D I P L O M A R B E I T

InterFLIP++
a user interface for StarFLIP++

ausgeführt am Institut für Informationssysteme
Abteilung für Datenbanken und Expertensysteme
der Technischen Universität Wien

unter Anleitung von
o.Univ.Prof. Dr. Georg Gottlob
und
Univ.Ass. Dr. Wolfgang Slany
als verantwortlich mitwirkendem Universitätsassistenten

durch
Stefan Mayer
Matrikelnummer 87 25 743
Zeinlhofergasse 5/3
A-1050 Wien

Wien, Juni 1997

Contents

Chapter 1

Introduction

I describe a graphical user interface (GUI) for the fuzzy inference tool FLIP++. Fuzzy logic allows to deal mathematically with vague concepts, such as a “delicious cake”. A fuzzy truth value can take intermediate values between true and false. FLIP++ enables the computer to “think” using rules that can be vaguely formulated. Because humans tend to use many rules of thumb in their decision making, FLIP++ is an ideal tool to model such imprecise human decision making.
In Chapter 2 I analyse some existing GUI tools and GUI builders. First, basic terms in the world of GUIs are described. Then I describe how to measure toolkit quality and thus how to compare different toolkits. A short section contains information about style guidelines. The main part of Chapter 2 is about several specific GUI tool builders. I describe how to use them and what their advantages and disadvantages are. One of these tools, WxWindows, was used to develop InterFLIP++.
Chapter 3 introduces the StarFLIP++ project. StarFLIP++ is the fuzzy logic inference project which InterFLIP++ is part of. Because StarFLIP++ implements a fuzzy inference system, a brief survey introduces necessary fuzzy logic terms and concepts. Some characteristics of FLIP++ are given to motivate the work done by myself on the GUI for FLIP++.
Chapter 4 is the main part of this thesis and describes InterFLIP++. First I show how InterFLIP++ deals with FLIP++ objects and how a fuzzy control problem can be defined. Then a real-world problem taken from a steel making application is solved with InterFLIP++. The problem concerns the control of desulphurisation in a steel plant at VA Stahl Linz, Austria. Then I describe the event driven interaction between InterFLIP++ windows and make some remarks on implementation details.
Of course, InterFLIP++ is not the only GUI for fuzzy inference tools. Chapter 5 describes other fuzzy tools and compares them to

FLIP++/InterFLIP++.
Finally, in Chapter 6 I summarise the thesis, conclude and give an outlook on future developments and improvements of InterFLIP++.

Chapter 2

GUI tools

2.1 Introduction

In this chapter I describe several Graphic User Interfaces (GUI) and GUI builders and show their main differences. I tested several GUI-builders on Unix and MSWindows platforms.
Human computer interaction (HCI) considerations demand

- Information representation to be tailored to the end-user's need
- Support for actions that correspond to human reasoning about the application
- Continuous visual feedback in response to user actions
- Prevention of invalid options being issued
- Context sensitive help
- Provision for short cuts for the experienced user
- Prevention of the need for excessive a priori knowledge of the underlying system

GUI development tools have a broad spectrum:

1. Toolkits: A toolkit provides a set of widgets that have a specific look and feel and a rich set of application callable routines to specify and manipulate both of these. Toolkits are just a collection of library routines for GUI development and do not directly support application data display. The application programmer is required to use appropriate tools, for example Xlib, in conjunction with the toolkit to implement these.

The programmer writes his application, for instance in C++, compiles, and then executes his evolving user interface and the application. There is no rapid prototyping support.

2. GUI Builders are built on top of specific toolkits and provide an interactive WYSIWYG editor for GUI development. The application programmer can develop his application's user interface by a simple point and click mechanism. It allows the programmer to rapid-prototype the application layer and experiment with different designs. Once the interface has been satisfactorily designed and all callbacks are registered, the GUI builder generates the code for the user interface on the native toolkit on which it was based.

3. User Interface Management Systems (UIMS) provide an interactive editor for graphical user-interface development and also provide a high level language for user-interface behaviour specification. An important quality of UIMS ([see [19]) is strict separation of GUI-part and application. This allows platform independent development. Optionally, they also provide a set of application callable convenience routines that are at a higher level than traditional programming libraries. UIMS have utilities that take the behaviour specification and translates these onto the domain toolkit. Then the UIMS generate makefiles, compile the application code together with additional libraries, create the executable and at runtime read the data files in which the user-interface presentation layer is saved, and implement them on the screen.

4. Cross platform development kits (CPDK) can be regarded as UIMS with extended support for specific classes of applications.

2.2 GUI toolkit quality

Multitasking and multiwindow computer environments are in widespread use. Under these circumstances, interactive GUI which enhance information presentation and HCI have come to be recognised as essential norm rather than exotic decorations.

Windowing systems support windowing management, event management, input/output, attributes of GUI objects (widget). Examples of windowing systems are:

- MSWindows: MSWindows supports data exchange between different applications (OLE, DDE)

- Presentation Manager of OS/2
- NeWS (Network Extendible Window System), UNIX, server oriented
- X, MIT, almost all UNIX-versions support it, client-server architecture.

A GUI toolkit is a library providing often used windowing techniques, e.g.

- Frame: ... main window frame
- Menu: different styles (toggle, pull-right, ..)
- Canvas for graphics
- Panel (button,choice, item, list, message, slider, ..)

The quality of user interfaces is improved by using GUI toolkits (see [24]).

- Rapid prototyping
- Quick adaptations and easy changes
- Same look and feel if using equal GUI toolkits
- Concentrating on design of GUI, not on implementation problems
- Higher modularization, user interface can be changed without changing application functionality
- Improved reliability
- Decreased hard and software-dependency and thus more portable

What functions should a toolkit support?

- Coordination of all action with interaction-hardware
- Error-handling
- Special user operations, e.g. Undo
- Feedback
- Help
- Automatic suggestions concerning user-machine interaction
- Shifting of data on a screen without manipulation of data

2.3 GUI style guides

There are a number of norms and guidelines defining software ergonomic minimum requirements or a common look and feel (see [10]). A group of style guides is producer dependent.

- The Motif style guide is a main style guide in the area of UNIX user interfaces.
- The MSWindows Interface describes all Windows specific elements.
- Several versions of the Common User Access, COA, (IBM 89,91,92) describe alphanumeric and graphical interface objects, its appearances and functions. OS/2 and its presentation manager follow this style guide.

The following norms are producer independent:

- EU 90/270: European guideline for European governments which describes also minimum requirements for human-computer interface.
- DIN 66234: This German norm defines five basic principles and parameters to measure dialogue quality.
  1. Appropriate for given tasks
  2. Shall explain itself
  3. Controllability
  4. Conformity to expectations
  5. Error-sturdiness

  These principles base on an empirical study in 1988.
- ISO 9241/EN 29241 describes in part ten dialogue principles of DIN 66234 in a more detailed way.

2.4 GUI design

In order to cope with the complexity of GUI design it is useful to evaluate each step of the software life cycle (see [8]). Several iterations of

- Analysis phase / evaluation
- Design / redesign

- Implementation

may be necessary to achieve the required qualities.
Facts have to be gathered from potential users to detect critical parts of the GUI.

Some questions to be raised:

- What sort of jobs have to be modelled?
- Who contributes to the process to be modelled?
- Which objects have to be manipulated?
- Which data is used?
- Do any interfaces exist to other processes?
- How long takes a single process step?
- What in particular causes annoyance of users?
- How is the error rate of a process step?

Such field-study can be done with forms. Observation of users is important to optimise GUI modelling.

How to measure GUI quality?

- Training time required to achieve satisfactory performance
  It is important that new software tools are quick and easy to use.
- Number of errors
  Reasonable error rate by competent users measured in units of time or per number of operations.

- Exasperation responses
  Such reactions express annoyance and frustration of the user. Frequent exasperation responses may cause a strong rejection of a tool. But the absence of such a reaction does not imply a real acceptance of a tool or technology.

- How many people want to use the tool
  It reflects the attitude of actual or potential users towards the system.

- Habit formation rate
  This rate shows how quickly people learn to use a facility and how quickly that use becomes automatic.

- Irrelevant supporting actions required to perform a task
  This are incidental actions required for a job, but not directly related to the job, e.g., performing extensive save operations because of non stable software.

- Irrelevant messages
  These include information that must be disregarded but attract the user's attention.

- Time and frequency for user warm up
  How long does it take to relearn the necessity skills involved in using infrequently used tools. How much time is required to warm up each time a frequently used tool is used before satisfactory speed and accuracy is achieved.

- Decision making time
  The amount of time required to decide what to do after receiving the information necessary to analyse a problem and to select a suitable action.

- Working time
  The time or the number of operations a person can work without becoming fatigued.

- Failure recovery time
  That is the time or the number of operations it needs to recover from failures caused by user or system.

A software system and the user interface have to be

- adaptive, that means a system shall be adaptive to knowledge and intellectual traits of different users.
- transparent,
- comprehensible,
- natural,
- predictable,
- responsive,
- self-explanatory,
- forgiving,
- efficient,
- flexible and
- available.

See also [14, 5].

2.5 Selected GUI tool-builders

2.5.1 Introduction

X is a windowing system based on a network protocol, the X-protocol. This allows the design of a client-server model, a network protocol for 2 processes: The server is responsible for input and output devices and window manipulation. The client is an application that uses server facilities. It issues requests to a server to alter window size, position or other characteristics like colour. An important difference to other GUIs like MSWindows and MacIntosh, is the fact that X does not enforce a specific interface style. It only provides methods. So there are several user interface-toolkits like XView

(X window-system-based Visual/Integrated Environment for Workstations) or Motif. MSWindows and MacIntosh define and use only one GUI each.

XView is an object-oriented toolkit that follows the OPEN LOOK GUI specification and is built upon Xlib, see [11]

2.5.2 WxWindows

InterFLIP++ uses this toolbox to implement the GUI of FLIP++. WxWindows is a C++ library providing GUI facilities for more than one platform (Motif, XView, MSWindows NT and MSWindows3.x) from Artificial Intelligence Applications Institute, University of Edinburgh (http://www.aiai.ed.uk/ ~jacs/wxwin.html). WxWindows takes elements of other GUI API's and adds some elements of its own. It does not cover every native API completely. The following is a collection of main features of WxWindows:

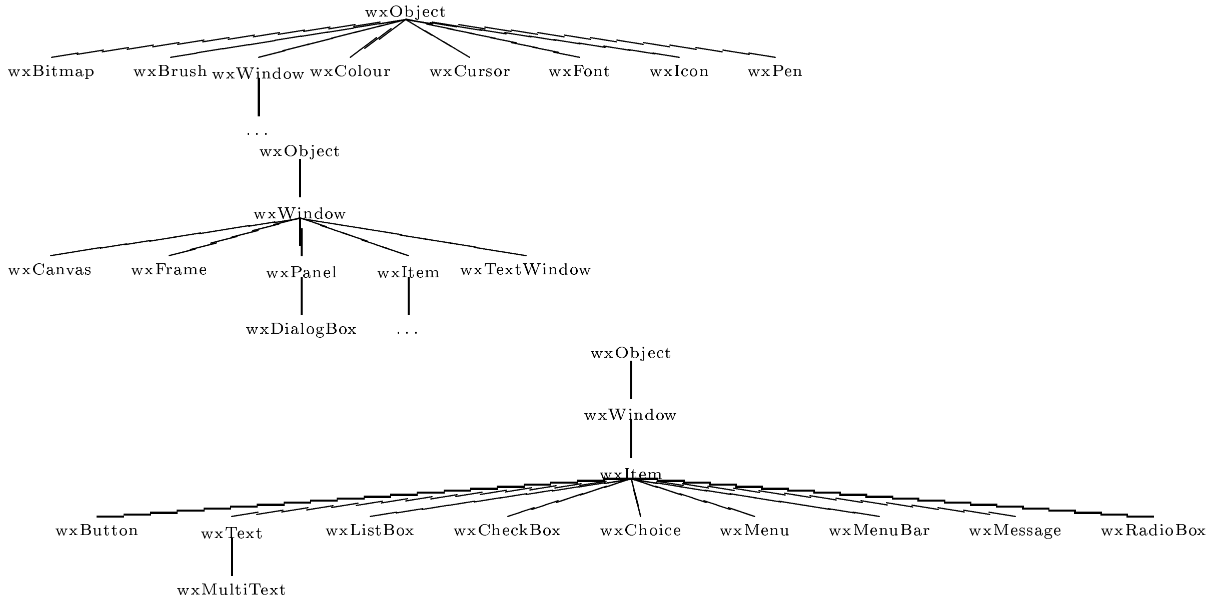
- Object-orientated API
- Development of applications for several platforms on PC, one of MS C/C++, Borland C++ 3.1/4.0, Symantec C++, Watcom C++ on UNIX, XView 3.x or Motif 1.2.x and GCC/G++ (most tested on Suns using GCC).
- Graphics include splines, polylines, rounded rectangles, ...
- Menu bars and status lines
- Toolbar class
- Simple programmatic panel/dialog item layout
- Pens, brushes, fonts, icons, cursors, bitmaps
- WxHelp hypertext help system or Windows Help

The common widgets are available (button, canvas, checkbox, choice, dialog, label (text), menus, scrollbar, textarea (several lines of text), textfield).

There are some weaknesses, like

- Large executables (small applications already need 3 MB)
- A few bugs
- Compromises made for portability

The WxWindows class hierarchy (it is split up in three parts because of its width) looks as follows:



A window can make up with a frame and a number of subwindows like the XView model. A frame window may have a menu-bar and a row of pull-down menus on top of the frame. Subwindows of a frame can be of three types: panel, canvas and textsubwindow. A panel is used for user input items like buttons, lists,.... A canvas is used for drawing graphics. Several objects are provided for drawing: colours, fonts, pens and brushes. Menus are used in menubars or popup menus. In MSWindows and Motif it is a standard UI component, in XView it is simulated.
Event handling is a very important part of every GUI toolkit. WxWindows objects receive events by calling user overrideable handlers like OnEvent, OnChar and OnSize. In XView and Motif events are handled by Xlib callbacks, for panels there is a OnCommand member function. WxWindows has its own event system like wxMouseEvent wxEVENTTYPE_LEFT_DOWN.
The graphics system origin is the top left corner of a window. Dimension numbers in WxWindows are pixels. Other APIs as MSWindows use a more flexible size format called 'dialog units' based on the size of the standard system font. A canvas has a mapping mode that determines the memory of dimensions. Drawing may be done using mm, 1/10 mm, pixels or points.

To compile and link an application WxWindows provides standard *makefiles* for each platform. UNIX and PC makefiles are not compatible, so one makefile is required for each environment.

WxBuilder

The wxBuilder is one of the utilities available for wxwindows. It is a tool for interactively building a GUI for a WxWindows application. First the user chooses and places frames, dialog-boxes, subwindows and panel items, and then the wxBuilder generates appropriate C++ code for compiling with the WxWindows library on MSWindows or X. The wxBuilder produces skeleton code that the user has to complete. Another possibility is to use the builder as a prototyping aid. It can be useful for beginning WxWindows users to use wxBuilder as a learning aid, because there is a clear correspondence between GUI and C++ code.

WxBuild or wxbuild.exe (MSWindows) can be started with an optional project filename. Project files contain all information relevant to an application like bitmaps, files and icons. When wxBuilder starts two windows open:

- A main window where the application can be build (see Figure 2.1).On top of the window there is an object editor canvas. Below there is a listbox containing a list of all top-level windows (all the frames and dialog boxes). Clicking on a window in this listbox creates the window and updates the object editor. The status line is divided in two fields: the left field displays help messages and the right field shows the current 'mode', either edit or test mode.In test mode, certain events from project action (like pressing buttons or selecting menu items) will do the user-defined action. In edit mode the same events pop up a dialog to allow editing of that action.

- A second window contains the wxWindow objects (see right figure in Figure 2.1).

In a menu and mouse driven dialog a hierarchy of window classes and its attributes can be build up from scratch. The first frame that is created is taken to be the main window. For each top level frame a window opens where items can be dropped. To put an item to a window an item has to be selected in the object window. Then it can be placed to the right place on window with a mouse click. The object canvas in the main window updates automatically. Here all items can be deleted or edited interactively. Another possibility to edit items interactively, as the object canvas, is the application window.

Figure 2.2 shows the wxBuild main window with a small example. A panel contains a listbox and buttons to add and delete items of the list and a text object to input information and a textwindow.

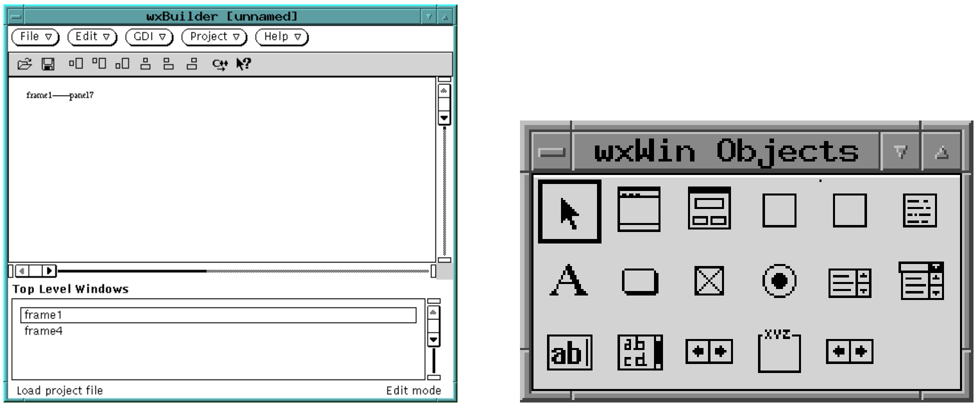


Figure 2.1: wxBuilder; left: the main window, right: object window

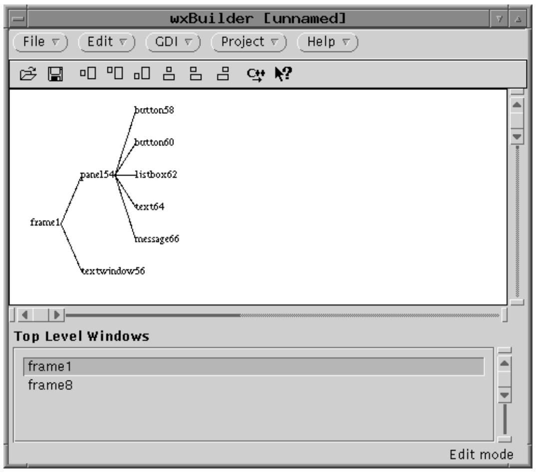


Figure 2.2: wxBuilder - main window with example

Figure 2.3 is the application window showing the example.

Each type of window has a property editor associated with it that allows tailoring of specific window characteristics. A right mouse-click on the panel item or the object in the hierarchy window invokes the property editor. This pops up a menu for editing or deleting the object. Figure 2.4 shows a property window for a button.

Click on the C++ tool on the tool bar, or choose Generate C++ from the Project menu. A report window will be displayed with error and warning messages, and if successful, source files will be written in the current project directory.

