some differencies between multi-view objects and classes it cannot look exactly the same though.

In this window the user is able to make selections that will make the Multi-view object browser window react and show a new object. The Multi-view object browser window can on the other hand never change anything in the class browser window, which means that there is only a one-way connection between class browser and multi-view object browser.

When the class browser was completed, it looked like Figure 5.9. Dividing the window into smaller pieces makes it somewhat easier to describe, hence that has been done.

5.3.2 Classes subwindow

Class definition objects currently in the knowledge base are all displayed in "Classes subwindow" as shown in Figure 5.10. It does not only show multi-view objects classes, it also shows attribute classes, composite object classes and so on. When the program starts this subwindow will be the only one in the class browser containing information.

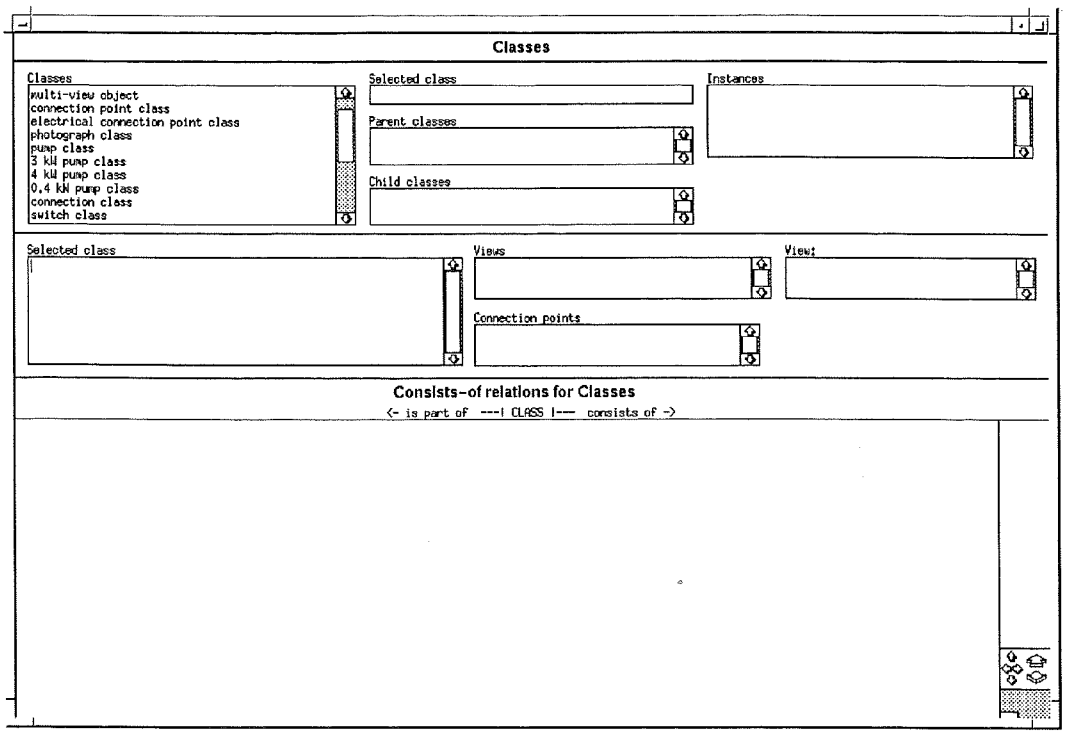


Figure 5.9: The completed class browser

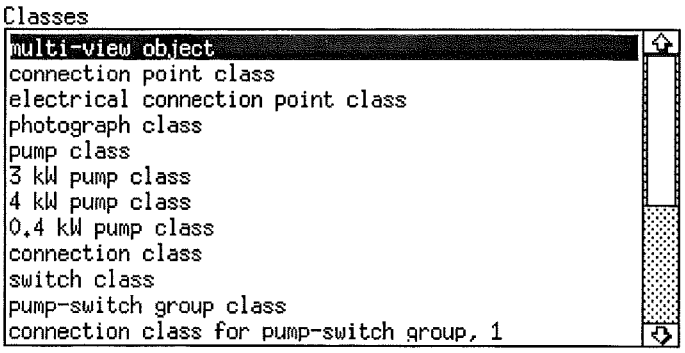


Figure 5.10: Classes subwindow when multi-view object class has been chosen.

