SESSION C7

Paper No. 1

OECD HALDEN REACTOR PROJECT

The Picasso-3 User Interface Management System

by

Øystein Jakobsen, Terje Johnsen, Hans Olav Randem

May 1994

Keywords: **User Interface Systems**, Computer Graphics, Development Tools & Environment, User Interface Design

The Picasso-3 User Interface Management System

Øystein Jakobsen, Terje Johnsen, Hans Olav Randem

Institutt for energiteknikk
OECD Halden Reactor Project
P.O. Box 173, N-1751 Halden, Norway

ABSTRACT

Picasso-3 is a User Interface Management System supporting object-oriented definition of graphical user interfaces (GUIs) in a distributed computing environment. GUIs are defined using an interactive editor, however GUI components can be created, modified, and deleted at run-time. Attributes of GUI components can be connected to application data in order to reflect some state of an application. The Picasso-3 system is designed to handle large amounts of application data in a real-time environment, and offers complete separation of user interface and application. The Picasso-3 system runs on standard UNIX workstations, and is based on industry standards such as C++, C, RPC, SQL, and X-Windows.

Presented at the OECD Halden Reactor Project
EHPG Meeting, Storefjell, Norway, 8th-11th March, 1993

1. MOTIVATION

User Interface Management Systems (UIMS) have been developed as part of the OECD Halden Reactor Project since 1984. The Main areas for these applications have been process control, simulation and emulation of existing control systems. Throughout the duration of development, several distinct generations of the system can be identified.

The first version of Picasso ran on proprietary platforms and the second version ran on standard UNIX workstations. This made Picasso-2 available to more users, and there are today more than 30 industrial applications using the Picasso system.

Experience from use of Picasso-2 in industrial applications raised new functional requirements. This indicated that further development of Picasso-2 would become increasingly difficult. Most of these requirements came from the more dynamic domain of network management systems. These applications often need to change the user interface "on-line" to reflect changes in the topology of a network or the contents of a database.

In a Picasso-2 application, when the database was to be reconfigured, the system had to be shut down. The new element could then be added to the database and the system restarted. Classes and variables were added in the the same manner. There was no way of defining attributes in pictures and classes or passing parameters to functions. Re-use of modules became a key issue, but this required an object-oriented design throughout the system. Another much desired feature was to utilize the benefits of window managers such as OSF/Motif's `mwm` or Open Windows' `olwm`.

Other motivations for Picasso-3 included the need for a better editor, easier maintenance and a better user interface for applications. These changes in the domain, together with the introduction of object oriented design and programming techniques resulted in Picasso-3, the third generation of Picasso. This is a complete redesign and reimplementation of the system, but built on past experience.

2. SYSTEM OVERVIEW

Although all features of a system like Picasso-3 can be viewed as equally important, the rest of this paper will nevertheless focus on just a few items; this is to give an impression of the power of Picasso-3. But first it is necessary to give a brief overview of the system; more extensive descriptions can be found in [8], [9], and [10].

2.1 Modules

User interfaces are often difficult to develop and maintain because even small adjustments to the user interface require major changes in application code. A UIMS should therefore offer *complete separation of user interface and application code*. Picasso extends this by offering complete network distribution of all modules.

Figure 1 shows the system architecture of Picasso-3. The major components are:

- *Graphics Editor*, GED — the tool used to design the user interface. User interface components (classes) can be defined, pictures drawn, and dialogues and dynamics can be specified.
- *User Interface Database* — the result of the work in GED is saved in this database, which then contains the complete specification of the user interface.
- *Run-Time Manager*, RTM — the part of Picasso-3 that actually realizes the application's user interface when the application is executed. RTM loads parts of the user interface from the UI Database as needed.