Problems:

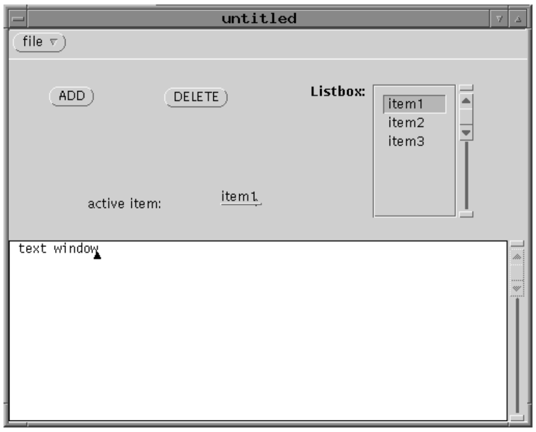


Figure 2.3: wxBuilder - application window with example

Figure 2.4: wxBuilder - property window

- No UNDO
- Little support for graphics device like colours, pens, brushes, cursors, ...
- At least basic knowledge of WxWindows is necessary
- Little event handling support
- No source file editor
- No debugger

I used WxWindows for UNIX and XView.

2.5.3 Tcl/Tk

Tcl (Tool Command Language) is a script language and an interpreter for this language. Tk is the X-Windows toolkit for Tcl and consists of about three dozen Tcl commands to create and manipulate GUI widgets. A Tk application consists of an event (defined by X-Protocol) driven control flow. New features can be implemented in C to extend Tk [27]. There is an interesting article in German language in [21] describing XF and SpecTcl.

XF

XF allows to build and modify interactively a user interface based upon the Tcl/Tk package. I tested XF release 4.0 under Linux. The output generated by XF is a Tcl/Tk script that can be executed with a Tcl/Tk interpreter called wish or any other Tcl/Tk interpreter. XF may be used for prototyping and for the complete implementation of an application. To implement a complete application, it is necessary that the user is able to write Tcl/Tk code. XF can free the user from a lot of implementation work, but still, parts of the functionality have to be implemented by him. So a basic understanding of Tcl/Tk is required.

In the following part I describe how to build a GUI application from scratch.

- After starting XF, two toplevel windows appear. One is the main window (see Figure 2.5). The empty toplevel is the workspace. Widgets are inserted into the application by selecting the widget class to insert and pressing one of the two buttons at the bottom of the main window. Another possibility is to double-click on the widget in the item list. A new widget is inserted into the current widget path and is shown in the workspace.

- The layouting process starts, if all widgets are inserted to the widget tree. Typically, the basic layout structure of an application is defined by using *frame* widgets. Widgets are grouped together by inserting them into the same frame. The user can choose between two ways of layouting with XF. He can manipulate the widgets directly (usually when the widgets are placed), or he can use special dialogs where all parameters for the layouting can be set (the usual way for packed widgets). Direct manipulation is only possible when the layout dialog (Configuration/Layouting) is activated.

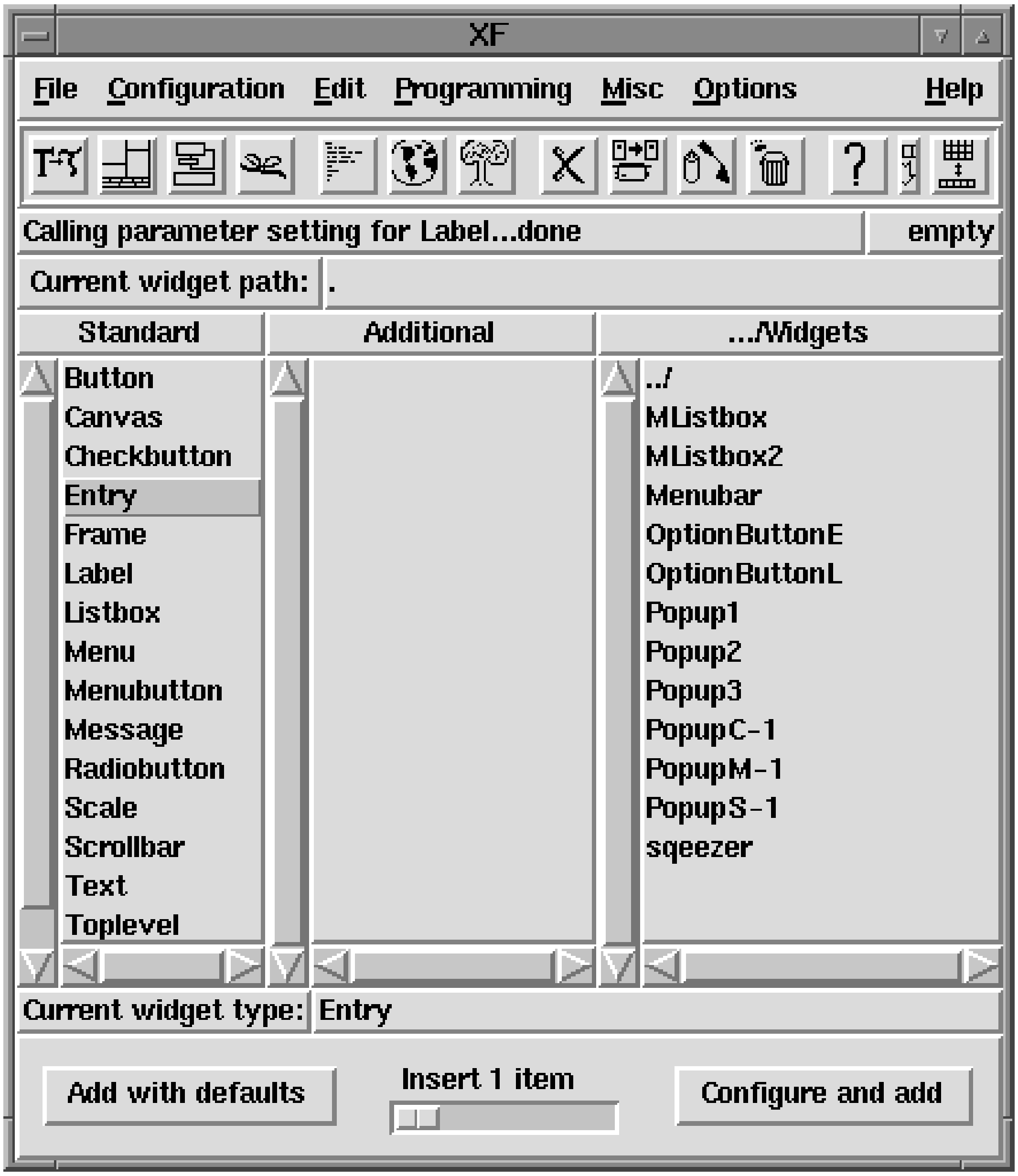


Figure 2.5: XF main window

- Then widget parameters can be set, for example text fonts, the foreground colour or a bitmap. There is a the menu item (Configuration / Parameters) that starts the parameter dialog (see Figure 2.5.3). To bind functionality to a widget, there is a *command* parameter. Each widget has its own dialog and set of parameters. Of course, certain parameter are identical for a great number of widgets.

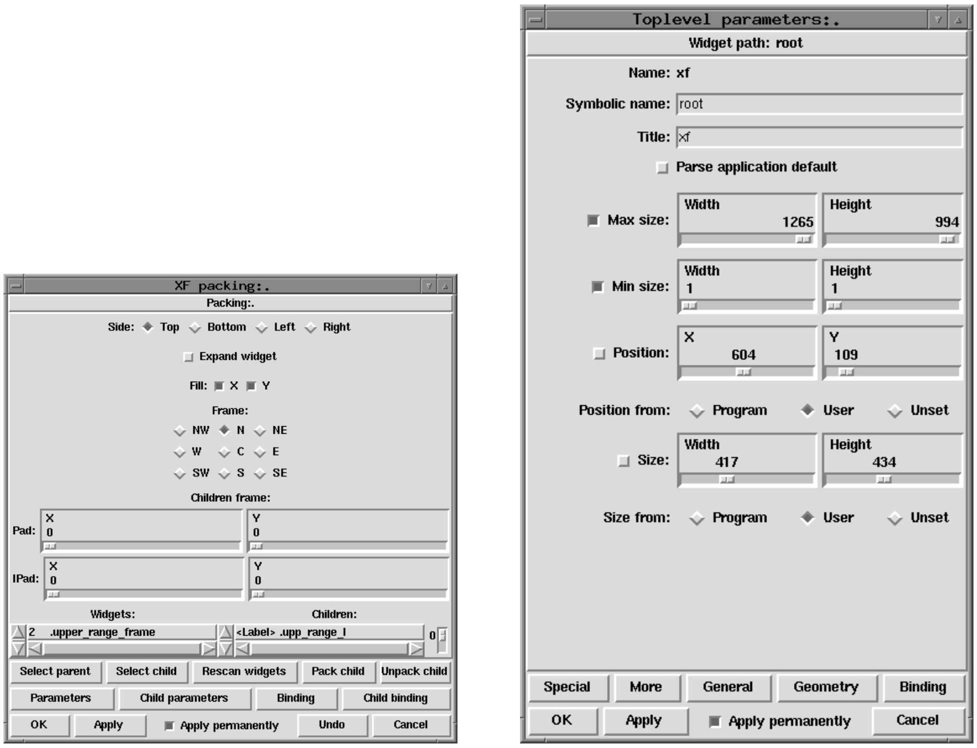


Figure 2.6: XF parameter dialog; left: top parameter window

- If the interface is finished, the functionality must be implemented. Tcl/Tk procedures have to be implemented. These procedures are called by widgets like buttons, listboxes, menus, ... Here Tcl/Tk programming skills are needed. XF supports the implementation of procedures with special dialogs (see Figure 2.7), which are invoked by the menu item Programming/Commands. This dialog helps to create, edit and delete procedures and commands.

XF provides the concept of templates. Templates are files that contain Tcl/Tk code defining a widget structure or procedures. A widget can be loaded by the user to add it to the application program. Templates behave the same way as other code written by the user. Available templates are shown in the main window. Special types of templates to reduce the size of the application program are available (autoloadable templates). XF provides no debugger to check the code.

Technical details:

- XF runs under the following platforms: UNIX/Linux (2.0.x) Windows95/NT.

Figure 2.7: XF procedure dialog

- Software requirements: wish for Tk4.0 and Tcl 7.4 or higher.

- The disk space requirements are about 7 MB.
- It is free.
- XF can be obtained from

  `http://www.cimetrix.com/sven`

  .

SpecTcl

SpecTcl is an interactive GUI-builder with a lot of features:

- It can build graphical user interfaces for Tk.
- It is able to generate Java code, if it is started with specJava that is a script.
- It is portable for Unix, MSWindows and MacOS.
- A HTML-viewer shows help texts.
- Templates

Figure 2.8 shows the main window and a general view of the major functional regions. The main window appears if SpecTcl is started. The canvas in the middle of the window is an area where widgets are placed in order to build a user interface. This builder uses geometry gridding, which means that the geometry is in terms of some unit other than pixels. A widget may occupy a rectangular group of cells. A cell is created by the intersection of a row and a column. A widget is created by dragging a widget from the widget palette to the canvas. This widget is created with the appropriate Tk defaults. Double-clicking the widget in the canvas brings up an option sheet window. To alter the generic options of a widget, the widget in the widget palette has to be double-clicked. The widget-palette contains the following widgets and main options:

- Button has common options like bitmap and help like almost every other widget. The command option specifies the event the button triggers.
- Checkbox Special options are variable, onvalue and offvalue.

- Container: Predefined modules can be reused with this object using the option class.

- Listbox has a selectmode and a setgrid option.

- Message can have a textvariable that contains the text.

- Scale has a *from* option and a *to* option to specify the range of the scale. Another option called variable that contains the name of the Tcl variable that is altered with scale and a command option to trigger an event.

- Text has options to specify such things like tabulator steps.

- Canvas has few options. Xscrollincrement and yscrollincrement are one of the more canvas-specific ones. The GUI builder does not provide drawing functionality.

- Entry is an input area and has a textvariable option but no command one.

- Label has all common text-specific options.

- Menu button has a bitmap, a menu and a textvariable option.

- Radio button has a variable, a textvariable and a value option.

- Scrollbar has a command option.

Common attributes that can be set easily by selecting a menu instead of dealing with the option sheet:

- An attribute to select which side the widget shall stick in the cell (centred, upper left corner, ...).

- Text justification attributes: centre, left or right.

- Text styles like Bold or Italic.

- Font size

- Foreground colour

- Background colour

- Border styles like flat, raised, sunken, ...

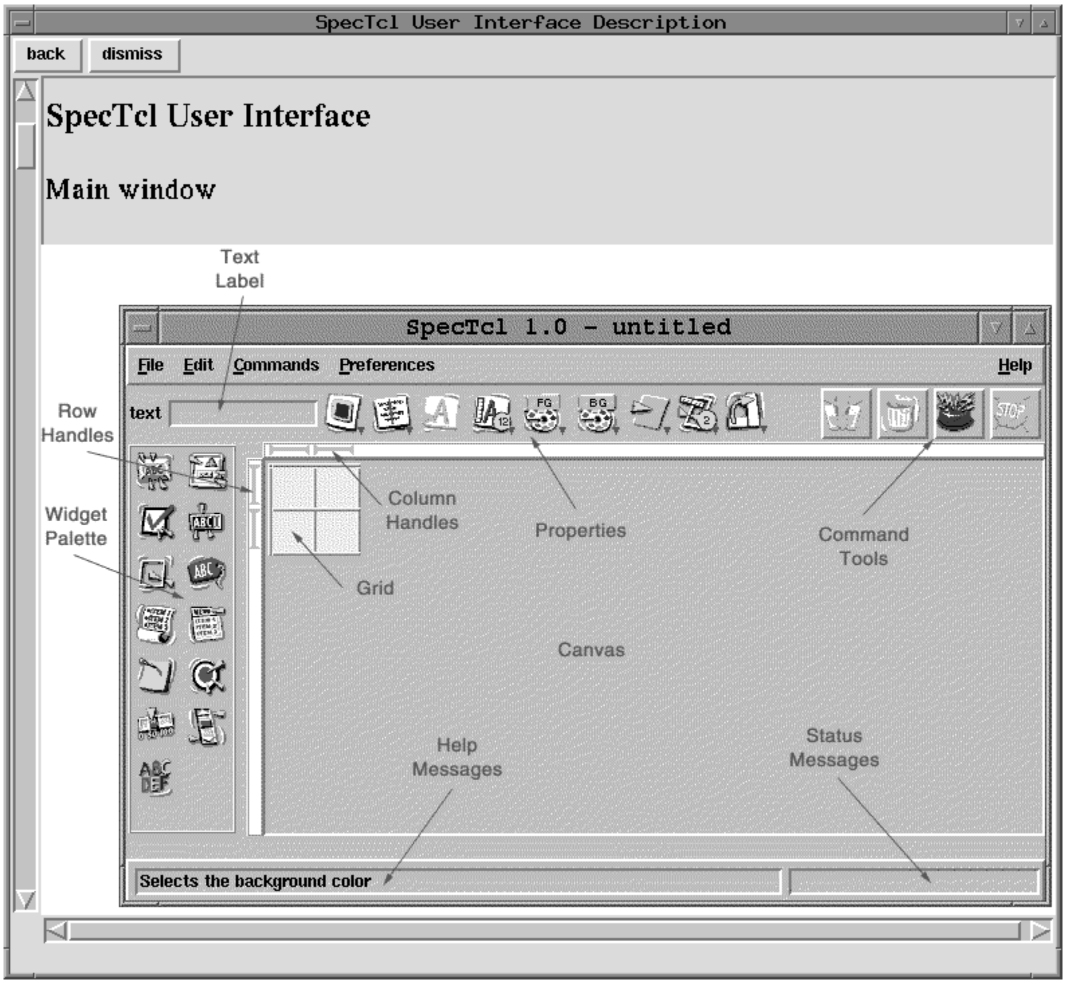


Figure 2.8: SpecTcl user interface - help

- Vertical or horizontal orientation of the widget.
- An attribute to set the thickness of the border line.

If all widgets are defined, the user has an opportunity to test the GUI by clicking a button. Figure 2.5.3 shows a GUI for an InterFLIP++ window (see Chapter 4). The resulting Tk code is stored in a file

```
filename.ui.tcl
```

.

Technical details:

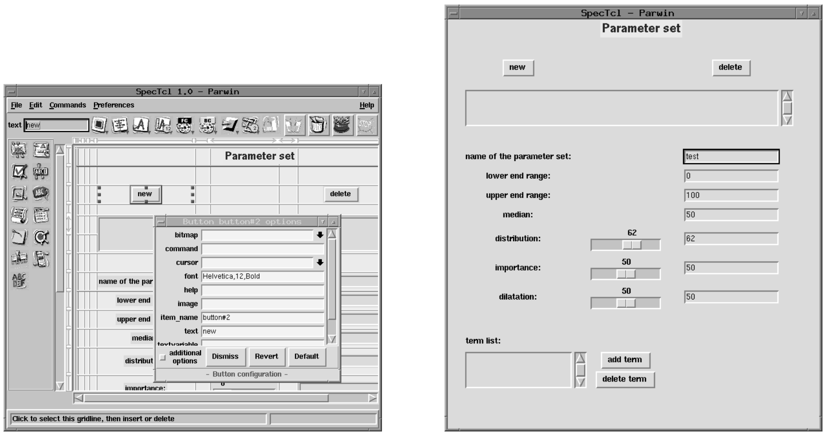


Figure 2.9: SpecTcl example; left: the main window, right: interface test

- `http://redsonja.sunlabs.com/research/tcl/spectcl`
- Software requirements: Tcl 7.5, Tk 4.2
- Disk space requirements: about 2 MB
- The tested GUI builder was a beta release 1.0.
- SpecTcl is, as XF, entirely written in Tcl.

2.5.4 Java

In this part I examine two GUI builders for Java. Java is a new, cross-platform, object-oriented programming language. First I introduce main features of Java, then I show two examples of visual development environments. Both of the development environment tools have visual GUI builders.

Introduction

Java resembles C++. It is object-oriented and supports inheritance, encapsulation, multithreading, polymorphism and dynamic-linking. It offers some differences to C++:

- The run-time management handles memory management and garbage collection. Therefore classes do not need destructors. There is no need to allocate memory directly. That is why there is no need for pointer arithmetic. Java does not use pointers at all.

- Java software is inherently cross-platform. This is a major advantage for developers of software for different platforms. Java can run on any computer system with a Java Virtual Machine (JavaVM), which simulates a computer with software. The result of a Java compilation is Java byte code. In other words Java byte code corresponds to binary executables in other languages. That is the code for the theoretical machine, the JavaVM. JavaVM implementations exist for MSWindows, UNIX, MacOS and other operation systems. JavaVM interprets byte code. A byte code verifier ensures security (viruses). Without modification the same byte code executes on different platforms. C++ code for WxWindows, that I described before, has to be recompiled for every platform.

- It is not possible to overload operators.

- There is no multiple inheritance allowed, but interfaces allow to sprinkle design ideas among different classes. Classes that use interfaces must fully implement any methods the interface defines, or they must be defined as abstract.

- Java supports multi-threading.

- Java has a efficient model for error handling.

- Java has no constructs like unions, templates or macros, but classes.

Java applets are interpreted by web-browsers. Stand alone Java software is called an application. It runs through a Java run-time environment (byte-code verifier, byte-code interpreter, JavaVM, Java class libraries). The Java package *java.awt* stands for Abstract Windows Toolkit. It consists of classes to construct GUIs. Graphical operations can be performed in a platform independent way. The class contains common widgets like buttons, frames, labels or text fields Java AWT includes only elements that are supported by all major operation systems. Therefore Java AWT has a limited vocabulary of GUI widgets and features. Special elements or methods of a single native API are not fully encapsulated. A Java application needs at least a basic window class like a Frame or a Dialog class. An applet uses the web-browser window instead. GUI elements are called components. Button, Canvas or Checkbox are different components. For further information about Java, there is special information for C++ programmers like [4].