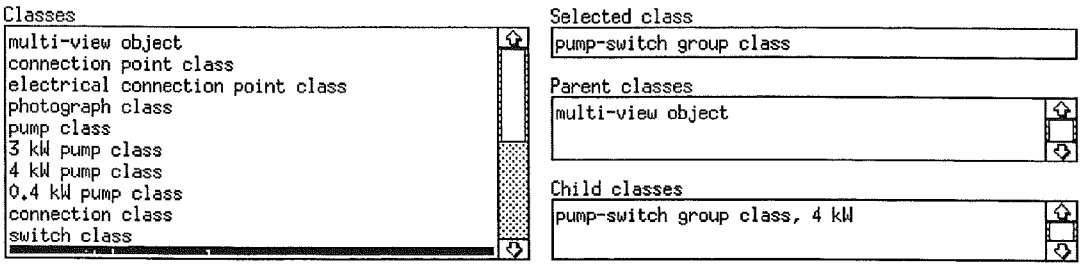


Figure 5.11: Hidden selected class.

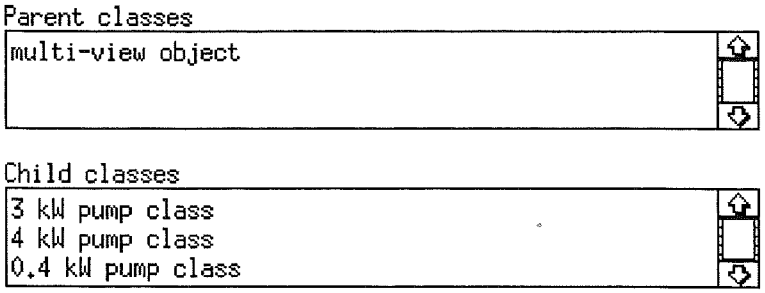


Figure 5.12: Child and parent class subwindows.

5.3.3 Selected class subwindow

The "selected class subwindow", shown in Fig. 5.11, shows the class last selected in "classes subwindow". This might seem a bit unnecessary but it can be useful though. A class can be chosen not only in the "classes subwindow", hence the selected object might not be visible in that subwindow, but it is always visible thanks to "selected class subwindow".

5.3.4 Child and parent class subwindows

Child classes, i.e. those classes directly derived from the chosen one, are displayed in the "child classes subwindow", see Figure 5.12. Furthermore parent classes are shown in the "parent classes subwindow", see also in Figure 5.12. These are classes from which selected classes are derived. Multiple inheritance makes it possible to have more than one class in "parent classes subwindow".

In both subwindows users are able to make selections. All choices give identical results: the choice made is marked in the "classes subwindow".

5.3.5 Instances subwindow

The "instances subwindow" is a subwindow that interacts with the "Multi-view object Browser". Furthermore it shows all instances created from the selected class and from child classes derived from the chosen class. In Figure 5.13 multi-view object class has been

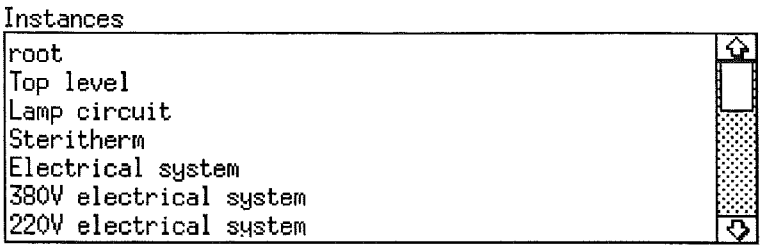


Figure 5.13: Instances subwindow when multi-view object class has been chosen.