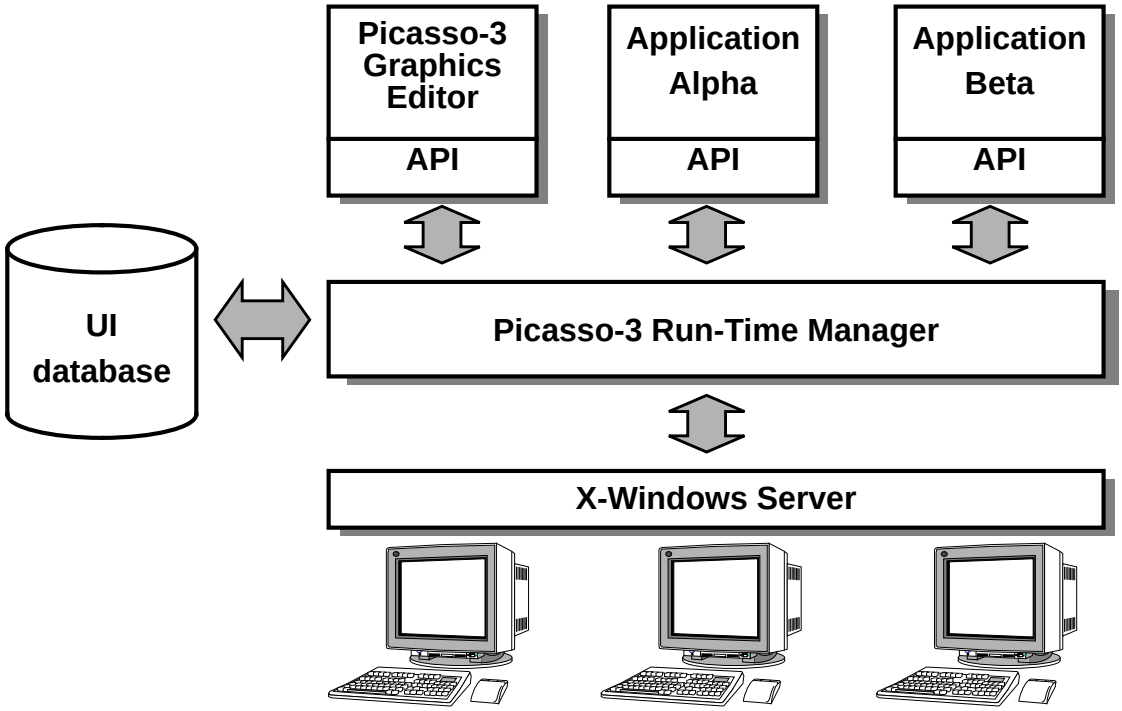

Figure 1 *System overview. The Run-Time Manager can serve several applications simultaneously. When an applications starts and connects to RTM, the user interface description is loaded from the UI Database and realized on the screen. The Graphics Editor is itself a Picasso application.*

- *Application Process* — the part of the application independent of the user interface; for example a simulator programmed in C. Several application processes can be used and distributed in the network to make up a complete application.
- *Application Progammer' Interface*, API — a library of C functions that is linked to the application process to enable it to communicate with RTM.
- *Communication System*, CS — not shown in Figure 1, but is utilized by both API and RTM as a means for communication. CS is a layer over the standard *Remote Procedure Call* facility, RPC.

When an application is started, the application process calls functions in the API library in order to connect to the Run-Time Manager. RTM responds by loading the application's user interface from the UI Database and displaying it on the screen. RTM will then continue to handle incoming events from X-Windows, generated by the end user, and from the application process(es), generated by the processes by calling API functions.

The user interface basically consists of *pictures* displayed in *windows*. Each picture contains *shapes* and *dialogues*. User interface components can be generalized in *classes* that can be instansiated in pictures.

2.2 Dynamics and the Function Language

When a user interface component depends on application data or another interface component, we have a *dynamic connection* between them. A dynamic connection is typically that the position, colour, visibility, or size of an interface component is a function of some data.

The dynamic connection is described by the user interface designer using the Picasso-3 function language. The language is a subset of C, with several object-oriented constructs from C++, and some other useful extentions are also included. The language is easy to use to set up simple dynamics, yet powerful enough to express complex logic and dynamics. The following example shows a dynamic connection to control the visibility of a user interface component:

```
visibility = 'valve34.isOpen() && container23.pressure < 100';
```

This expresses that a given interface component in a picture will only be visible if the valve object `valve34` is open, and the pressure in `container23` is lower than 100. `Valve34` is typically another interface component, `isOpen` is a function in that component, and `container23` could be a record (aggregate) variable in the application process with a field called `pressure`. Once this dynamic connection is set up, RTM will automatically update the picture so that the given interface component is visible or invisible, depending on the state of `valve34` and `container23`. Whenever the data changes, the user interface will change correspondingly to reflect the new data values.

2.3 Dialogues and the Function Language

As described in the previous section, dynamics are used to make the user interface dependent on application data. *Dialogues* are used to make the interface dependent on operator (end-user) actions.