Mojo

Mojo is an object-oriented, visual development tool for creating Java applets and applications. It can be obtained from Penumbra Software, Inc. I used a free evaluation version of Mojo 2.0. Mojo offers a drag-and-drop environment to develop Java applications with minimal coding. The rapid development part is called Mojo Designer. The second part of Mojo is the Mojo Coder. The Coder organises the object into a project hierarchy and visualises it. A code editor is included. Mojo includes a compiler, a debugger and a viewer (browser) for testing. The user can replace these tools if he likes.

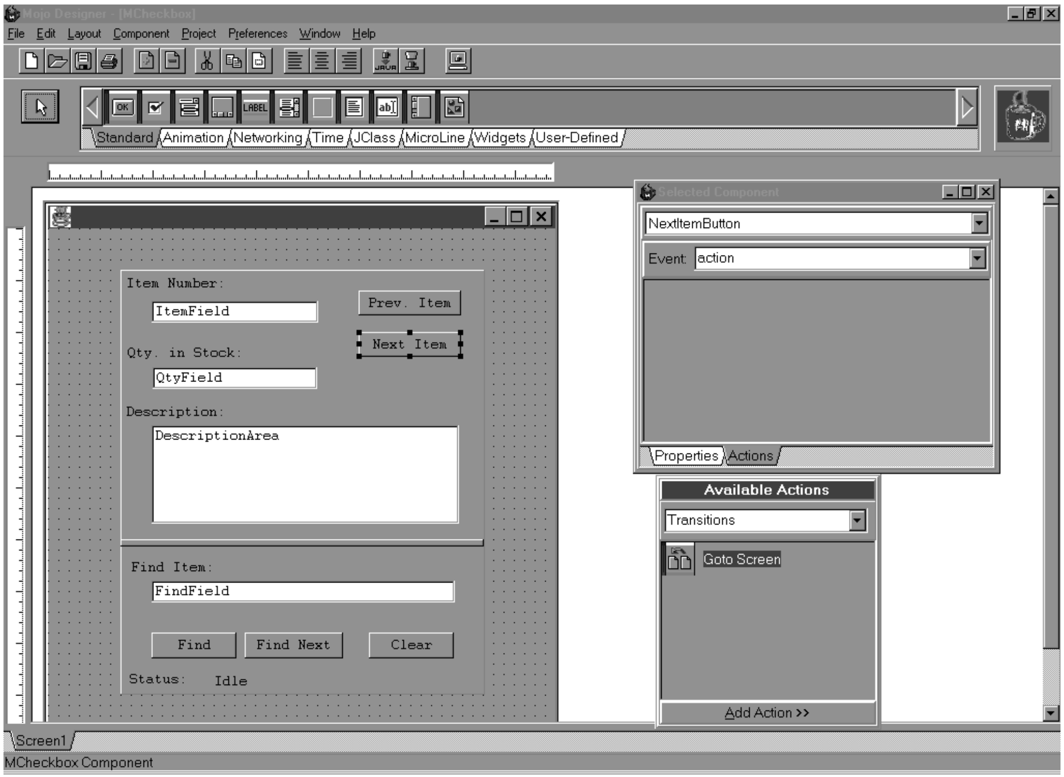


Figure 2.10: Mojo Designer window

I will mainly describe the Mojo GUI builder, the Mojo Designer. The Mojo Designer helps to create screens and simple Java applications with a drag-and-drop look-and-feel. More experienced Java programmers who want to create more complicated applications can use the Java Coder to write Java code. Figure 2.10 shows the Mojo designer with a small example. The *menu bar* on top of the window provides access to all Mojo Designer features. Below the menu bar is a toolbar for often used features. Underneath there is

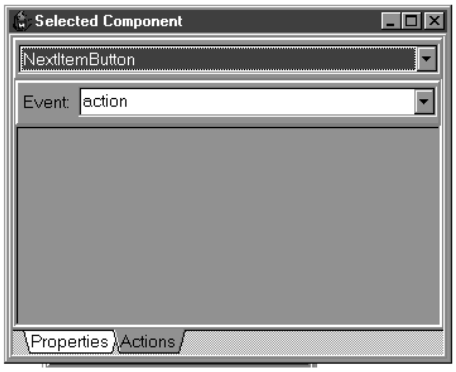


Figure 2.11: Mojo Selected Component window - action tab

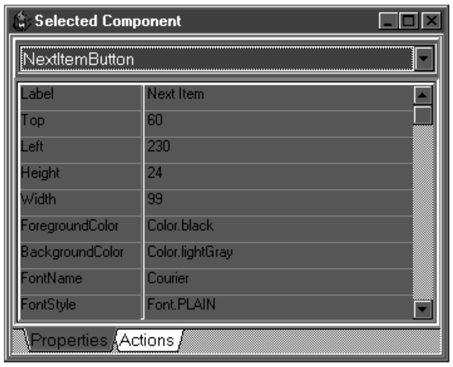


Figure 2.12: Mojo Selected Component window - property tab

the *component palette* that lists all components available in the component library. The component palette is organised in a hierarchical way.

- The standard component container includes components such as button, textarea or lables.
- The time container include different clocks and a calendar.
- The widget container offers a ticker tape, blinking text and a gauge object.
- Other containers in the component palette are animation, networking jclass and microline.

The *drawing area* is a central part of Mojo Designer. This is where the design of the layout of the application is built. The user selects a component by clicking it with the mouse in the component palette. The selected component is added at the position of the current screen where the user clicked. That is the common drag and drop functionality. In the drawing area components can be repositioned, resized or destroyed.

If more then one screen exists for a project, then the *screen tab* at the bottom of the Mojo Designer will be helpful to select one.

To manipulate actions and properties for components there is a special window, the *Selected Component Window*. This window displays actions and properties available for the currently selected component in the drawing area. Properties define visual aspects of the component, such as size, position, font and color, see Figure 2.12. A certain property can be selected by a mouse click. Then the user is able to edit the value of the property.

The *Action Tab* of the *Selected Component Window* defines the actions for the components in response to events. Mojo provides a number of predefined actions. The user can quickly add an action to a selected component. There are actions for

- Multimedia, e.g. play sound files,
- Transitions, that means screen-to-screen actions,
- Networking, e.g. e-mail and FTP functionality.

All events for the selected component are listed in the *Selected Component Window*, see Figure 2.11. To associate an action with an event, the user has to select the event first. Then an action for the event can be selected from a drop down list box in the *Available Action Window*, see the smallest window in Figure 2.10. After selecting the action the user has to drag-and-drop the action from the *Available Actions Window* to the *Actions Tab*. Then the user has to enter appropriate parameters for the action.

The *Mojo Coder* is an important tool to add or override functions for classes created with the Mojo Designer. The *Mojo Coder* shows all defined components of the current project, see Figure 2.13. The top left window shows the *Current Project* using hierarchical graph. The user can click a component to see more details. The top right window shows all defined actions for the selected component. To edit the code of a function the user simply has to click it and the code is presented in the bottom code editor, see Figure 2.13. If the user clicks the right mouse button in the *Available Functions Window* of the *Mojo Coder*, then he can

- override,

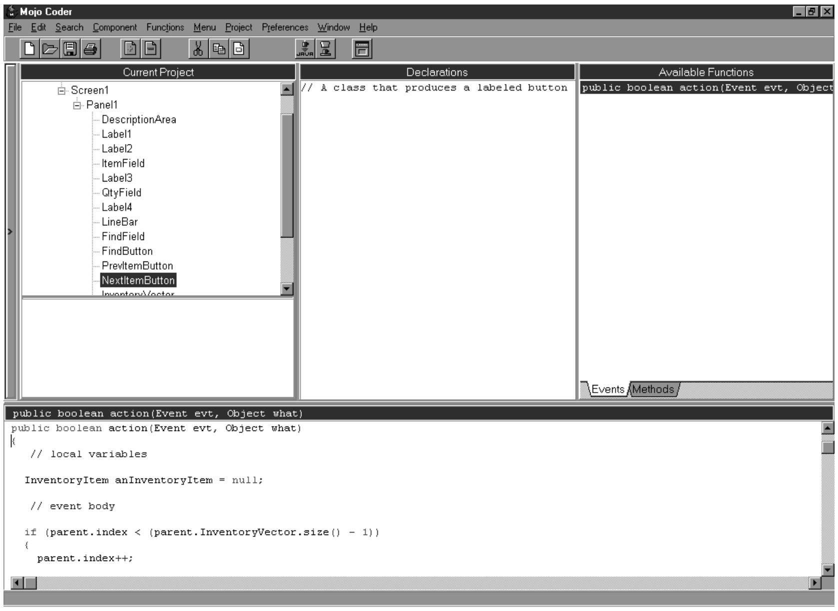


Figure 2.13: Mojo Coder Window

- add, or
- remove

functions.
All user defined class definitions, class constructors, and global variables must be created under the Project component. Mojo supports extending of existing classes. User can do this by overriding existing functions.

New components can be created. A new project has to be defined with the Mojo Designer and the Mojo Coder. Then the project can be stored as a new component using the *Make Component* item of the *Component* menu item both in Mojo Designer and Mojo Coder.

Mojo has a Debugger. The user can set breakpoints, step through the code and examine variables. Mojo includes on-line help, predefined examples of projects and a turorial. All predefined components are described in the on-line help, including properties, events and available actions.
System requirements for Mojo 2.0:

- 486 DX2/66 or higher
- Windows95

- 16 MB RAM (32MB recommended)
- 30 MB free disk space

Mojo2.0 uses Java Development Kit (JDK) 1.1.

Java Workshop

I examined the Java Workshop 2.0 (JWS) pre-beta version from Sun with Windows95. JWS includes

- project management,
- version control
- a source file editor,
- a visual Java GUI builder,
- a project builder,
- a debugger,
- on-line help, project examples and a tutorial.

I will explain only the main parts of JWS that are necessary to use the GUI builder.
The first step is to create a new project. There are three different types of a project: applet, standalone application and package. A project can have either no GUI, a manually created GUI or a GUI to be created with the *Visual Java GUI Builder*.
The *Visual Java GUI Builder* helps to design GUI applets and applications. When the GUI builder starts, a separate *Layout Window* opens with a default panel container, see Figure 2.17. The layout window uses a grip system that makes it possible to layout components relative to each other. At any time of the layout process the user can switch the GUI builder mode from edit to preview to view an approximation of how the layout will look in the application. If the *Visual Java GUI Builder* is active then a GUI builder panel is displayed in the JWS under the JWS toolbar, see Figure 2.14. The GUI panel contains basic GUI objects, such as a radio button or a label.

JWS calls a callback function an operation. It consists of a filter that defines when an action happens, and an action. An operation is usually

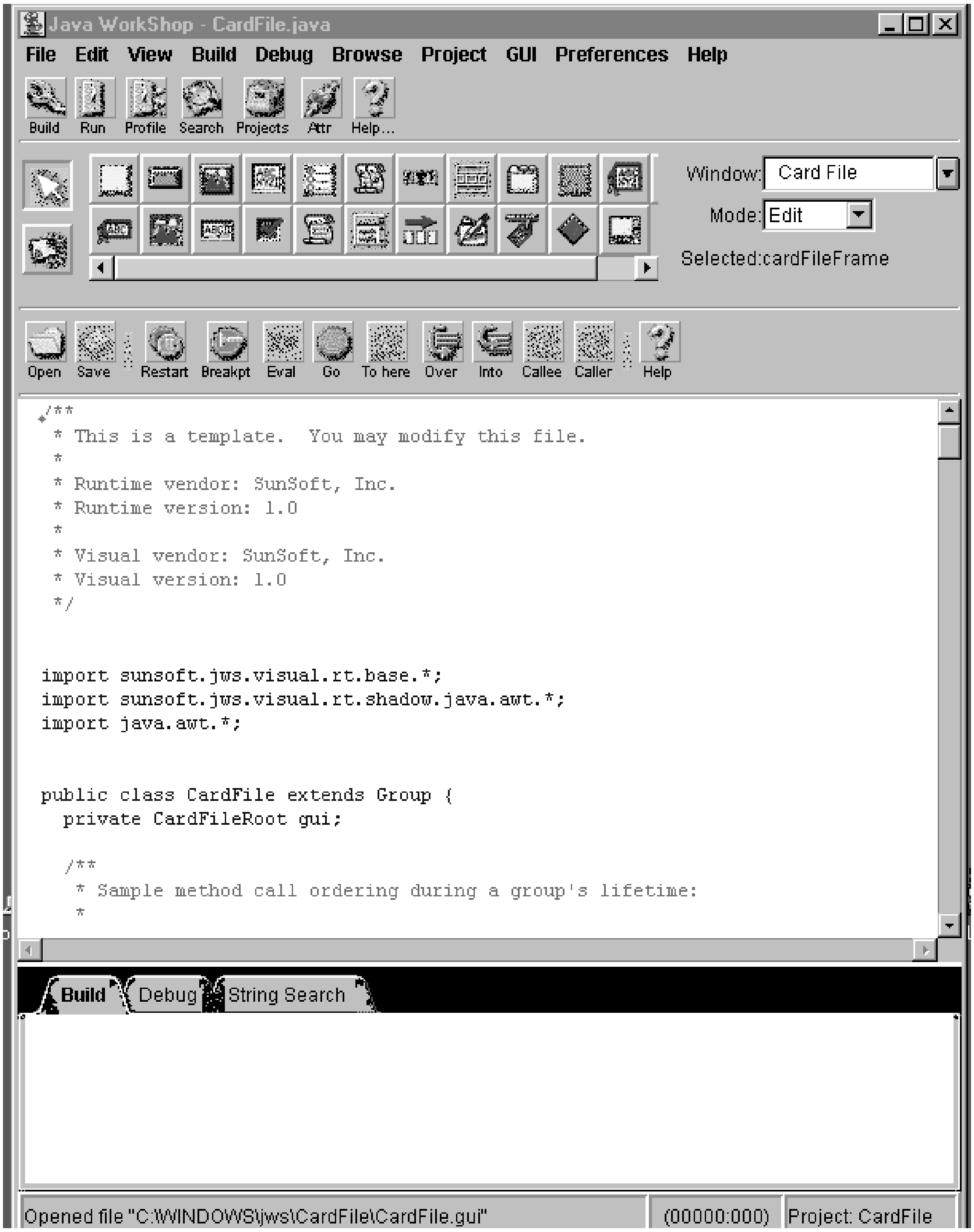


Figure 2.14: JWS main window

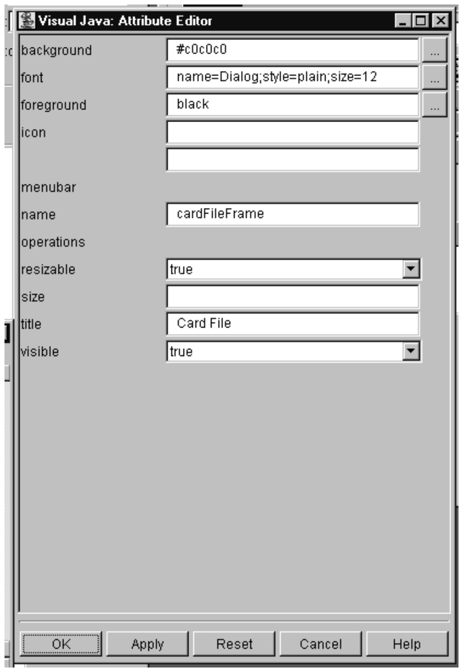


Figure 2.15: JWS Attribute Editor

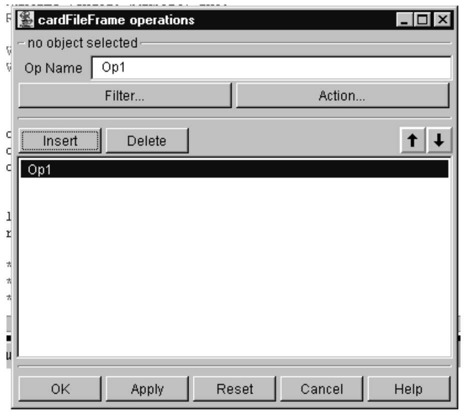


Figure 2.16: JWS Operations Editor

triggered by events from AWT groups. Another possibility are messages from other groups. The GUI builder has an *Operations Editor Dialog* that helps to associate new behaviour (actions, event filters) to components. In many cases the *Operations Editor Dialog* has predefined operations, which might automate all of the code generation. A second way to add operations is to edit the group file directly in the source editor.
JWS supports modification of attributs of GUI-components. The user has to

Figure 2.17: JWS layout window with an example(cardfiles)

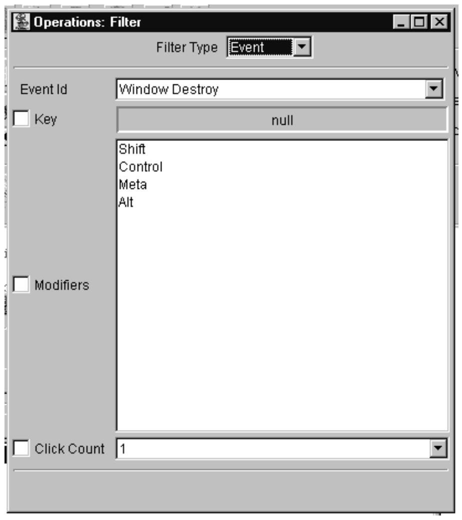


Figure 2.18: JWS Operations Filter Event Window

generate source code, add behaviour to components and build an application before he can run it. JWS offers three ways to add operations:

1. Using only the Operations Dialog box

2. By editing the source code

3. Through a combination

In the following I describe how to use the Operations Dialog Box. First of all the user has to select a component in the layout window. Then the *attribute editor* has to be opened by clicking the attribute editor in the main toolbar, see Figure 2.15. By clicking the edit operations button in the attribute editor, the operations dialog box opens, see Figure 2.16. A new operation can be added by clicking the input button in the operations dialog box. The triggering event can defined in a dialog that can be opened by clicking the filter button, see Figure 2.18. By clicking the actions button the response to the trigger can be defined.

A main disadvantage of the current version of JWS is a very slow creation of window-pages. That fact makes it difficult to jump between different JWS windows. JWS is based upon HTML-pages.

2.5.5 ET++, MET++

ET++ is an object-oriented application framework written in C++ by Gamma and Weinand at the University of Zurich and the UBILAB (Information Technology Laboratory of the Union Bank of Switzerland) see [9],and [26]. MET++ is a object-oriented multimedia application framework, see [1]. MET++ is based on ET++ and extends it to 3D graphics, audio, video, and music.

1. ET++
   ET++ stands for Editor Toolkit realised in C++. It is an object oriented class library. It supports development of 2D GUIs with direct manipulation. Source Code is available. ET++ is running on UNIX-platforms (SunView or X-Windows) and for MacIntosh OS.
   Main features are:

   - Creating, saving and opening of documents (file dialog)
   - Handling of several documents in different windows
   - Scrolling with scroll-bars, auto-scrolling
   - View-splitting
   - Automatic layout management
   - Mouse and keyboard event handling

- Undo with multiple levels
- Clipboard cut/copy/paste

Most of the about 600 classes are derived from the base class *Object*. The exploration environment consists of an

- Object structured browser, which visualises object hierarchy information with meta information from *Class* objects
- Source code browser, so the developer has always direct access to the source code of an object and a
- Class hierarchy browser, which displays the static inheritance tree.