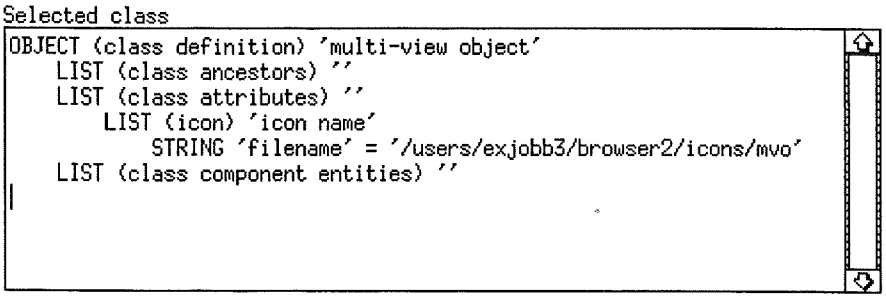


Figure 5.14: Selected class subwindow II when multi-view object class has been chosen.

chosen. Those objects that are multi-view objects can be chosen in the "Multi-view object Browser" too, hence any selection of such an object in "instances subwindow" makes the multi-view object browser react and display whatever choice was made. Also shown in this window are the "connection objects". These can be selected, but nothing will happen though.

5.3.6 Selected class subwindow II

"Selected class subwindow II" shows the selected class just as "selected class" subwindow does. But subwindow II does not write out the name of the class, it displays all information about the chosen class as it is represented in the knowledge base (with some formatting for readability). In the example in Figure 5.14 multi-view object class has been selected.

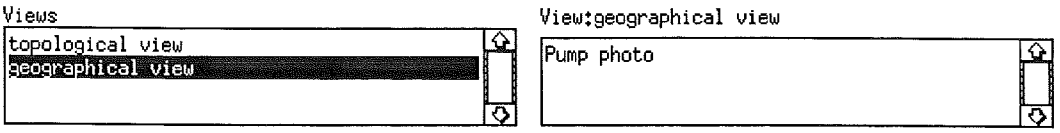


Figure 5.15: Views and View: subwindows.

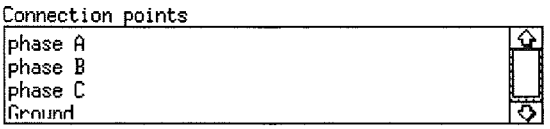


Figure 5.16: Connection points subwindow.

5.3.7 Views and View: subwindows

When a selection of a class is made, all its views will be shown in the "Views" subwindow, see Figure 5.15. Click with the mouse in this subwindow and the chosen view will be displayed in "View:" subwindow as shown in Figure 5.15. Reversed colours are used just as before, to mark the choice in "Views" subwindow. If the class is a definition of an attribute, then this subwindow will be empty, because attributes do not have views.

5.3.8 Connection points subwindow

After any selection of a new class the browser finds out if the chosen object has any connection points. If so the names of all connection points are written out in "Connection points" subwindow. In Figure 5.16 pump class is the chosen class.

5.3.9 Consists-of relations for classes subwindow

There is one subwindow in class browser that does not look like any other subwindow. This particular subwindow is named "Consists-of relations for classes". Rectangles, lines, labels, icons, etc. are used here. Any selected class is shown in the subwindow as in Figure 5.17. If the chosen class is a component of other classes these classes are drawn like in Figure 5.18.

The selected object is marked by a surrounding light-grey rectangle in both of the windows 5.17 and 5.18. Another facility in this subwindow is the ability to select any visible object. Press the left mouse-button when pointing at the objects label and the chosen object will be marked instead. Furthermore the selected object will be centered and its components etc. will also be drawn.

Selections made in this subwindow have similar effects on class browser as any other selection of a class. All subwindows will be cleared, the chosen class will be marked in "classes" subwindow, its name will be written in "selected class" subwindow and so on.