A dialogue is a trigger/action pair; the *trigger* defines the conditions that should trigger the dialogue, and the *action* defines what should happen when the dialogue is triggered. A dialogue can be associated with user interface components in a picture, to the whole picture, or globally to the application as a whole. The following shows an example dialogue:

```
event  = LeftButtonPress;
action = statement '{ if ( valve23.isOpen() ) valve23.close(); }';
```

This defines a dialogue that is triggered whenever the operator presses the left button on the mouse when the cursor is within the dialogue sensitive area, i.e. within the interface component (for example a push-button). The action is expressed using the Picasso-3 function language. The example action specifies that if the interface component `valve23` is in the open state, it should be closed. Again, `isOpen` and `close` are functions (also expressed in the function language) defined in the valve component.

2.4 Changing the User Interface "On-line"

Most user interfaces are dynamic, also in the sense that the interface cannot be completely described *off-line*, i.e. before the application is run. One application can for example load some data from a database and then generate a graphical representation of these data; perhaps a network with nodes and edges. Although the interface components (classes) for 'node' and 'edge' can be described off-line, the network itself must be generated when the data is actually read from the database.

There is only a weak distinction between off-line and on-line in Picasso-3. Typical off-line activities performed by the user interface designer, such as defining interface components, drawing pictures, and setting up dynamic connections and dialogues, must also be carried out on-line when the application runs. Consequently the function language compiler must be available on-line as part of the Run-Time Manager, as shown in Figure 2

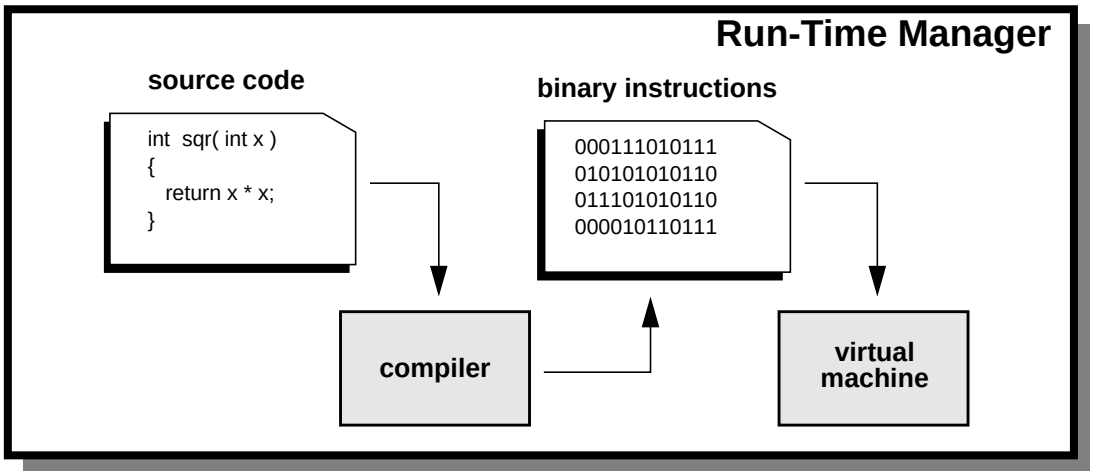

Figure 2 *Source code in the Picasso-3 function language is compiled and binary code is produced. This binary code is executed by the Virtual Machine, which is a software stack machine.Both the compiler and the Virtual Machine are parts of the Run-Time Manager. New source code for dialogue actions, dynamic connections, or functions can be created while the application is run-*

3. PICASSO-3 AS AN X CLIENT

Together UNIX and X-Windows form a multitasking environment where all running programs must be prepared to cooperate with other running tasks. X-Window itself does not force any specific behaviour as it provides *"technique, not policy"*, so rules for proper behaviour must be sought elsewhere. The *"Inter-Client Communication Conformance Manual"*, ICCCM [1], defines a set of rules that any well behaved X client is supposed to obey, some of which are described below.

3.1 Pasting Data

Copying data from one application to another under X-Window requires cooperation from both the giving and receiving programs. There are two strategies for copy/paste operations, *cut buffers* and *selections*.

Cut buffers allow programs to write text to one out of eight buffers held by the X server and to read the contents of any of these same buffers. Typically any text marked in some way by the operator is copied to the buffers, where some other application may fetch it when the operator selects a paste operation. Each application is free to utilize all eight of the buffers or just some of them. The most common strategy is to view the buffers as a ring queue. Cut buffers are restricted to text only. Picasso-3 uses cut buffers, so text can be copied between a Picasso-3 picture and any other program that supports cut buffers.