Although ET++ is a complicated tool with many classes and rich functionality, only 20% of Xlib procedures are used for X11-ported ET++.

2. MET++
MET++ is a multimedia tool for ET++. 3D graphics can be used in views, dialogs and menu items. A hypertext link mechanism (of LinkManager) allows the usage of hot words, hot images and buttons in textual and graphical views.

2.5.6 Conclusion

In this section I listed some ideas about GUI-toolkit quality and GUI quality. I described a number of different GUI-builders.
WxWindows helps to develop applications for different platforms, such as Unix (Motif and Xview) and MSWindows. It supports a number of C++ compilers (GCC, Borland, Symantec, Watcom, ..). The GUI-builder is quite simple. It can be used to create a layout, but the dynamic part of the GUI of an application has do be implemented with a source editor that is not an integrated part of the GUI-builder.
XF is a builder for Tcl/Tk. It is easy to create a layout structure and set parameters for widgets. XF offers a procedure dialog to create procedures (callback functions).
SpecTcl generates Tcl/Tk code or Java code. It supports drag-and-drop functionalty and is portable for Unix, MSWindows and MacOS.
Mojo is a visual development tool for Java. It supports developing layout and procedures. Mojo supports writing procedures for events. It has a debugger and a code editor included. A builder creates byte-code for Java-applications.
Java Workshop is a development tool for Java like Mojo. It has a similar functionality, too.

name	visual layout builder	procedures	viewer	on-line help	debugger	Undo
WxWindows	+	−	+	−	−	−
XF	+	+	+	+	−	−
SpecTcl	+	−	+	+	−	−
Mojo	+	+	+	+	+	−
JWS	+	+	+	+	+	−

Table 2.1: Comparison of GUI-builders

Chapter 3

StarFLIP++

InterFLIP++ is part of the StarFLIP++ project, which is described in [22]. The GUI for FLIP++ (first introduced in [3]) is described in the next chapter. It will be shown how to create a knowledge base and how to work with InterFLIP++ for FLIP++. The knowledge representation will be shown in an example that is typical for the whole project. The example consists of constraints needed for scheduling a steelmaking plant.

3.1 Introduction

In the first part of this chapter I present basic ideas and methods of fuzzy logic. A little bit theory of fuzzy logic is nessecary to understand the fuzzy terms used in this chapter.
Then I give on overview of the StarFLIP++ project. I describe two parts in greater detail. First Flip++, the fuzzy logic inference library and then - of course - InterFLIP++. I explain important features of FLIP++ and briefly show how to use FLIP++, because InterFLIP++ uses its data model and methods.

3.1.1 Fuzzy set history

First publication was made by Lotfi Zadeh in 1965. In the 70ies E.H. Mamdani used fuzzy logic to control a steam generator. This was one of the first industrial applications in Europe. Others followed, but fuzzy logic techniques could not reach broad acceptance in industry. Japanese companies started to use fuzzy logic in control engineering after 1980. These early applications raised a lot of interest in Japan. One reason is that a fuzzy logic system supports prototyping and incremental optimisation. It remains plain and easy to understand, for example no differential equations are used. Now fuzzy logic is used in many application areas for control or data processing (photo and video cameras, cars, factory automation, optimisation of chemical and biological processes ...).

3.1.2 Basics

The real world abounds with uncertain data. Many mathematical disciplines deal with the description of uncertainty, such as probability theory, information theory and fuzzy set theory. Stochastic uncertainty deals with the uncertainty of an event. A degree of probability describes this uncertainty. Lexical uncertainty is an attribute of human language.
The meaning of human words is very complex. Human words, in particular adjectives such as hot or small, have no exact definitions. Of course one can define the adjective *hot* for water like this: If the water has more than 35 degrees then it is hot water. Is water of 34 degrees Celsius not hot at all? Psycho-linguistics tells us how a human beeing evaluates concepts and derives decisions. An adult has a different concept of height than a child. Also the context and the background play a role.
Fuzzy logic is a method of information processing in a more human way. It is computation with words. Its vocabulary consists of fuzzy variables that can be defined in a very intuitive way. Fuzzy tools help to combine fuzzy variables with operators to sentences or so called rules. With a number of sentences or a database of rules it is easily possible to describe a situation. To do this almost no understanding of the mathematical background is needed. [25] and [20] are German language books that introduce fuzzy logic.

Fuzzy sets

Example: *tall person* A fuzzy set defines the degree to which 1.83m or 1.65m match the prototype of a tall person. $\mu(1.65) = 0, \mu(1.83) = 0.9$, which means that 1,65 is not tall at all, representing no membership to the set, but 1,85 matches the prototype of a tall person well. The range of μ is from 0 to 1, representing no membership to the set and complete membership.

Membership functions

The degree to which the value of a technical figure satisfies the linguistic concept of a linguistic variable is called degree of membership. For continuous variables, this degree is represented through membership functions (MSF).

The membership function maps each value of the technical figure to the membership degree to the linguistic term. The technical quantity is called base variable. The use of fuzzy sets defined by membership functions in logical expressions is called fuzzy logic. A German language book, [6], provides theoretical, mathematical concepts of fuzzy logic. Some of the classic books are [28] and [7].

Linguistic variables

The primary building block of any fuzzy logic system is the "linguistic variable". Multiple categories describing the same context are combined. Example: In the case of adult height not only tall but small and normal exist, too. These are called linguistic terms and represent the possible values of a linguistic variable. Usually membership functions of all terms of a linguistic variable are plot into the same graph. A linguistic variable translates real values into linguistic values.

Fuzzy rules

The rules of a fuzzy logic system represent the knowledge of the system. They use linguistic variables as the vocabulary to express the control strategy of a fuzzy logic controller. Production rules represent the relation among the linguistic variables and derive actions from the inputs. A rule consists of a condition (If-part) and a conclusion (Then-part). Preconditions are linked together by linguistic conjunctions like AND and OR.

3.1.3 Computation

Fuzzification means using the membership functions of linguistic variables to compute each term's degree of validity of a specific operation point of the process. The computation of fuzzy rules is called fuzzy rule inference. The inference consists of two main steps: aggregation and conclusion. The first step of fuzzy inference - aggregation - determines the degree to which the complete IF-part of the rule is fulfilled. Special fuzzy operators are used to aggregate the degree of validity of the various preconditions.

If a degree of membership for output terms is too fuzzy as a result for the user, there is a possibility to generate a single real number additionally. This is called defuzzification. Several methods can be used like centre of maximum or centre of area. These methods are implemented into various defuzzify operators.

3.1.4 Fuzzy controllers

Fuzzy control is a special application of fuzzy logic. A specific process has to be controlled. From a very basic angle the system consists of two objects:

- a system (process) which has to be controlled.
- a controlling unit (the controller).

A fuzzy controller is a tool for processing a fuzzy form of information in a non fuzzy scheme of reasoning. Process data has to be fuzzified as an input for the fuzzy controller. The fuzzy output of the controller must be defuzzyfied as an input of the process.
Control engineering has been extensively studied. Fuzzy control can avoid the problem of overmodeling. Heuristics are applied in the decision-making process to increase the performance of the controller. A main problem of conventional controlling methods can be avoided: the time-consuming stage of model-building. Model-building for a fuzzy controller will probably be much quicker.
A main source of knowledge to define the control algorithm is a set of rules. These rules consist of simple If-Then statements. The first part of a rule is called condition or antecedent. The second part defines the action that has to be taken. It is called consequent. This is the way a human operator tells a control expert what has to be done.
A generic structure of a fuzzy controller has the following elements:

- The information:
  A collection of If-Then rules that forms the basic knowledge about the task.
- An aggregation mechanism defines how rules are put together.
- An inference mechanism that explains how to calculate with rules.
- An output interface.

A number of different fuzzy applications are described in [16], [17], and [18]. Theoretical background on fuzzy control is given in [15].

3.2 The StarFLIP++ Project

This chapter gives a general overview about the StarFLIP++ (pronounce: StarFLIPPlusPlus) project (see [23],[22]). StarFLIP++ consists of:

- FLIP++: the basic fuzzy logic inference processor library.
- ConFLIP++: the static fuzzy constraint library.
- DynaFLIP++: the dynamic fuzzy constraint generation library.
- DomFLIP++: the domain knowledge representation library.
- OptiFLIP++: the heuristic optimising library; several heuristics have so far been implemented and tested, namely:
  - A tabu list min-conflicts repair based hill climbing heuristic,
  - A min-conflicts repair based genetic algorithm heuristic.
- CheckFLIP++: the knowledge-change consistency checker library that also allows to fine-tune the configuration parameters of a problem.
- **InterFLIP++**: the graphical user interface for all other libraries, described in this thesis.
- DocuFLIP++: the on-line documentation available separately for end-users, knowledge-engineers, and programmers, and accessible via the World-Wide-Web as HTML documents.
- ReaFLIP++: the reactive optimiser as an extension for DomFLIP++.
- NeuroFLIP++: the learning library for FLIP++ using neural nets.
- TestFLIP++: the version control and test environment for the complete library set.
- SimFLIP++: the simulation toolkit library.

StarFLIP++ allows optimising with uncertain data. Building models with antagonistic constraints is possible.

3.3 FLIP++

FLIP++ is a C++ library and the basic layer of the StarFLIP++ libraries. It performs calculations with uncertain, fuzzy data, evaluates fuzzy rules with different selectable operators. The rules are associated with priorities and structured in hierarchies, see [2], [3]. This section introduces some basic FLIP++ classes and its methods. InterFLIP++ uses them and it can be useful to know about FLIP++ methods to be able to manipulate fuzzy-system defined with InterFLIP++ (see 4.2.5). To build a fuzzy system with FLIP++ one has to split the model into different parts. To define fuzzy variables FLIP++ uses a class with information for general types of fuzzy variables called *parameter set*. After defining all fuzzy variables needed to construct for instance a fuzzy controller, fuzzy rules can defined. A fully defined fuzzy system can evaluate variables with appropriate input values. A (real) value and an instantiated membership function of the output value are the results of an evaluation.

3.3.1 Basic elements: fuzzy variables

FLIP++ provides objects to define fuzzy variables and fuzzy terms. The FLIP++ class *CombinedVariable* provides the concept of a fuzzy variable with a membership function. The variable contains data like:

- Lower and upper support value:
  That is the range where the variable is defined.
- A term list
- And the associated membership functions
- Instantiated membership functions

3.3.2 Rules

FLIP++ defines a rule like this:

- If antecedent Then consequent with certainty factor X
  - Antecedent = [clause operator clause]*
    Two or more clauses are aggregated with two different types of operators (and, or).
  - Consequent = clause

– Clause = variable Is term

– Operator = (And — Or)
  Two operators are available. The meaning of a specific operator can be chosen in the set of available aggregation operators.

– X = [0,1]
  consequence = clause operator clause

3.3.3 Operators

This section describes which operators are available.
Set of aggregation operators:

- Minimum (t-norm)
- Maximum (t-norm)
- Fuzzy and
- Fuzzy or
- An associative and compensatory and operator
- An associative and compensatory or operator
- Product
- Sum

Defuzzification operators:

- Cog - center of gravity
- Fimax - first maximum
- Mom - mean of maximum

3.4 Defining a decision making system with FLIP++

In order to write rules, more basic parts of FLIP++ have to be defined first. Every rule consists of some variables. If a variable in FLIP++ language has to be defined, the first step is to define something like a pre-variable. FLIP++

calls such a pre-variable parameter-set. It defines a type of variables (the support, the number of terms, how membership functions shall look like, ...). A variable (*CombinedVariable*) can be defined simply by giving it a name and referring to a parameter-set.
Then rules can be defined. When all necessary input variables have an input-value, an output value can finally be evaluated.

Chapter 4

InterFLIP++

4.1 Qualitative knowledge representation

A fundamental concept of fuzzy systems is the translation of qualitative knowledge, e.g., represented in human language-like terms, into a machine-usable quantitative representation.

For the evaluation of rules that contain qualitative variables, we assign a quantitative representation to the used terms so that ordinary calculation can be performed on them. Each term is described by a membership function which states the membership of this term for each crisp value in a range of possible values. The membership function can have values from 0 to 1. A value of the membership function of 1 means that the concerned crisp values fully represent the term. A membership function value of 0 states that the crisp values in these areas have nothing to do with this term. All other function values between 0 and 1 express a partial membership of the crisp value to the term.

InterFLIP++ is based on WxWindows, a public domain toolkit from Julian Smart of the Artificial Intelligence Applications Institute at the University of Edinburgh. It is compatible with most UNIX platforms (XView3.x or Motif and GCC/G++ or virtually any C++ compiler) and on PC (Symantec C++, Borland C++, Watcom C++, MS C++), see [13]. InterFLIP++ for FLIP++ is a graphical user interface and allows to create, edit, save, reload and evaluate fuzzy variables and rules.

4.1.1 Example

In this example we want to schedule a steel making plant. Three types of constraints are used:

- Chemical compatibility constraints
- Temporal constraints, to represent delivery dates
- Chemical capacity constraints, for human or machinery restrictions

Each of these constraints consists of many sub-constraints. For every relevant chemical element, for example, there is a sub-constraint. Every constraint type has a parameter set which describes common parameters (range, membership function, importance).

4.1.2 Parameter set

A parameter set defines a type of fuzzy variables. Thus it is possible create very similar fuzzy variables by using one parameter set. This helps to define rather equal variables more quickly. It defines a variable (the number of terms, naming the terms, and determining parameters which are used to shape and position the membership functions as well as colours). It is allowed to save and load parameter sets in a special file. Figure 4.1 shows the parameter set for the 3 types of constraints in the example and a rating type, which represents the root of the constraint tree and states whether the variable is satisfied or not.

How to influence the shape of a membership function with parameters. Figure 4.2 and Figure 4.3 shows different membership shapes for an example with 3 terms, range = [0..10] and a median which is right in the middle (5). The only parameters which are altered are dilatation (dil) and distribution (distr).

- The left part in Figure 4.2 distribution is 0 and dilatation is 0. Upper/lower terms are 1 for almost all of the upper/lower part of the range. The term in the middle is very small.
- The right part in Figure 4.2 distribution is 0 and dilatation is 1. Upper and lower terms are very important, too. The term in the middle is a triangular function and has a decreasing importance.
- The left part in Figure 4.3 distribution is 1 and dilatation is 0. The term in the middle has a range equal to the other terms. The MSF-shape is rectangular.
- The right part in Figure 4.3 distribution is 1 and dilatation is 1. All terms have the same range and shape (triangular).

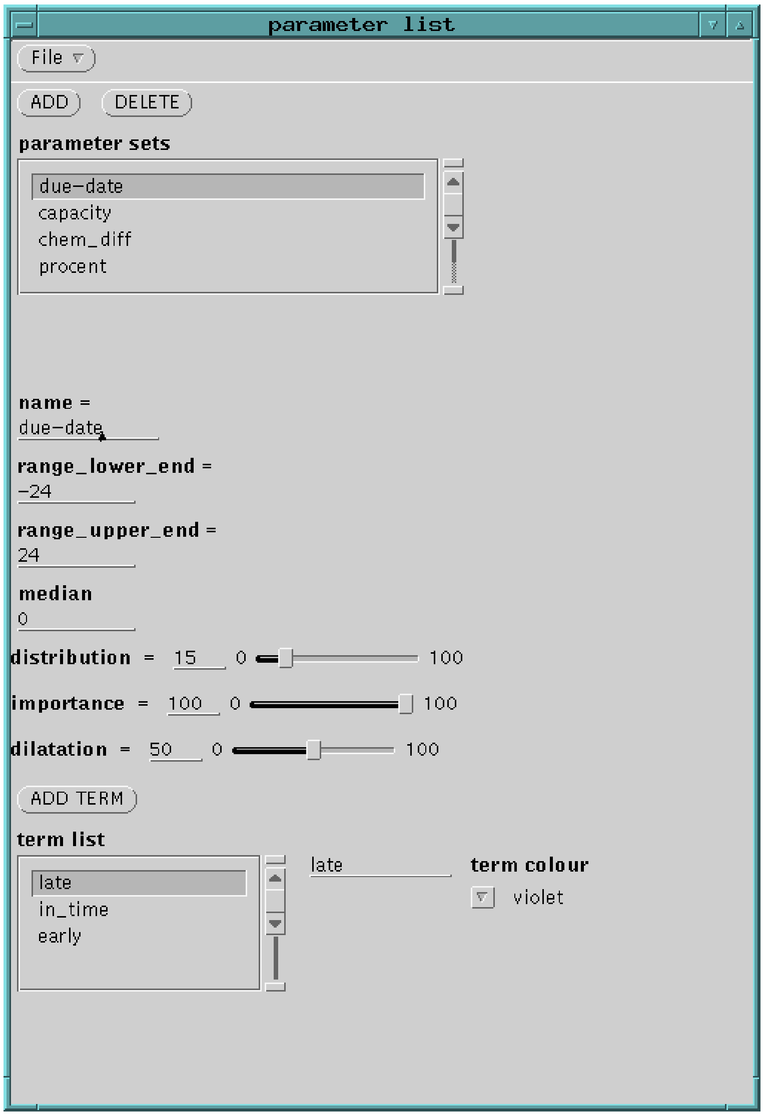


Figure 4.1: parameter window

4.1.3 Variables

A variable must have a name and an associated parameter set, so variables with similar parameters can be created more quickly. Every variable can be found in a list, the so called constraint list in Figure 4.4. By clicking the name in this list, its terms, rules and membership-functions can be shown and edited.

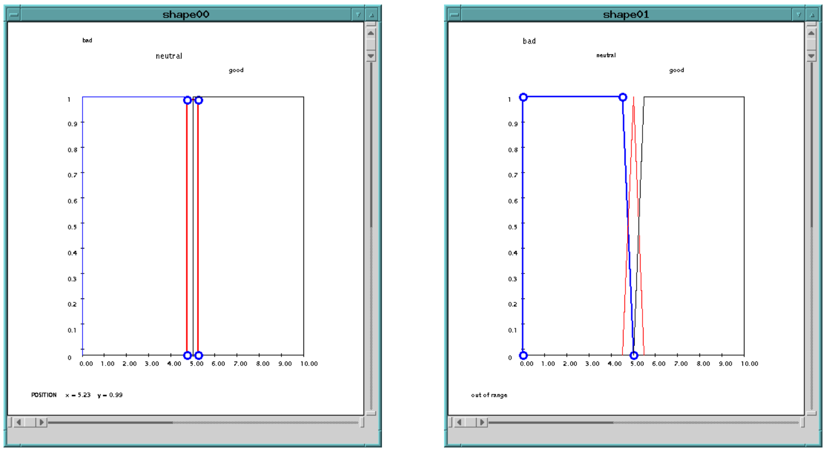


Figure 4.2: Different shape styles: left: distr=0,dil=0; right:distr=0,dil=1

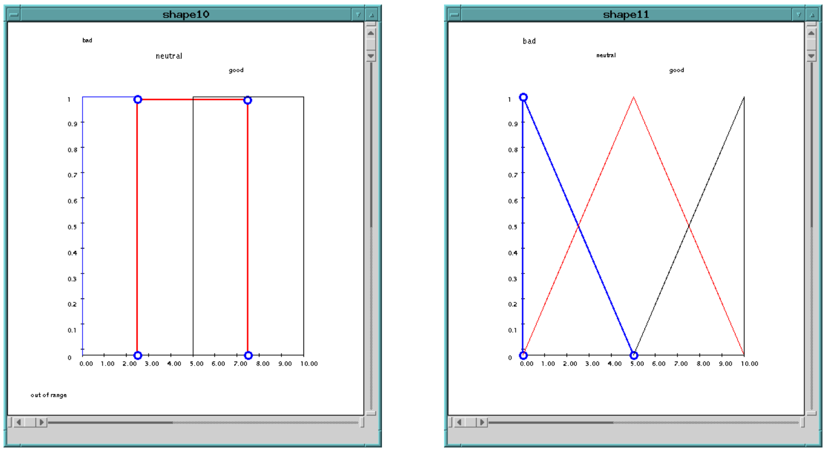


Figure 4.3: Different shape styles: left: distr=1,dil=0; right:distr=1,dil=1

4.1.4 Membership functions

Membership functions for a term of a special variable can be edited graphically, by dragging and dropping points or adding new ones to the function or deleting some. For editing membership functions there is a special window (see Figure 4.5). Refinement of membership functions will be needed to change the default function defined in a parameter set. The default membership functions are triangular. Changes of membership functions have immediate effect on the decision function.