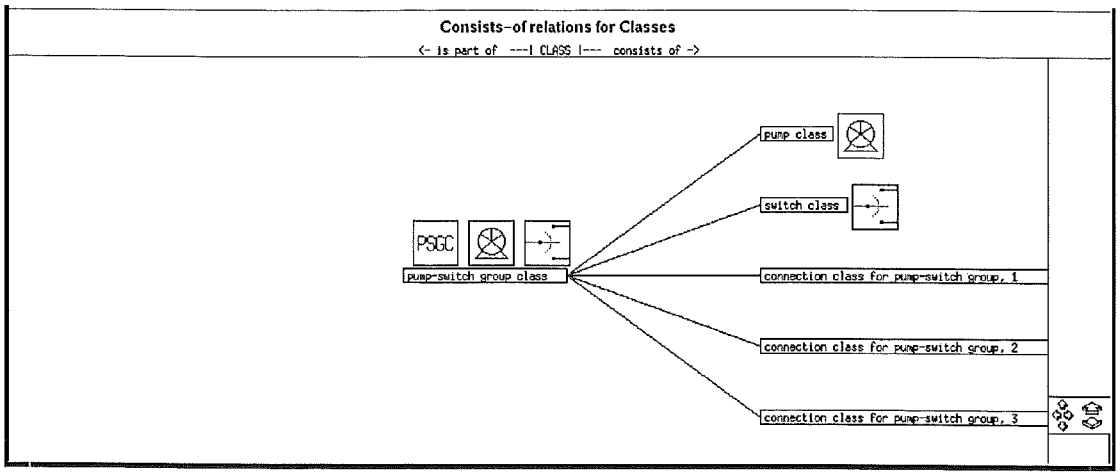


Figure 5.17: Consists-of relations for pump-switch group class.

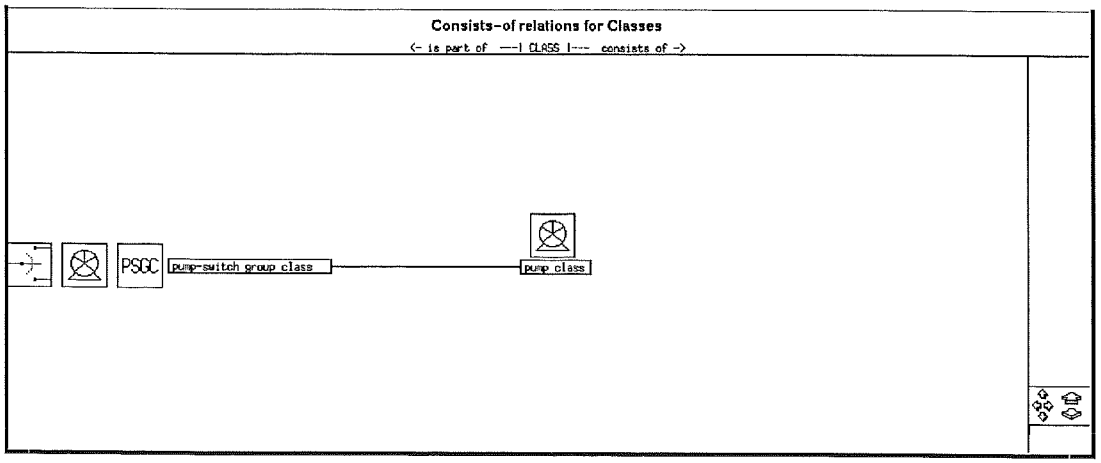


Figure 5.18: A class that is a component of other classes.