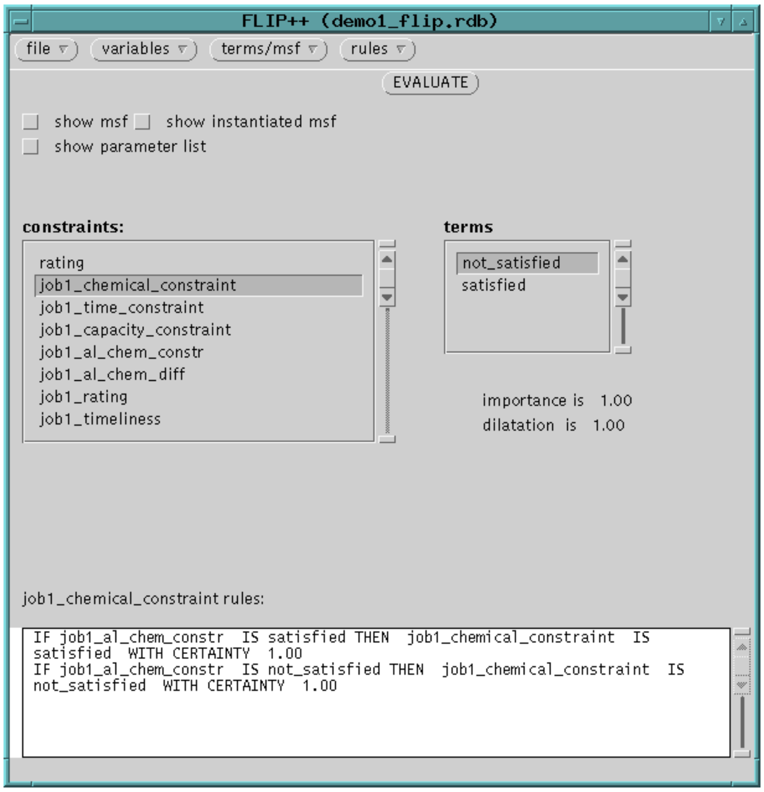


Figure 4.4: InterFLIP++ main window

4.1.5 Rules

Rules represent the knowledge about the connections of the qualitative variables, which in turn represent themselves the static knowledge of the domain. They also describe implicitly a constraint tree. This makes it possible to reduce the complexity by using smaller modules. Such modularity makes complex problems more understandable for users. The syntax of a rule is a simple If-Then one (see bottom of Figure 4.4). The meaning of And/Or can be defined by selecting a special operator. The fuzzy operators we implemented so far in the library are: as t-norm the *minimum* operator, as t-co-norm the *maximum* operator, two parameterisable operators *fuzzy_and* and *fuzzy_or*, and an associative compensatory operator [12].

4.1.6 Evaluation

Evaluation is done for a single variable by pressing the evaluate button for the selected variable. All rules that contribute reasons why this one variable should get a certain fuzzy value are searched among all rules in the set of

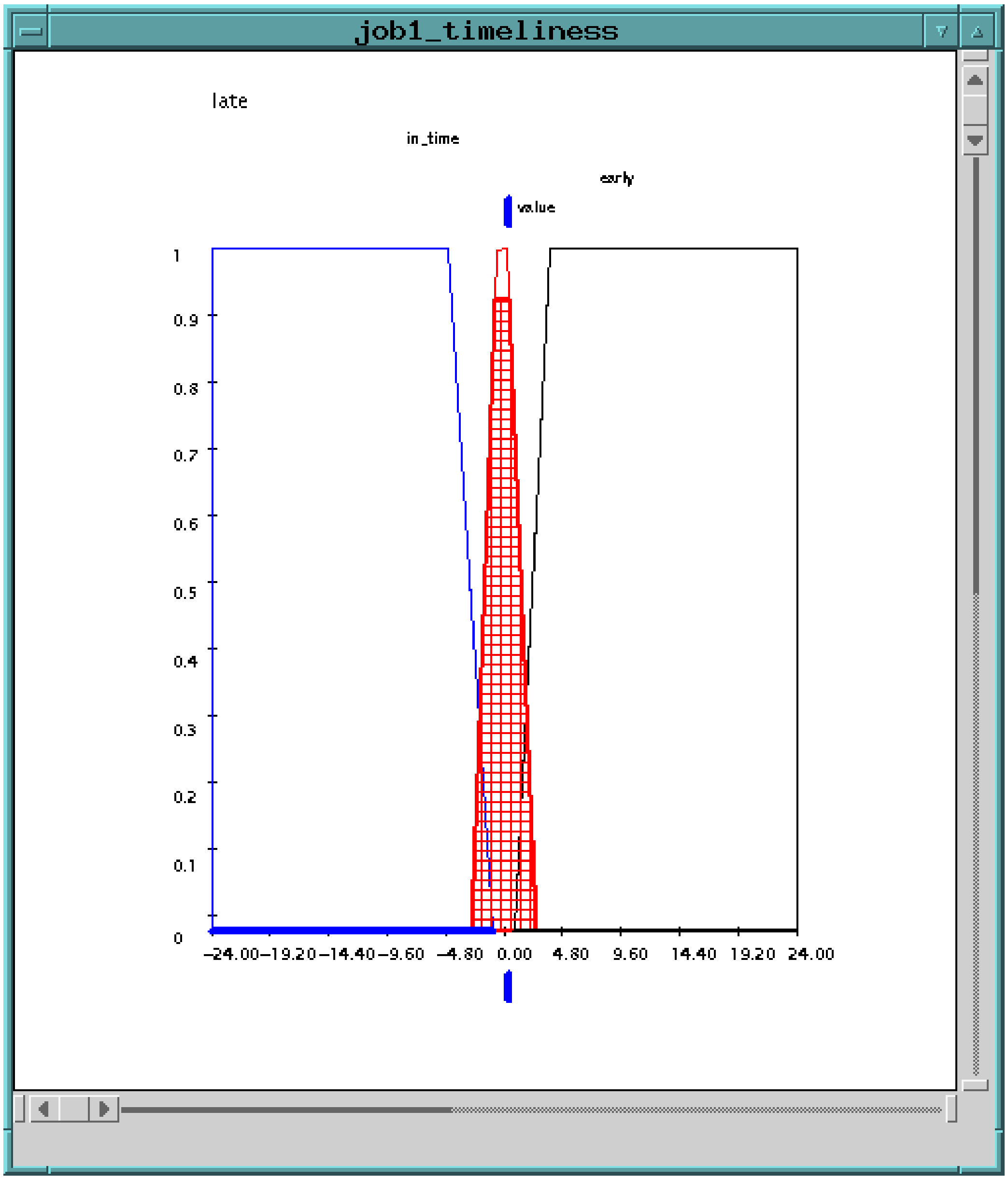


Figure 4.5: InterFLIP++ membership function window

rules. For all variables the same is done recursively. Basic variables have to be instantiated with a value by dragging the value mark in the membership window. So all the static knowledge and all the known relations for the resulting variable are searched in the hierarchy and then evaluated. During the evaluation, all the uncertainties about variables and the uncertainties about the relations are used in the calculations. The rules can be evaluated by applying fuzzy methods on the quantitative representations of the variables. As a result, a degree of membership is attached to each possible term of the resulting variable. It is shown in the membership function window. For InterFLIP++ control applications, this representation of the result can be interpreted in different ways to get some crisp value for controlling machinery. To influence the inference mechanism different operators can be selected (see 4.6). What is the meaning of the different types of operators? In the following all operator types are explained.

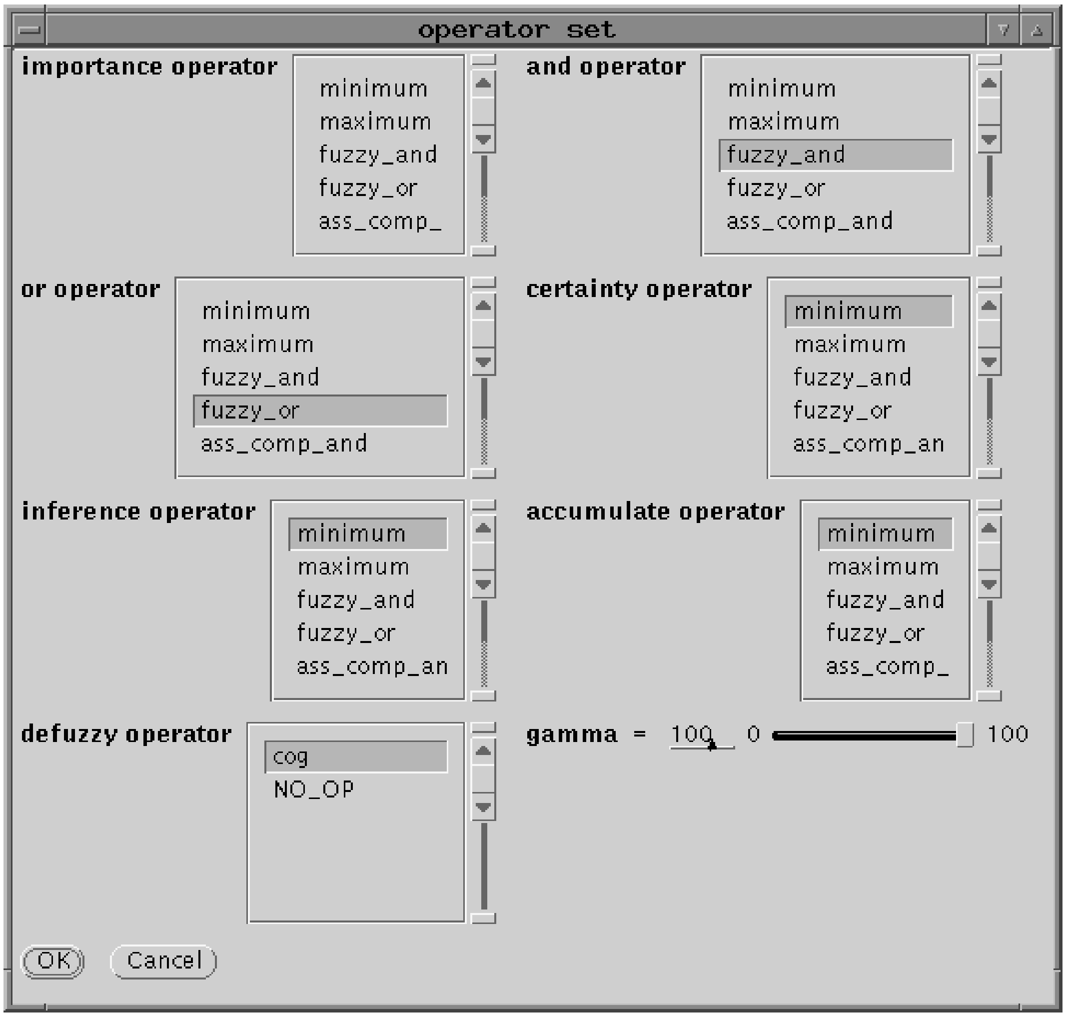


Figure 4.6: operator window

- AND operator
  This very common used aggregation operators are used to define how a *AND* operator in a rule is defined.

- OR operator
  The OR operator is used in rules, too. It can be defined using an aggregation operator, like the AND operator.

- Importance operator
  Every fuzzy variable has an importance. This data has to be recognised during an evaluation process.

- Certainty operator
  Like a fuzzy variable a rule has a measure for its importance, too. This is the certainty value at the end of each rule. The latter operator defines the its importance during evaluation of a rule.

- Inference operator
  In the inference step of the evaluation process this operator influences the way the result of the premis is calculated to the conclusion of a rule.

- Defuzzify operator
  The first result of evaluation of fuzzy variables are degrees of membership for all terms of output variables. For applications where a single real number is needed as an result (e.g., fuzzy control tasks) this is done with the help of the defuzzify operator.

Visualisation of evaluation-graphs

The process of fine-tuning membership functions and fuzzy rules is a important step on the way to define a fuzzy system, for example a controller. After some modification a single evaluation is not enough to detect major changes. A number of evaluations have to be done. InterFLIP++ supports easy evaluation and visualises the results in two and three dimensional graphs.
Figure 4.7 shows the InterFLIP++ window where the user controlls evaluation and visualisation. The top choicebox contains all output variables. Output variables are variables on the right side of a rule. For the selected output variable the user can define the range one or two input variables. All other input variables for the output variable are constants. In the middle of the window in Figure 4.7 there is a listbox to edit values of these constants.

It is possible to generate four outputs:

- A '*dat*'-file is generated. Figure 4.8 shows the *dat*-file for the example in Figure 4.7. Defuzzified output values are evaluated for all input values in the defined range and a given step.

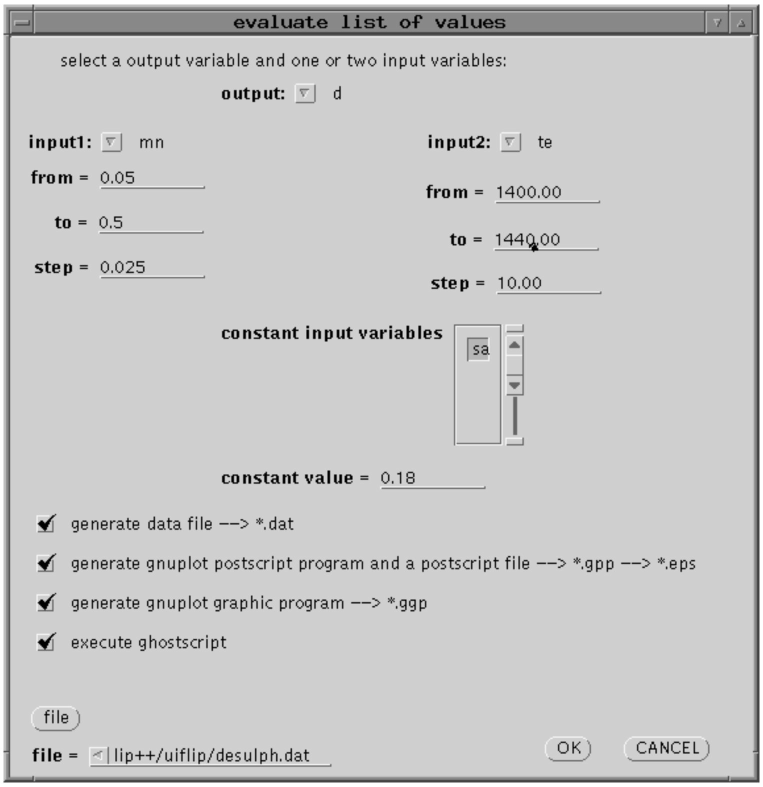


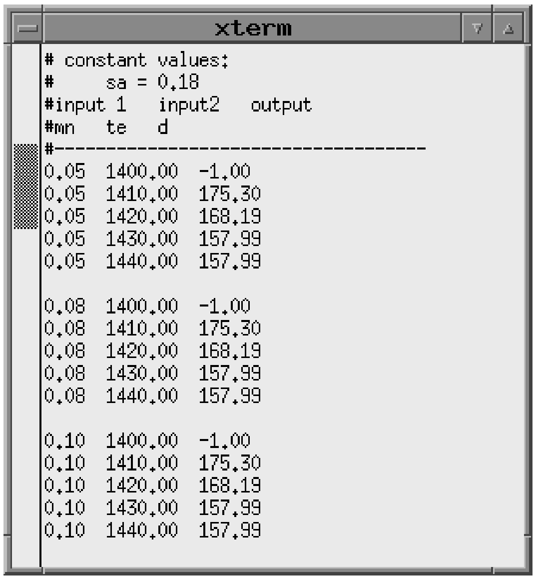
Figure 4.7: InterFLIP++, how to visualise evaluation graphs

- Produce a postscript file with two or three dimensional graphs of the *dat*-file. Gnuplot, which is a command driven interactive function plotting program, loads the *dat*-file and produces the postscript file. Gnuplot is executed automatically. The script that Gnuplot uses has the extension *gpp* (Gnuplot Postscript Program).

- The Gnuplot produced graph can be shown directly by selecting the third item. Another script with the extension ggp (Gnuplot Graphic Program) executes Gnuplot to visualise the graph. The left picture in Figure 4.9 shows a Gnuplot graph for the example in Figure 4.7.

- Ghostscript can be executed to show the previously produced postscript file. Ghostscript can be used to print the graph. Ghostscript can be executed to show the postscript file (using the *gpp* script) and to print it, see the right picture in Figure 4.7 shows it for the latter example.

All generated graphs exist simultaneously in different windows and it is therefore easy to compare them with each other.



```
# constant values:
#     sa = 0.18
#input 1   input2   output
#mn   te   d
#-----------------------------------
0.05  1400.00  -1.00
0.05  1410.00  175.30
0.05  1420.00  168.19
0.05  1430.00  157.99
0.05  1440.00  157.99

0.08  1400.00  -1.00
0.08  1410.00  175.30
0.08  1420.00  168.19
0.08  1430.00  157.99
0.08  1440.00  157.99

0.10  1400.00  -1.00
0.10  1410.00  175.30
0.10  1420.00  168.19
0.10  1430.00  157.99
0.10  1440.00  157.99
```

Figure 4.8: evaluation graph, dat-file

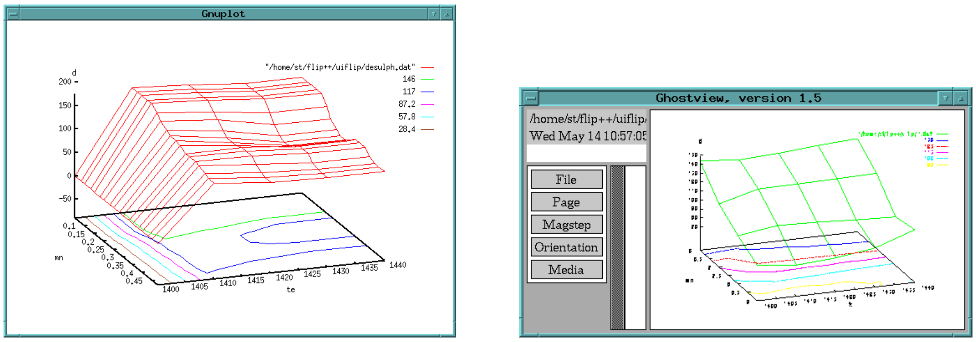


Figure 4.9: Visualisation with Gnuplot(left) and Ghostscript(right)

It is easy to adapt the automatically produced Gnuplot scripts to make specialised graphs, e.g., lay one graph upon another to see changes more easily.

4.2 Matrix interpolation with InterFLIP++

4.2.1 Introduction

In the following I show how to interpolate a variable that is defined by a matrix. The chosen example is a model to calculate the desulphurisation amount in a blast-furnace. The desulphurisation behaviour can be described by a linear equation. The linear equation can be defined with two parameters. The function of these parameters is described by a matrix. The function of one of these parameters, D, is interpolated in this example. The fuzzy model interpolates D using FLIP++ and InterFLIP++.

4.2.2 The desulphurisation process

Pig-iron incorporates too much sulphur as an input for a steel making plant. In the desulphurisation process a powder which is mostly calciumcarbid is added to the pig-iron. The basic chemical reaction can be described like this:

$$CaC_2 + S = CaS + 2C$$

The resolving CaS of the reaction is part of the dross swimming on top of the pig-iron. The latter reaction is done in a special plant. Pig-iron is delivered by rail. CaC_2 is blown into the pig-iron through a pipe. The hole process takes about 30 minutes depending to the target value of sulphur.
The example control application deals with the problem how much CaC_2 is needed to reach a certain sulphur level given the mass of , temperature of the (TE), the percentage of sulphur (SA) and the percentage of manganese (MN). A number of mathematical models describes this problem. Here a simple linear equation is used to determine the only non measurable parameter of the model equation. A matrix defines the parameters that characterise the equation. This section shows how to model a fuzzy inference system for one of these parameters.

4.2.3 Model definition with InterFLIP++

This section describes the example (in other words the matrix) and how to build an fuzzy set model to interpolate in the matrix using InterFLIP++/FLIP++. For further information about fuzzy set theory see [6] and [7].

The matrix

The model has three input-variables: percentage of sulphur SA, percentage of manganese MN and the temperature TE. The output-variable D is a parameter of a linear equation to characterise the desulphurisation mean value (see Table 4.1).

SA	MN	TE	D	output terms
$\leq$0.01	$\leq$0.2	$\leq$1420	185.2	d32
		[1420,1450)	165.14	d30
		[1450,1480)	148.14	d23
		>1480	163.33	d29
	(0.2,0.3]	$\leq$1420	152.53	d25
		[1420,1450)	159.0	d26
		[1450,1480)	128.2	d19
		>1480	120.0	d12
	(0.3,0.4]	$\leq$1420	129.8	d20
		[1420,1450)	125.5	d16
		[1450,1480)	120.0	d13
		>1480	105.9	d7
	>0.4	$\leq$1420	117.9	d10
		[1420,1450)	102.5	d5
		[1450,1480)	87.7	d2
		>1480	127.5	d18
>0.01	$\leq$0.2	$\leq$1420	177.3	d31
		[1420,1450)	161.3	d28
		[1450,1480)	151.0	d24
		>1480	160.5	d27
	(0.2,0.3]	$\leq$1420	141.0	d22
		[1420,1450)	102.5	d3
		[1450,1480)	121.9	d15
		>1480	116.8	d9
	(0.3,0.4]	$\leq$1420	130.0	d21
		[1420,1400)	121.6	d14
		[1300,1425)	114.8	d8
		>1425	105.0	d6
	>0.4	$\leq$1420	118.0	d11
		[1420,1450)	102.6	d4
		[1450,1480)	87.2	d1
		>1480	126.9	d17

Table 4.1: The matrix

Parameter sets and fuzzy variables

The given matrix leads to a fuzzy model with three input variables (sa,mn,te) and an output variable (d). The first step using FLIP++ is to define a parameter set as a prototype of a fuzzy variable. A range of values and a term list are the most important facts.

The matrix separates *sa* in two areas. Each area will be a term, i.e., *sa* has two terms called *small* and *big*. Other input variables are created in an analogous way. Therefore *mn*, (see Figure 4.11) and *te* are variables with four terms each.

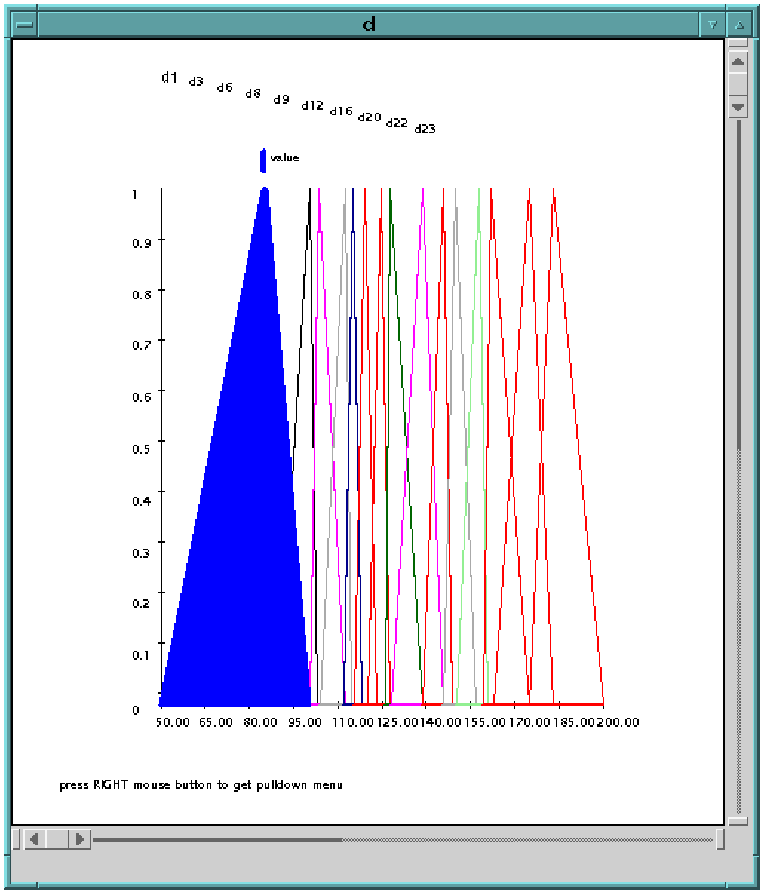


Figure 4.10: membership functions of d

The output variable (d, see Figure 4.10) is more complicated. Each value of d in the matrix can be used to create a term for the fuzzy variable d in the following way: The membership functions are trapezoidal. The value is the maximum of a trapezoidal function. The more its base range overlaps with

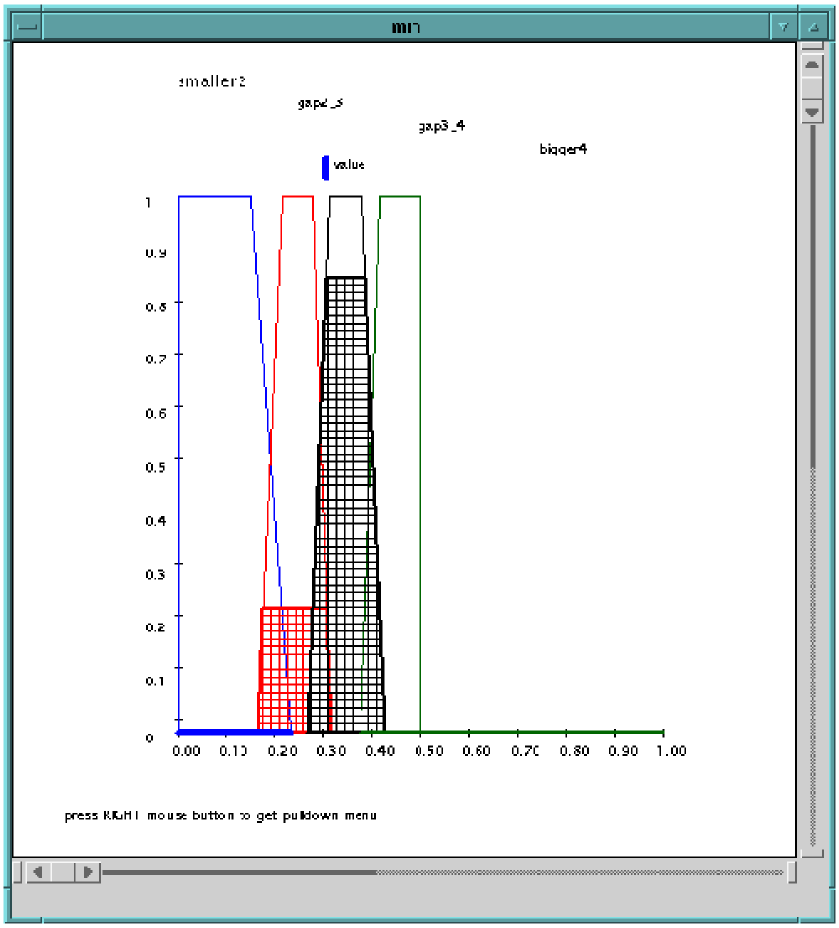


Figure 4.11: membership functions of mn

sa	mn	te
small	smaller2	cold
big	gap2_3	warm
	gap3_4	hot
	bigger4	very_hot

Table 4.2: Fuzzy input-variables and their terms

the base ranges of neighbouring terms, the smoother the output variable will be.
The next step is to reduce the number of terms for *d*. D-values are sorted by increasing values which means that d25 is the 25th greatest value. Equal or similar values can be combined and be represented with a single term. Thus the number of terms can be decreased. This increases

clearness of the output variable and the fuzzy rules, because the number of values decreases too. In this paper two fuzzy models are compared. One expanded fuzzy system with 32 terms (see Table 4.1) for the output variable (d1) and a second one with 15 terms. Which d-values are matched to a mean value is shown in Table 'List of eliminated terms' in 4.2.3.

mean	eliminated terms
d1	d2
d3	d4,d5
d6	d7
d8	
d9	d10,d11
d12	d13,d14,d15
d16	d17,d18,d19
d20	d21
d22	
d23	
d24	d25
d26	d27,d28
d29	d30
d31	
d32	

Table 4.3: List of eliminated terms

If all parameter sets for fuzzy variables are defined, the fuzzy variables themselves can be build simply by giving them a name. A first membership function for each term is build automatically. Each membership function has to be refined to match the needs of the modeling more exactly.

Refine membership functions

All membership functions of input variables have a trapezoidal shape, whereas membership functions of output variables are triangular. Elimination of overlapping membership functions does not greatly influence output characteristics. Massive overlapping base ranges of membership variables cause an increasingly smooth output variable, see Figure 4.15. On the other hand, non-overlapping neighbouring membership functions cause more rectangular output functions.

The definition of variables with membership functions makes ist possible to change the behaviour of the matrix interpolation locally at any point. This makes the model easier to understand and to use. Manipulation of the overlapping in of one input dimension, changes the smoothness of the interpolation. This technique can be used for multi-dimensional models. Another method to influence the interpolation locally could be to manipulate with a mouse points in a graph as the one shown in Figure 4.12. But this is restricted to two and three dimensional graphs, e.g. only one or two input variables. FLIP++ allows to handle any number of input variables.

Rules

Rules associate input values to output values in a very understandable way for humans.

e.g.: line five of the Table 4.1:

```
IF sa IS small AND mn IS gap7_8 AND te IS cold THEN d IS d25.
```

Evaluation

If the fuzzy set model is defined, membership functions have been refined, and all rules are OK, then the system can be used. Input values for all variables can be set in the membership function window. The instantiated function shows which term matched the input value to what grade. After selecting the output variable, the output value can be evaluated. An instantiated function for the output value and a defuzzified (crisp) value are produced using the minimum t-norm as an AND-operator.

4.2.4 Results

I did not focus primarily on optimal fitting, e.g. by minimizing the sum of error-squares in four dimensional space. Indeed, the goal of using FLIP++ to model the matrix interpolation was to facilitate manipulation of the interpolation behaviour of the system by human operators. The deeper reason is that it is actually possible to change this behaviour locally at any points, whereas it it seems impossibe to define some theoretical optimal curve.

Both fuzzy models (with 15 and 32 terms) are evaluated with the same input values. In order to get a three dimensional graph out of a four dimensional matrix the variable with the fewest number of terms was treated as a constant (sa=0.7). The rectangular shaped graphs in these figures are build out of matrix data.

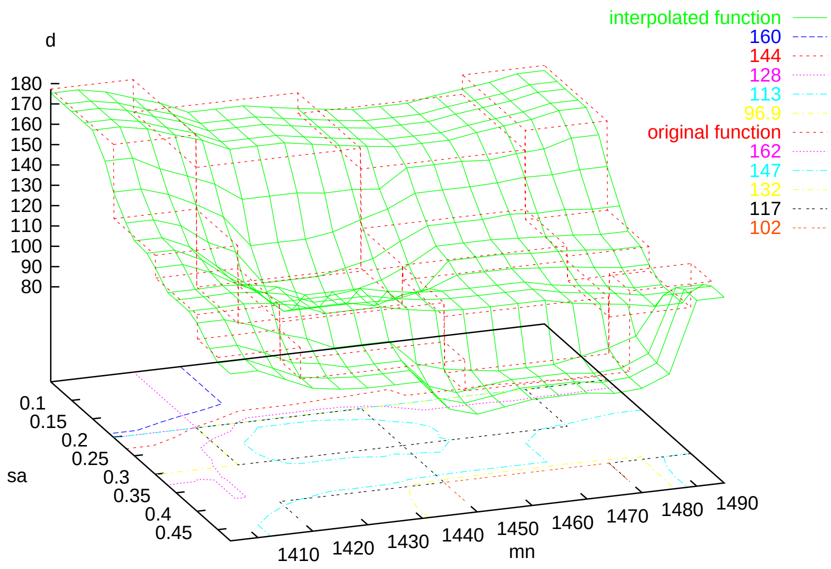


Figure 4.12: output variable da has 15 terms

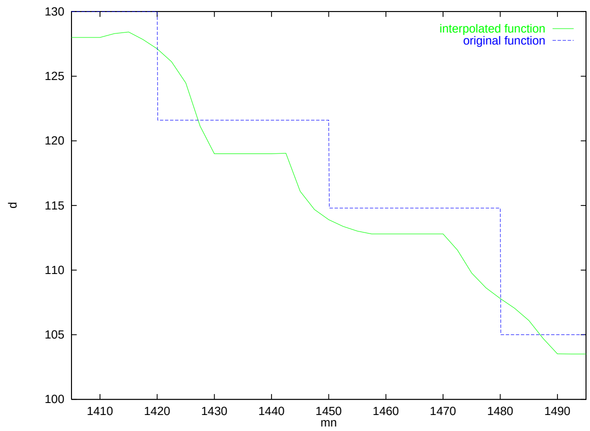


Figure 4.13: d has 15 terms (sa=0.16, mn=0.35)

Figure 4.13 shows a two dimensional graph for the output variable d. Two input variables are constants (sa=0.16, mn=0.35). Hence one

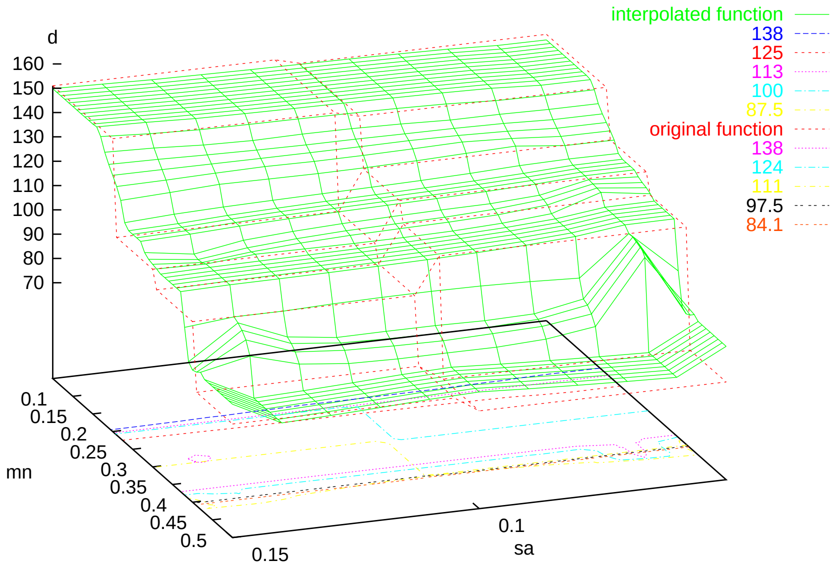


Figure 4.14: d has 15 terms, TE = 1460

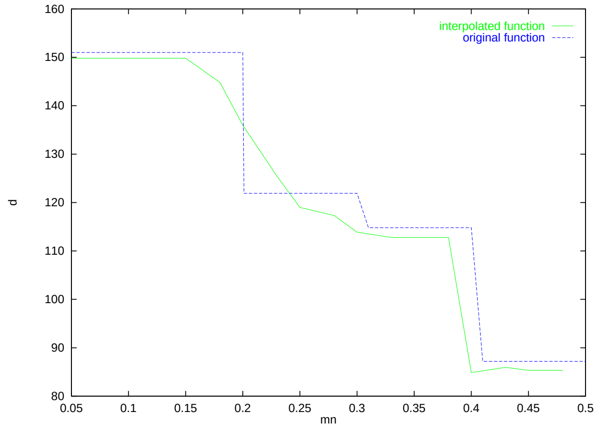


Figure 4.15: d has 15 terms, TE = 1460, SA = 0.15

graph represents the matrix from the 25th line to the 28th line and has 4 steps. The other graph in Figure 4.13 shows the defuzzyfied values (the interpolated function).

Very interesting parts of the two dimensional graphs are these parallel to the x-axis. There nothing but a temperature (te) term matches. Its width

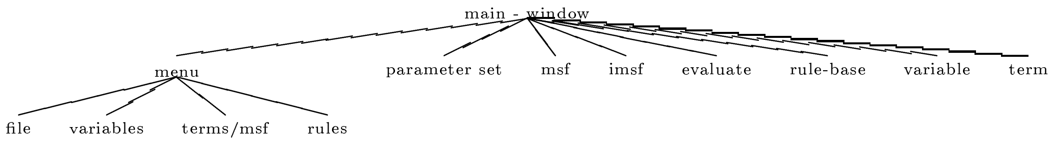
can be controlled through membership functions. The fewer these functions overlap the larger becomes the part of the function parallel to the x-axis. This means the function will look more rectangular.
The 32-term system is equal to the matrix-graph in its parallel part because very exact matrix data is used. The 15-term system, with its fuzzier data, touches the matrix graph in the x-parallel to a lesser degree because it does not use exact matrix data in most rules.

4.2.5 Integration in the desulphurisation model

The second parameter, which is needed to define the linear equation for the desulphurisation process can be modelled like the first one. The fuzzy model can be stored in an ASCII file and reused in a small program implementing the two equations describing the desulphurisation process.
First the InterFLIP++ modeled data has to be loaded with the appropriate flip-method. For all new input values the fuzzy inference system has to be evaluated with FLIP++-methods and the defuzzified results can be used directly in the desulphurisation equations.