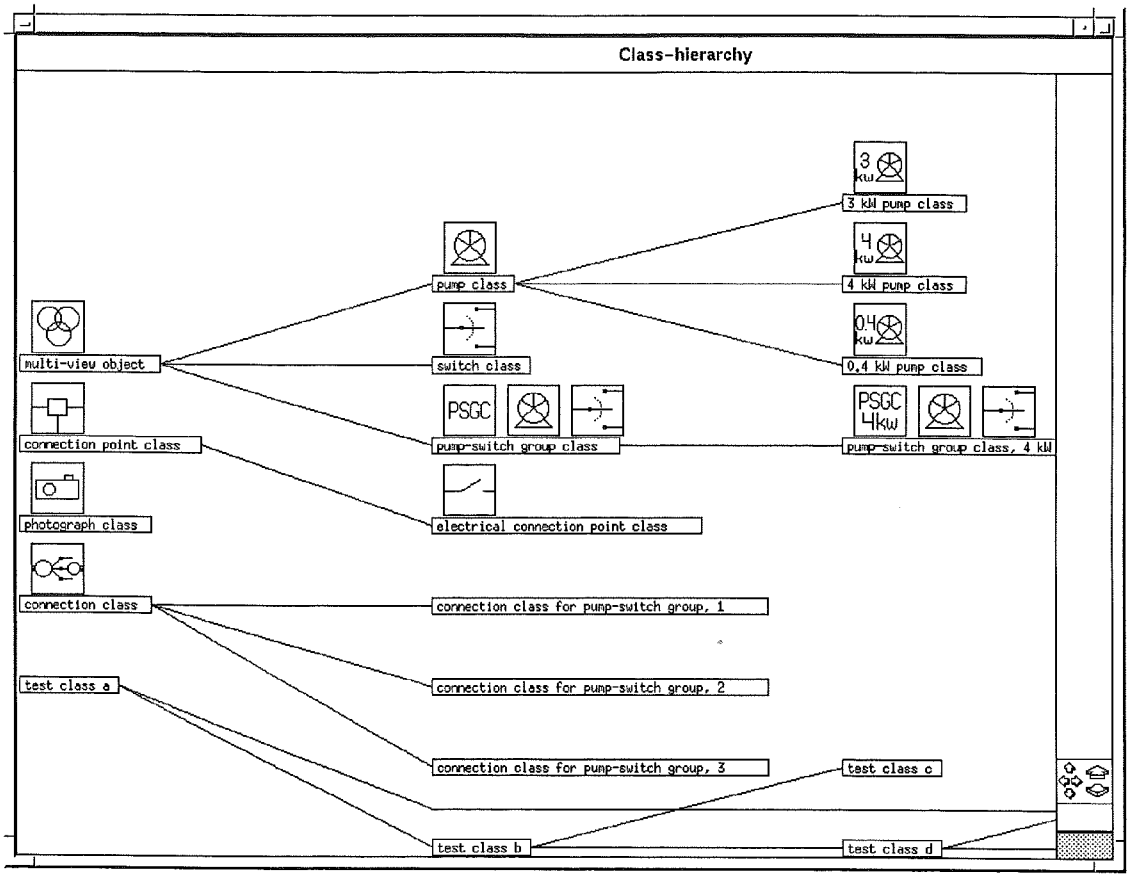


Figure 5.19: The Inheritance tree display.

5.4 Inheritance tree display

When a child class is created from another class it inherits all its characteristics. In most of the cases new ones will be added to the child class. The knowledge base contains a lot of classes most of which are child classes, hence a class inheritance tree could be interesting to any user. As this tree is static, at least from the browser's point of view today, it only has to be drawn once. The Inheritance tree display window, which is automatically iconified at the start of the program, shows the whole tree at the same time. Figure 5.19 shows the final result.

Multiple inheritance makes it possible for a class to inherit from more than one class. This fact became a problem when implementing the procedure drawing the graph. A problem because a child class may be derived from classes which are at different "levels" in the tree. For example "test class f" in Figure 5.19, should have come just after "test class d" to the right. It has got one "parent" at level one and another at level four. The difficulty lays in drawing the lines between them without crossing any other object. Today there already are defined programs that draws graphs, which I maybe could have used, for

more information see [DAG1].

When creating this window a procedure is called that will iconify it directly when the program starts. To make the browser start up a bit faster nothing in this window is composed until the user clicks on the icon with the mouse.

Chapter 6

Implementation of the browser

6.1 Introduction

To implement the browser InterViews, the object-oriented programming language C++ and the UNIX operating system was used. C++ is a programming language derived from C. It has a lot of new features compared to its parent language. Features suiting this project best are those supporting data abstraction (with classes) and object- oriented programming (with virtual functions), which make it possible to define new own types. The second feature mentioned adds the facility to define data types and functions operating on the types. Having learned Pascal as my first programming language I thought it was rather easy to learn C and C++.

InterViews is an object-oriented graphical user interface developed at Stanford University. It is implemented as a library of C++-classes that define common interactive objects and composition strategies. InterViews supports composition of three different object categories of which I have used two, Interactive Objects and Structured Graphics Objects. Some examples of the first category are buttons and menues which are derived from the parent class **Interactor**. Circles and polygons are examples of the second category and they are derived from the parent class **Graphic**.

Almost all my classes in the implementation are derived from an InterViews class, and, if not, I have defined a new class. Each window is implemented as a class and so are each subwindow in the larger windows. Problem number one that was bumped into was the question of which of the InterViews classes were to be chosen to implement the subwindows inside the class browser and multi- view object browser. Could already existing classes be used or did it have to be child classes of these?

The final result included 7800 lines of source code and about 30 different self-defined classes, all of which are subclasses of already defined classes from InterViews. A HP 9000/375 was used for this project.

6.2 StringBrowser-class

Most of the subwindows in the browser have at least one of the following abilities: adding and deleting strings, making selections of those strings. InterViews has a class matching these needs very well, the class *StringBrowser*. Now comes the question of what to use: the

already defined class, or a child class of it? The answer to that question is child classes in all cases. One of the reasons for that is that whenever a selection is made in a subwindow, different actions have to be taken for different subwindows. Therefore the original procedure in StringBrowser taking care of events, (actions such as mouse or keyboard-activity), is changed and implemented in the child classes instead. Another reason is the procedures creating the subwindow, i.e. drawing the rectangle and the name of the subwindow on top of it. This procedure is added to the child class.

6.3 TextEditor-class

In the subwindows displaying how the selected class or object is implemented in the knowledge base, only text is shown and no selections are allowed. *TextEditor* is an InterViews class used to implement these subwindows. A lot of facilities in TextEditor are not used in the windows and some procedures needed in the implementation are missing, see above. Hence, child classes are used again.

6.4 Scrolling

Scrolling is a facility needed in all subwindows except those only displaying one string at a time. Other windows might contain so many strings that if shown at the same time would make the window huge. Therefore scrolling can be done in all those subwindows, either by using the scroller on the right side or by using the mouse inside the subwindows. To implement the scrollers the class Scroller in InterViews was used. Mouse-scrolling was already defined in StringBrowser and TextEditor.

6.5 Graphics, a part of InterViews

Up to this point in the project nothing else has been done than making windows, writing text into them and adding scrollers. Has InterViews got any already defined class for drawing pictures such as lines, rectangles, circles etc ? The answer is yes! *Graphics* is a part of InterViews taking care of all these things. Included in Graphics is a class called *GraphicBlock*. Child classes of this are used to implement the two consists-of relations subwindows in the class browser and multi-view object browser and the Inheritance tree display window. Inside the GraphicBlock class another class is included called *Picture*. An object created from this class is the actual class in which all drawing operations take place. There is one Picture in each of the GraphicBlock child classes. Instead of scrolling with scrollers in this subwindows a panner is used. The panner, created from the InterViews class Panner, controls the Picture in the GraphicBlock. Zooming can also be done with the panner.

6.6 Bitmap-class

The smaller pictures inside the rectangles in the drawing-windows were implemented using the class Bitmap. First I made the pictures using a graphical bitmap editor called Bitmap.

As a result I got a file with a bitmap-matrix. This was then used when creating an icon using the Bitmap-class.

6.7 Experiences of using C++ and InterViews

Most of my earlier experiences of debuggers are that often one has to use them for a rather long time before they really help you find faults in your program. The C++ debugger I used was easy to use and when a fault occured that made the program exit, the debugger showed exactly at what line it went wrong. This facility, among others, made this debugger a good help when searching for errors.

Compiling and linking the C++ program was something I tried to do as seldom as possible. The reason for that was that the linking part took a while to finish. Having a lot of different files that had to be linked together, which I had, did not make it faster.

For more information see [C] and [C++].

I have never used any graphical user interface before InterViews, therefor I have nothing to compare it to. Anyhow I thought that InterViews was easy to use after a while. The manuals to this interface contains descriptions of what the different functions connected to a class do. But how to use them in your own program was not always that simple to find out.

I could not get the Panner, used to control the subwindows containing pictures, to work as it is supposed to. Sometimes when zooming back and forth the pictures are not drawn as they should be, and if the size of the whole subwindow becomes smaller the top of the controllers are cut off. I have not been able to find any solution to these problems, so maybe there is something wrong in InterViews.

Another problem is that if the user changes the size of any window before anything has been drawn in the consists-of relations subwindows, the next picture drawn can be difficult to find.

Chapter 7

Conclusions

The prototype knowledge base used in this project was very small compared to a knowledge base containing all the information about a process plant. This makes you wonder how much slower the browser will work using the complete knowledge base. Maybe some changes of the structure in it will be necessary. Everything I wanted to implement, I was able to do using InterViews classes. Hence, InterViews suited the project very well. While InterViews is built up by C++-classes, using C++ in the application I think made the whole project easier.