4.3 UI event-flow description

4.3.1 Main window



1. Menu

   (1) File
   Initiated by: menu-item: file
   Description: loading and saving functionality
   Mode ends: ok or cancel button in file handling window

   (2) Variables

      1. ADD
      Initiated by: menu-item: variables - ADD
      Description: in a new window a name for the new variable

can be edited and a parameter set can be chosen out of a choice box.
Mode ends: ok or cancel button in add-a-new-variable-window

2. EDIT
   Initiated by: menu-item: variables - EDIT
   Description: the name of the variable that is selected in the variable list can be edited in a new window
   Mode ends: ok or cancel button in variable-name window

3. DELETE
   Initiated by: menu-item: variables - DELETE
   Description: the variable that is selected in the variable list is will be deleted
   Mode ends: after selected variable is deleted

(3) Terms/msf

1. ADD a node
   Initiated by: menu-item: variables - ADD
   Description: add-a-new-variable-window
   Mode ends: ok or cancel button in add-a-new-variable-window

2. Set value
   Initiated by: menu-item: variables - set value
   Description: set a value in membership function window
   this is only necessary if this is the first time a value is to be set.
   Mode ends: after selecting the menu item

3. DELETE a node
   Initiated by: menu-item: variables - DELETE a node
   Description: delete-a-node mode in the MSF-window is enabled
   Mode ends: after selecting the menu item (how the delete-a-node mode ends is described in the MSF-window part below)

4. Choose a colour
   Initiated by: menu-item: variables - choose a colour
   Description: a new colour can be selected in window with a choice-box to select a colour
   Mode ends: ok or cancel button in colour-window

(4) Rules

1. Update edits
   Initiated by: menu-item: rules - update edits
   Description: updates edited rule base
   Mode ends: after selecting the menu item

   A. Show all rules
      Initiated by: menu-item: rules - show how much rules-show all rules
      Description: changes rule base window mode
      Mode ends:after selecting the menu item

   B. Show only rules for selected variable
      Initiated by: menu-item: rules - show how much rules-show rules for variable
      Description: changes rule base window mode
      Mode ends:after selecting the menu item

2. Change output rule style
   Initiated by: menu-item: rules-show output rule style
   Description: changes rule style to the other one (long or short style)
   Mode ends: by selecting the menu item

3. EDIT operator set
   Initiated by: menu-item: rules / EDIT-operator-set item
   Description: In the operator set window all available operators can be selected
   Mode ends: Ok or cancel button in EDIT-operator-set-window

2. Parameter set
   Initiated by: radio box
   Description: changes rule style to the other one (long or short style)
   Mode ends: by selecting the radio box a second time the parameter window closes

3. MSF
   Initiated by: radio box
   Description: If the IMSF-radio box in not selected, the MSF-window shows only MSF graphs. If MSF-window is already on, then it shows MSF-graphs in addition to IMSF graphs.
   Mode ends: by selecting the radio box a second time the MSF window closes or it shows only IMSF graphs.

4. IMSF
   Initiated by: radio box
   Description: If the MSF-radio box is not selected, the MSF-window shows only IMSF-graphs.
   Mode ends: by selecting the radio box a second time the MSF window closes or it shows only MSF graphs.

5. Evaluate
   Initiated by: button
   Description: Enables evaluation of selected variable in the variable list.
   Mode ends: If evaluation is done and results are shown.

6. Rule-base
   Initiated by: Some editing in the rule base box.
   Description: Changes rule base with wxTextWindows functionality (Copy and Paste and all other common edit functionality), but changes are not saved.
   Mode ends: by leaving the rule base box in the lower part of the window.

7. Variable list box
   Initiated by: variable in variable list box selected

Describtion: All terms for the variable are updated in the term list box. Parameters like a crisp value are updated. If show-only-Rules-for-variable mode is on then other rules will be shown. If MSF window is open then graphs for the variable will be shown.
Mode ends: If all output information is updated.

8. Term list box
   Initiated by: term in term box selected
   Deskribtion: If the MSF is on then all graphs for the new term are marked.
   Mode ends: If all output information is updated.

4.3.2 Parameter set window

parameter set - window (edit mode)

file handling   add set   delete set

1. Parameter set

2. Edit mode
   Initiated by: default mode in parameter window
   Description: active parameter set can be edited
   Mode ends: ok or cancel button in file handling window

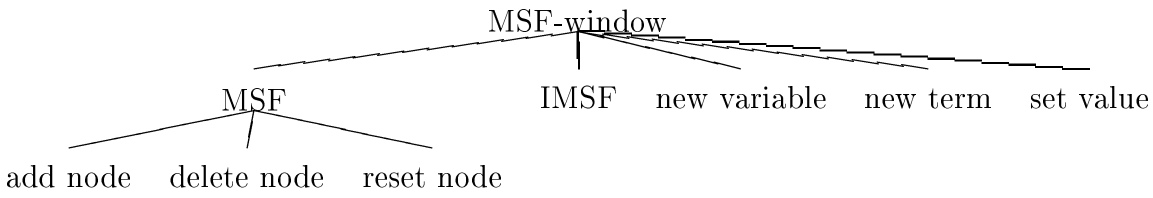
3. File handling
   Initiated by: file button
   Description: WxWindows file-handling object
   Mode ends: ok or cancel button in file handling window

4. Add set
   Initiated by: add button
   Description: a form in a special window has to be filled out
   Mode ends: ok or cancel button in add window

5. Delete set
   Initiated by: delete button

Description: the selected parameter set in the list box will be deleted
Mode ends: after delatation of the object

4.3.3 Membership function window (MSF-window)



1. MSF window

   (1) MSF shown
   Initiated by: radio box in main window
   Description: If no MSF-window is on, one is opened showing only MSF graphs.
   If MSF-window is on, MSF-graph are shown in addition to IMSF graphs. Mode ends: by selecting the radio box a second time, the MSF window closes or it shows only IMSF graphs.

      1. Set add node mode
         Initiated by: Pressing right mouse button and selecting add node item in the button menu. Note: only if IMSF are not shown.
         Description: The left mouse button has to be pressed on the right place in the graph to create a new button there.
         Mode ends: After creating a new node, the default mode is active.
      2. Delete node mode
         Initiated by: pressing right mouse button and selecting delete node item in the button menu. Note: only if IMSF are not shown.
         Description: The left mouse button has to be pressed in the area of a node of the selected term which shall be deleted.
         Mode ends: after deleting a node, the default mode is active
      3. Reset node mode
         Initiated by: pressing the left mouse button in the area of a

node of a selected term. Note: only if IMSF are not shown.
Description: The selected node can be dragged to another place. Note: FLIP++ has some restrictions to allowed MSF shapes, but this is done automatically.
Mode ends: after releasing the button, the default mode is active

(2) IMSF shown
Initiated by: radio box in main window
Description: If no MSF-window is on, one is opened showing only IMSF graphs.
If MSF-window is on, IMSF-graphs are shown in addition to MSF graphs. Mode ends: by selecting the radio box a second time the MSF window closes or it shows only MSF graphs.

(3) Set value mode
Initiated by: dragging the value sign
Description: It is the default mode. The value sign can dragged to another input value.
Mode ends: after dragging the value sign

(4) Change variable
Initiated by: variable list box in main window
Description: New IMSF graphs and/or MSF graphs will displayed.
Mode ends: after showing the new data.
Note: only if IMSF are not shown.

(5) Change the editable, specially displayed term
Initiated by: term list box in main window
Description: If only MSF are shown the new term will be specially and is editable, but if IMSF are also shown, the graphs for the term will be shown.
Mode ends: after showing the new data.
Note: only if IMSF are not shown.

4.4 Implementation

In this section some implementation aspects and the object-oriented framework will be shown. The interface is based on WxWindows, which is a multi-platform, public domain interface of the Artificial Intelligence Applications Institute, University of Edinburgh. Note that all class names which

begin with wx... are WxWindows defined classes.
Frames, canvases and other WxWindows-defined classes are used as parent classes for classes enriched with special functionality used in the GUI. One of the most important special functionality is event driven, but a special type, e.g., window with common data and design, is grouped in a class, too.

4.4.1 Parameter window

The next tree shows the class inheritage tree of *Parameter_window*.

```
wxObject
   |
wxWindow
   |
wxFrame
   |
 MyWin
   |
  ...
   |
Win_def_file
   |
ParameterWindow
```

wxObject is root-parent class of (nearly) all wxWindow classes. WxFrame has the functionality to define a frame window, but to put widgets into the frame a subclasses are used. MyWin is the lowest level InterFLIP++-window class. It handles childwindows and has a special closing mechanism. Win_def_file adds methods to provide file handling. The class Parameter-Window has to create all needed wxItems like a list box or a slider and and put it to the right place. This job can be done with a builder, too. Its information is implemented in the constructor method of the class. All event driven operations are implemented with callback procedures.

4.4.2 Fuzzy_set_canvas

A different type of window compared to wxFrames are wxCanvas classes. The Fuzzy_set_canvas provides methods to draw MSF and manipulate it.

wxObject
|
wxWindow
|
wxCanvas
|
fuzzy_set_canvas

fuzzy_set_canvas has wxCanvas as parent class and can therefore inherit its functionality. Because fuzzy_set_canvas is used to visualise and edit fuzzy membership functions, certain methods are needed. Whereas a GUI builder can do a good part of the job in the case of the parameter window, especially the wxBuilder, most of the implementation work has to be done by hand. wxCanvas provides methods to

- Visualise flip membership functions
- FLIP++ instantiated membership functions
- Nodes for active terms which are editable
- Visualise and set input values
- Intern and extern (with other windows) event handling
- Translate and store edited graphical data to flip data
- Scaling, captioning
- ...

4.4.3 Rule_text_box

Another central flip data are rules. The class *rule_text_box* provides methods to visualise and edit rule base data. Its parent class is wxTextWindow which has special methods to manipulate text.

wxObject
|
wxWindow
|
wxTextWindow
|
rule_text_box

The main functionality *rule_text_box* has is its interpreter. The interpreter must handle the following features

- Different styles to visualise a rule are important to show the rule in a more general view or in a detailed style.

- If the interpreter detects a syntax error, the user has to be provided with information where it is and how to correct it.

- Because a rule-database can be very long, methods have to be provided to partitionate the database. One possibility, which is implemented here, is to use the latter part of a rule to group it.

Chapter 5

Other fuzzy logic tool interfaces

5.1 Mathlab

Mathlab is an interactive program for numeric computation and data visualisation of linear systems equations from TheMathWorks, Inc. I tested MathLab 4.2, especially its expo part where I concentrated on the fuzzy logic toolbox. There are a number of examples

- Fuzzy c means - This is a clustering task.
- Noise cancellation. This task uses ANFIS (Adaptive Neuro Fuzzy Inference System) to solve it. Input variables are shown in 2D and 3D graphs. After the ANFIS training, output is shown like input variables are.
- Chaotic time series
- Ball and beam control. This is a control task resolved with a neuro fuzzy tool, a model of a ball rolling frictionlessly an a beam. A motor generates a torques to tilt the beam in order to send the ball to a desired location. A fuzzy controller is designed to generate appropriate torque to achieve the control task. A block diagram shows the fuzzy controller control system. When running the simulation parameters can be modified and results seen in an animation window. The simulating has to run in SIMULINK.
- Water level control. This example will be explained in more detail later.
- Fuzzy grain scheduling

I will describe the fuzzy logic interface of the water level control example (see Figure 5.1).

5.1.1 Water level control

The main window shows a block diagram of the example: 2 input variables (level and change) and an output variable (value) (see Figure 5.2). On the bottom of the window each fuzzy operator (and, or, implication, aggregation, defuzzification) has its own choice box to select an appropriate operator type. It is possible to edit variable names in this window. The range of the variables is shown, but not editable here (there exists a special sort of window for this). A top-down menu has 3 items in the first level.

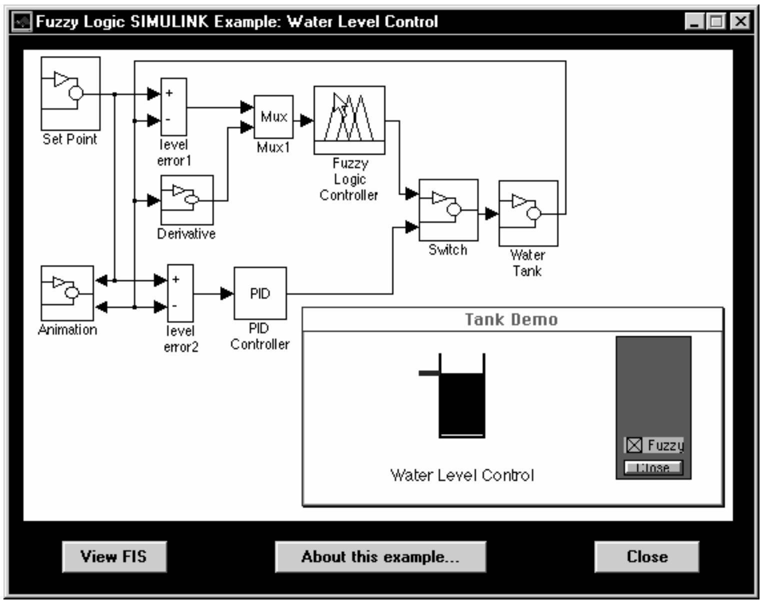


Figure 5.1: Mathlab - water level control

1. File (save, load ..)
2. Edit (Add input, Add output, Remove, Undo)
    - Add input
    - Add output
    - Remove It is possible to remove a selected variable or
    - Undo a last action.

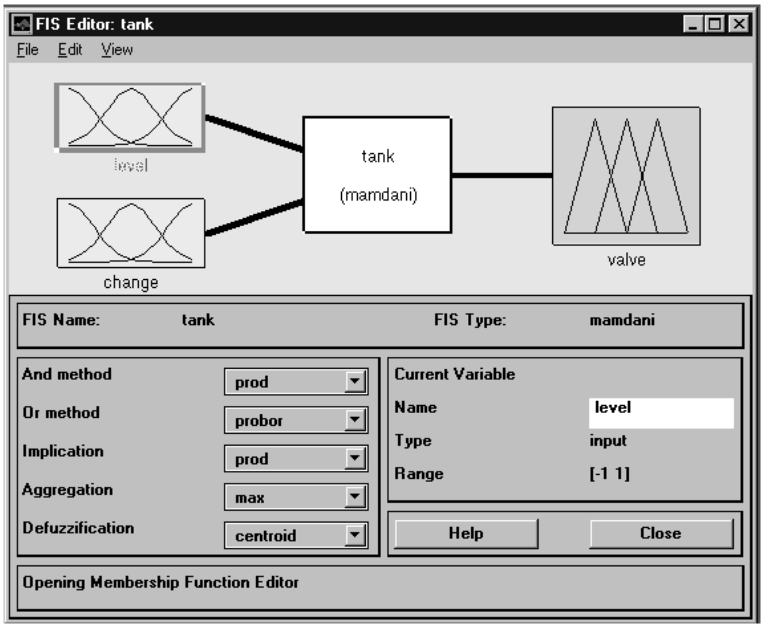


Figure 5.2: Mathlab - FIS editor

3. View (Edit membership, Edit rules, View rules, View surface)

   - Edit membership This is a window (see Figure 5.3) showing all variables (input and output). By clicking a certain variable its membership functions of all terms are shown in a 2D graph. The use of this 2D canvas of membership functions is to select a term - it is not possible to change a membership function with direct manipulation in the graph. Parameters in textual form can be edited.
     - Function type
       There are a number of function types (trapezoid, sigmoid, gauss, ..) which can be selected using a choice box.
     - Parameters
       Parameters which describe a membership function,
       e.g., (-0.15 0 0.5) for a trapezoidal function. They are is editable in a text box. Of course the parametrical and visual representation changes with the function type.
   - Edit rules. The rules are listed in a multiline scrollable textbox without hierarchy. But tree formats or rule representation can be selected in a choice box.
     - Verbose, e.g., 1.IF (level is low) then value is openfast (1)

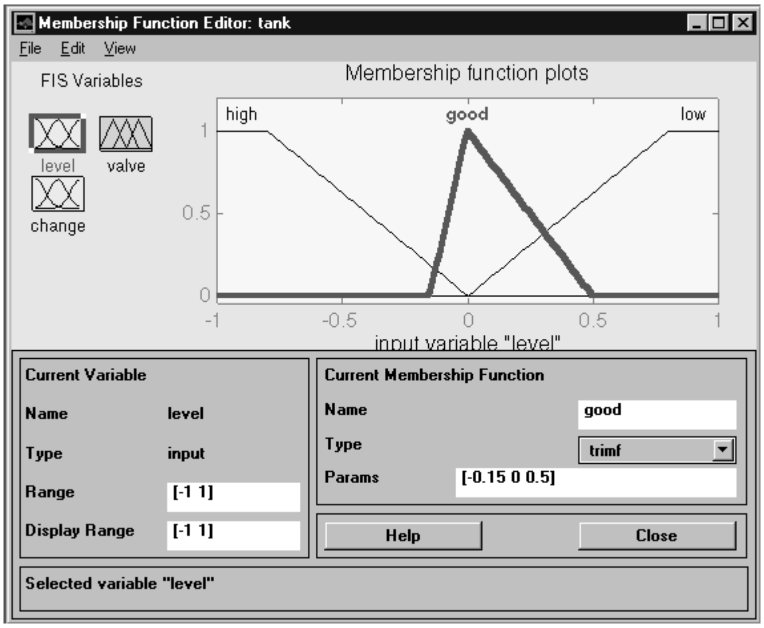


Figure 5.3: Mathlab - membership function window

– Symbolic, e.g., 1.(level==low) ⇒ (value = openfast) (1)
– Index, e.g., 1. 3 0.5 (1):1

It is possible to choose a language (English, German, French). For German language, rule one will show like this in verbose format:
1.Wenn (level ist low) dann value ist openfast (1)

5.2 FuzzyTECH

I tested fuzzyTECH4.0 from Information Software Corporation.

The main window shows a scrollable blockdiagram of the model. One can see input variables on the left, a symbolised ruleblock in the middle, and the output interface on the right side. Above the topdown menu there are buttons for main menu items like New Project, Create Variable, Help, …(see Figure 5.4). The menu has a 2 layer hierarchy:

- File (New, Save,Open, ..) - file handling functionality
- Edit - add and remove variables and rules
  - CreateNewVar Input Parameter: Name, Number of terms:
    If the button edit is pressed a new windows opens to edit membership functions (trapezoid functions only) of the new variable.