Concerning further developments of the graphical browser I have some suggestions. First the Multi-view object browser. In this window it might be of interest to show "connections" in a graphical way. Maybe in the same way as the consists-of relations are displayed. Currently in the consists-of relations windows all components of a composite object are shown independently of what view of the composite object they belong to. Another solution might be for the user to be able to select a view of the composite object and show only the components belonging to that particular view.

Another facility that could be added is a subwindow containing the attributes of the last selected multi-view object or class. At the moment the only way to find out what attributes an object have, is to look in the subwindow that displays how it is implemented in the knowledge base.

Finally, I would like to thank all employees at ABB, KLL Lund, for their support and kindness.

I also wish to thank Nicholas Hoggard, my first instructor and his stand- in Bo Johansson.

As my second instructor, I finally thank Karl-Erik Årzén from the Department of Automatic Control at Lund University of Technology.

Appendix A

Glossary

Attribute A property of an object. Also called slot.

Browser The browser constitutes a graphical interface to the knowledge base that is common to all users of the KBCS.

Child class = subclass

Class A group of objects that share the same attributes and behaviour. Organized into an inheritance lattice.

Composite object An object that has an internal structure consisting interconnected objects representing subparts of the superior object.

Functional view The functional view describes an object in terms of the goals that it should fulfill, the functions needed to fulfill these goals, and the process components that realize these functions.

Geographical view The geographical view describes a physical object in a geometrically and isometrically correct way.

Graphic Father class for structured graphics objects in InterViews.

Graphic block An interactor that displays a structured graphics object in InterViews.

Inheritance A process where new objects in a hierarchical structure can get new attributes deduced from more general objects in the structure.

Instance An object that describes a unique member of some object class.

Interactor Father class for interactive objects such as menus and buttons in InterViews.

KBCS Knowledge-Based Control System. A control system that integrates conventional programming techniques and knowledge-based techniques using a common data or knowledge base.

Multiple inheritance An inheritance mechanism where a class may have more than one parent class. Contrasts with single inheritance.

Multi-view object An object in the main knowledge base that represents all the individual views that the object can be described from.

Object-oriented programming A style of programming based on directly representing physical objects and abstract

Panner An interactor that supports continuous two dimensional scrolling and incremental scrolling and zooming in InterViews.

Parent class = base class = superclass

Picture Parent class for structured graphics composition objects in InterViews.

Single inheritance An inheritance mechanism where a class may have only one parent class. Contrast with multiple inheritance.

Structured graphics A graphics model in InterViews that supports hierarchical composition of graphical elements; Support is usually provided for coordinate transformations, hit detection, and automatic screen update.

Topological view The topological view described the internal structure of a physical object.

View A structuring primitive in the main knowledge base. Describing an object from several views is a way of structuring the knowledge of the object into "natural" parts. The most general views are the topological, the geographical and the functional view.

References

[IT4] Asea Brown Boveri AB, Satt Control AB, Department of Automatic Control Lund Institute of Technology. *Knowledge-Based Real-Time Control Systems IT4 Project:Phase 1.* .

[C] Al Kelley, Ira Pohl. *A Book on C.* The Benjamin/Cummings Publishing Company, Inc.

[C++] Stephen C. Dewhurst, Kathy T. Stark. *Programming in C++.* Prentice Hall Software Series, Inc.

[IV1] Mark A. Litton, Paul R. Calder, John M. Vlissides. *The InterViews User Interface Toolkit.* Stanford University.

[IV2] Mark A. Linton, Paul R. Calder, John M. Vlissides. *InterViews: A C++ Graphical Interface Toolkit.* Stanford University.

[IV3] Mark A. Linton, Paul R. Calder, John M. Vlissides. *Composing User Interfaces with InterViews.* Stanford University.

[IV4] John Vlissides. *A Tutorial for InterViews Programmers.* Stanford University.

[DAG1] Gansner, E.R., S.C. North, K.P. Vo. *DAG – A program that draws directed graphs.* John Wiley and Sons. Software – Practice and Experience Vol 18(11), 1047 –1062.