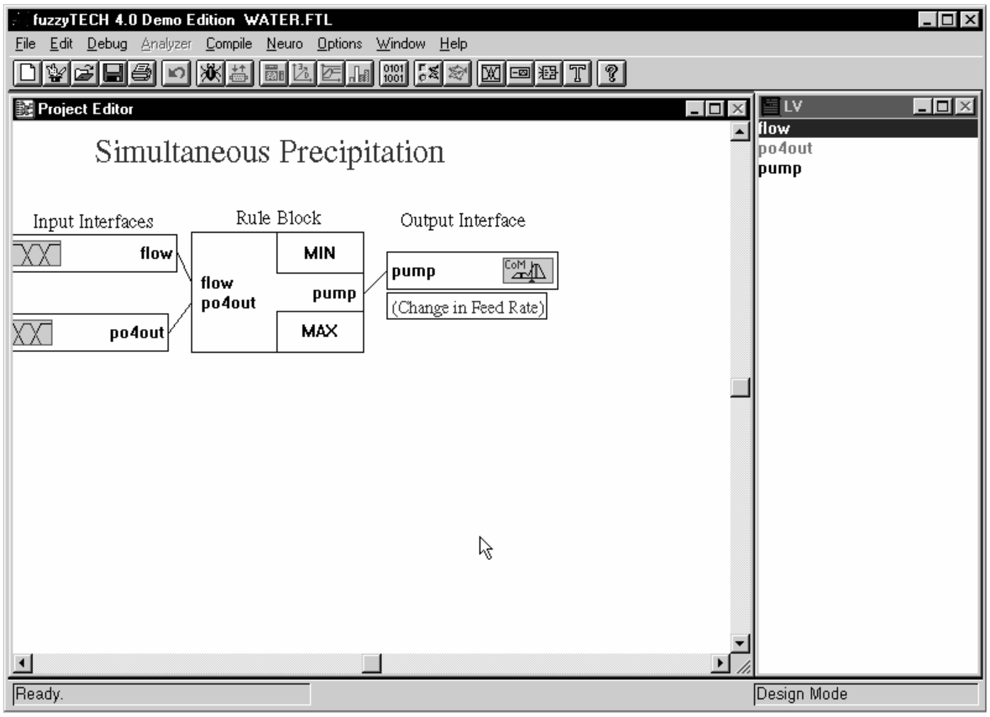


Figure 5.4: FuzzyTECH mainwindow

The membership manipulation window contains a term list, a 2D canvas, where the terms are graphically visualised (see Figure 5.5) and a x/y coordinate display, showing the actual position of the mouse cursor in the membership graph (canvas). By clicking a term in the term list a term function in term graph is selected and each corner point of the chosen trapezoid function can removed with drag/drop actions. A MSF can have different shapes (see Figure 5.6). other options available by clicking a help sensitive button

* Grid on/of
* Select next term (direct selection by clicking is possible too)
* Select previous term
* Create term (two types of functions: T-shape (trapezoid), S-shape (spline like function), but tree corner points like trapezoid ones (left corner, highest point, right corner), the shape of S-shape function can be influenced via parameter).
* Show term listbox on/off
* Create inverse term

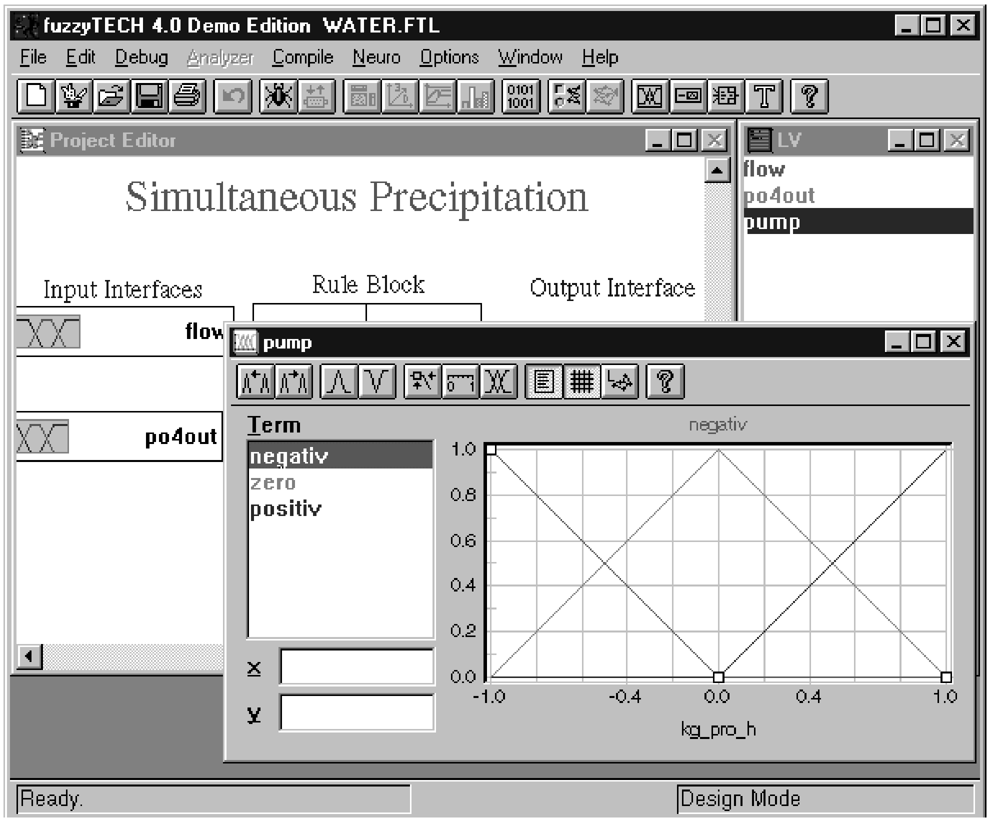


Figure 5.5: FuzzyTECH membership window

  * Grid resolution
  * Convert all terms to standard membership function - all functions will be trapezoid with equal shapes and equal distance form each other.

- Duplicate variable
- Create Interface
- Create RuleBlock
  A ruleblock is defined by selection input and output variables and aggregation and result aggregation operators. The 3D operator plot to show the influence of operator parameter changes. Each newly created ruleblock has its representation in the block diagram.
  Figure 5.7 shows how to add a new Rule to the system. The left part of the Figure shows how a simple rule with two input variables and one output variable is defined. An operator to aggregate

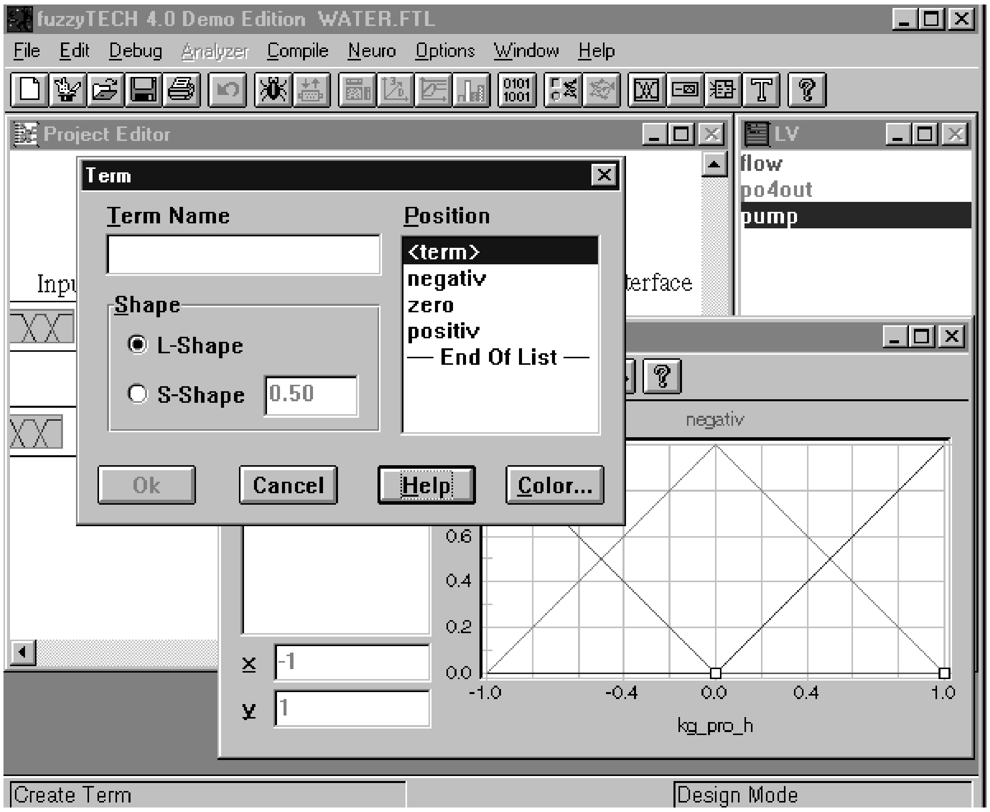


Figure 5.6: FuzzyTECH MSF shapes

the inputs can be defined interactively. In the right part of the figure a result aggregation operator has to be chosen. The result aggregation operator is used to aggregate fuzzy rules resulting in the same term of a variable. How the added rule is visualised is shown in Figure 5.8.

– Create Remark: This remark can dropped somewhere in the block diagram in the main window.

- Debug interactive debug mode allows to set input values (number manipulation) and a output value is shown immediately.

- Compile fuzzyTECH uses precompiler for implementation of the developed system to C, COBOL, .. .

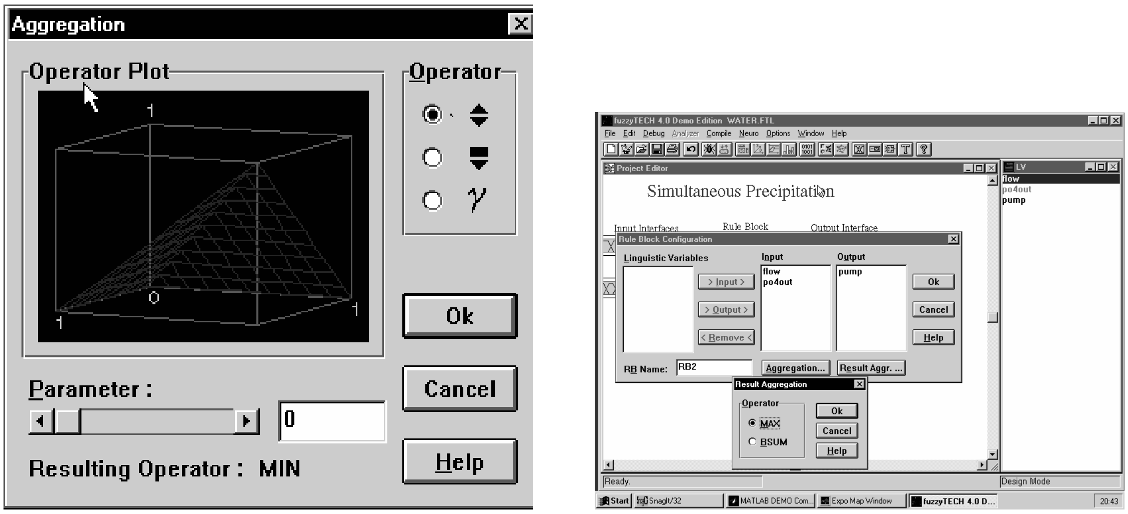


Figure 5.7: FuzzyTECH: right: aggregation operator, left: add a rule

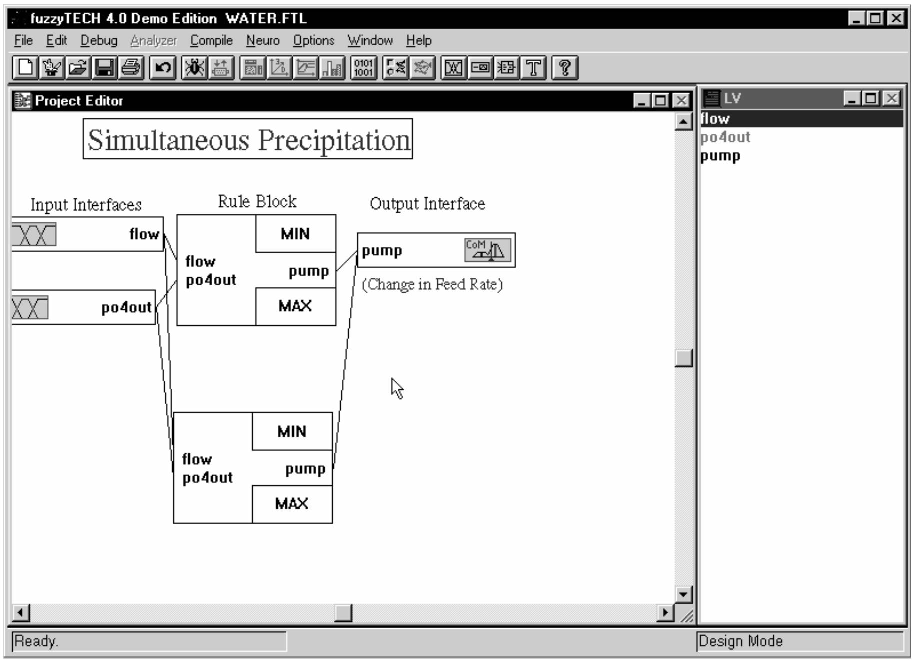


Figure 5.8: FuzzyTECH: if a rule is added ...

Chapter 6

Conclusion

In this thesis I described first how to identify a good GUI builder. I compared some GUI builders and described how to use them. In Chapter 3 I introduced the StarFLIP++ project. It is the main project which InterFLIP++ is a part of. InterFLIP++ uses FLIP++ methods, which are methods for fuzzy inferencing. Therefore I gave a brief introduction to fuzzy sets and fuzzy logic. The main part of this thesis is Chapter 4, where I showed how to use InterFLIP++. To demonstrate the capability to define real-world problems, a problem concerning desulphurisation in a steel making plant at VA Stahl Linz, Austria, is shown. The description of event-driven actions of InterFLIP++ and some implementation details finished the chapter.
In Chapter 5 I analysed other fuzzy logic GUIs.

By using InterFLIP++ it is easy to

- build a fuzzy system,
- refine it,
- and evaluate usable data.

The desulphurisation task of VA Stahl Linz provides me with real world data to define a fuzzy controller and use it.
InterFLIP++ has the following qualities:

- Consistency
  WxWindows guaranties the same look and feel for the whole application.
- Error management
  FLIP++ and InterFLIP++ share a common error management. That

allows the user to quickly react to problems.

- Transparency
  Fuzzy data is visualized and can be edited graphically and interactively.

- Feedback
  Direct manipulation of FLIP++ data like membership functions provides instant feedback. Changes of values are shown immediately during the whole design and evaluation process.

- Natural
  The dialog mimics the progess of the work.

- Predictable
  It is predictable how the system reacts within a specific context.

- Efficient
  Movements of hand and eye are minimized.

- Flexible
  Very different sort of tasks can be defined and evaluated with InterFLIP++.

InterFLIP++ provides the similar functionality as comparable fuzzy tools. However, InterFLIP++ was designed with cross platform portability in mind and uses only public domain software.

6.1 Future

- An improved visualisation of variable structures. This is important to get a general view of hierarchical dependencies of input - output variables reflected in the rule base. Unfortunately, big rule-bases are difficult to survey.

- Other formats of rule representation (very short ones for big rule databases)

- Numeric input value for all input variables of an selected output variable that shall be evaluated. The input variables can displayed in a scrollable list, associated with a slider and a numeric input field.

- A text sensitive help to increase usability, in particular for new users.

- Edit functionality for existing variable terms. When defining a medium sized fuzzy system it can be very helpful to add, delete or rename terms in a testing phase of the system.
- Another possibility to spare time is a copy function for existing variables. With such a function all defined data of an already defined variable can be reused. In particular variables with a lot of terms and very special shaped membership functions could be reproduced easily. The parameter set cannot do the whole job, because very complex MSF cannot be defined, and so need a lot of time to be redefined over and over.
- More than one windows which display MSF/IMSF graphs. If membership functions of a group of variables are visualised, then e.g. evaluation changes for different input value can be easy to survey. This can be implemented with a list of MSF windows. Of course, window control (e.g., when a window has to be deleted) has to be done by the user.
- More functionality can be put directly to the MSF-window, e.g., menus or the selection of variables in a list, which is done now in the main window. But this depends on the functionality of canvas classes in new WxWindows versions or other toolkits.
- An enriched functionality to search the rule database and split it can be important to cope with huger databases. The premise can incorporate a number of clauses. A rule can be visualised only if
  - The premise contains a certain clause.
  - The premise contains a certain variable.
  - The whole rule contains a certain clause.
  - The whole rule contains a certain variable.

Nevertheless, the current implementation of InterFLIP++ provides ample functionality and ease of use for the careful user. It should therefore become well used part of the StarFLIP++ project.

Bibliography

[1] Philipp Ackermann. *Developing Object-Oriented Multimedia Software.* dpunkt, Heidelberg, 1996.

[2] Markus Bonner. Learning fuzzy rules with genetic algorithm and neural nets. Master's thesis, Technical Universitiy of Vienna, 1996.

[3] Markus Bonner, Stefan Mayer, Andreas Raggl, and Wolfgang Slany. FLIP++: A fuzzy logic inference processor library. In *Workshop Notes of the IJCAI'95 Workshop on Fuzzy Logic in Artificial Intelligence*, pages 83–92, Montreal, Canada, August 1995. URL: ftp://ftp.dbai.-tuwien.ac.at/pub/papers/slany/ijcai95wsflai1.ps.gz.

[4] Barry Boone. *Java Essentials for C and C++ Programmers.* Addison-Wesley, 1st edition, 1996.

[5] Dermot Brown. *Structured User Interface Design for Interaction Optimisation.* Prentice Hall, 1994.

[6] B. Demant. *Fuzzy Theory.* Vieweg, 1993.

[7] Didier Dubois and Henri Prade. *Fuzzy sets and systems, theory and applications.* Academic press, New York, 1980.

[8] Thomas Fehrle. Empirische Evaluation von Benutzungsschnittstellen. In Strothotte Böcker, Glatthaar, editor, *Mensch, Computer, Kommunikation.* Springer, 1993.

[9] Erich Gama. *Objectorientierte Softwareentwicklung an Beispiel von ET++.* Springer Verlag, Berlin, 1st edition, 1992.

[10] C. Görner. GUI-Qualitätssicherung. In K. Faehnrich, editor, *Werkzeuge zur Entwicklung graphischer Benutzerschnittstellen.* Oldenbourg, 1996.

[11] Dan Heller. *XView Programming Manual for XView V3.* O'Reilly and Associates, Inc., 1991.

[12] Erich Peter Klement, Radko Mesiar, and Endre Pap. On the relationship of associative compensatory operators to triangular norms and conorms. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 4(2):129–144, April 1996.

[13] Stefan Mayer and Wolfgang Slany. InterFLIP++, a GUI for FLIP++. In *Proceedings on International Panel Conference on Soft and Intelligent Computing*, pages 197–203, Budapest, 10 1996.

[14] Deborah J. Mayhew. *Software User Interface Design*. Prentice Hall, 1992.

[15] Witold Pedrycz. *Fuzzy Control and Fuzzy Systems*. John Wiley, New York, 2nd edition, 1993.

[16] B. Reusch. *Fuzzy Logik, Theorie und Praxis, 2. Dortmunder Fuzzy Tage*. Springer, 1992.

[17] B. Reusch. *Fuzzy Logik, Theorie und Praxis, 3. Dortmunder Fuzzy Tage*. Springer, 1993.

[18] B. Reusch. *Fuzzy Logik, Theorie und Praxis, 4. Dortmunder Fuzzy Tage*. Springer, 1994.

[19] M. Schneider-Hufschmidt. Eine Entwicklungsumgebung für adaptierbare Benutzungsoberflächen. In Strothotte Böcker, Glatthaar, editor, *Mensch, Computer, Kommunikation*. Springer, 1993.

[20] U. Schulte. *Einführung in die Fuzzy Logik*. Francis, 1993.

[21] Michael Schumacher. Gui builder fuer tk. *Xi, multiuser multitasking magazin*, 1(1):46–51, January 1997.

[22] Wolfgang Slany. *Fuzzy Scheduling*. PhD thesis, Technical Universitiy of Vienna, 1994.

[23] Wolfgang Slany. Scheduling as a fuzzy multiple criteria optimization problem. *Fuzzy Sets and Systems*, 78:197–222, March 1996. Issue 2. Special Issue on Fuzzy Multiple Criteria Decision Making; URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/fss96.ps.gz.

[24] Christian Stary. *Interaktive Systeme*. Vieweg, Braunschweig/Wiesbaden, 1st edition, 1994.

[25] T. Tilli. *Fuzzy Logik*. Francis, 3rd edition, 1993.

[26] Andre Weinand. *Objectorientierte Architektur fuer graphische Benutzeroberflaechen.* Springer Verlag, Berlin, 1st edition, 1992.

[27] Brent B. Welch. *Practical Programming in Tcl and Tk.* Prentice Hall, 1995.

[28] L. Zadeh and K. Tanaka. *Fuzzy sets and their applications.* Academic press, New York, 1975.