Selections is the preferred method of interchanging data because this concept is not restricted to text. Although more flexible, selections require extensive cooperation between the programs involved. If the end user selects a trend curve object in a Picasso picture and pastes that object into a statistics program; the following takes place if both RTM and the statistics package cooperates:

- When the end users does the paste operation in the statistics package, this program will ask the X server which application currently owns the selection and gets the answer: RTM.
- The statistics package will ask RTM, the selection owner, which formats the current selection can be delivered in. RTM will respond by listing the formats on which the trend curve component can be delivered. Alternative formats will typically be a list of time/value pairs, or a text string containing the same data.

- The statistics package will choose the most suitable format and request the data from RTM. If none of the formats are acceptable, no request is sent and the paste operation fails.
- RTM will convert the trend component to the requested format and send it to the statistics package.
- The statistics package includes the received data in its internal structure, perhaps display data on the screen showing peaks, averages, and deviations, and the paste operation is complete.

Cooperation is obviously crucial, and the more formats in which data can be both sent and received by the actors, the greater the possibility that the paste operation will succeed.

The operation of pasting data from one application to another is frequently used in graphical user interface environments like X-Windows, MS Windows, and on the Macintosh, so any program that does not support it will no doubt be considered *badly-behaved*.

3.2 Interacting with the Window Manager

The *window manager* is a special X client which is responsible for administrating all windows on the screen. Most window managers decorate the windows, i.e. draw frames around them which include means for iconizing, resizing, moving, or closing the windows. The title of each window is also part of the decoration. Although only one window manager can run on each screen, there are many different managers to choose from, and they can be quite different in behaviour and functionality. A well behaved X client like RTM must cooperate with the window manager in many tasks:

- RTM must inform the window manager what should be displayed as the title of each window, typically the name of the picture being displayed in the window.
- RTM must be prepared to receive notifications stating that the window manager has resized some of the windows. RTM can then take steps to make sure the picture fits the new window size, for example by scaling the picture.
- When the end user chooses a window manager function to close a window, the window manager will send a message to the window owner (RTM). To actually carry out the operation is the owner’s responsibility.

Window managers typically intercept the comunication between anX client and anX server, so anX client can make very few assumtions on its way. If for example RTM sends a request to the X server to create a window on the screen, this request will probably be intercepted by the window manager in order to be able to decorate the window. RTM cannot assume that the requested window actually appears on the screen immediately, or indeed at all; the window manager could choose to iconify it right away. RTM will have to wait for various confirmations before it can draw graphics in the window.

4. INTEGRATING OSF/MOTIF

Ignoring OSF/Motif when designing Picasso-3 was impossible as Motif is the preferred interface style within the UNIX world. Early tests however showed that integrating X Intrinsics and the Motif widget set directly into the Run-Time Manager was not a good solution. Motif is really a “closed” concept in the sense that it demands full control of its user interface components, which made it difficult to combine them with Picasso’s own graphical objects. Picasso-3 wasn’t meant to be a tool for developing push-buttons and menus either; there are literally dozens of good tools available for doing this. The solution was based on the following facts:

- Few applications actually mix widgets and graphics; most are organized as an “administrative” shell with menus, push-buttons and scrollbars, i.e. widgets, and all the graphics in a dedicated window (drawing area or canvas) somewhere in the centre.
- X-Windows is unique in the way it handles most of its resources. Windows are publicly available so several programs can simultaneously access the same windows on the screen. Although one

application A created and hence *owns* a window, another application B may well draw graphics in it and receive operator events from it.

4.1 The Concept

A concept was chosen that, in contrast to the alternative of full integration, gives some very important benefits:

- Almost any commercially available tool for building Motif applications can be used in conjunction with Picasso-3, for example TeleUSE or Interface Architect.
- The integration concept is not limited to OSF/Motif; it holds for other toolkits as well, like Sun's Open Windows, and even plain Xlib programs.

The integration concept is schematically shown in Figure 3. The Motif part of the interface is managed by a separate program which can be "hand coded" or generated by some Motif interface builder. The Picasso-3 API library is linked with this process to enable communication with the Run-Time Manager. The Motif process should contain no more than the code necessary to produce the interface, as the application code is separated in another process; like always in a Picasso-3 application.

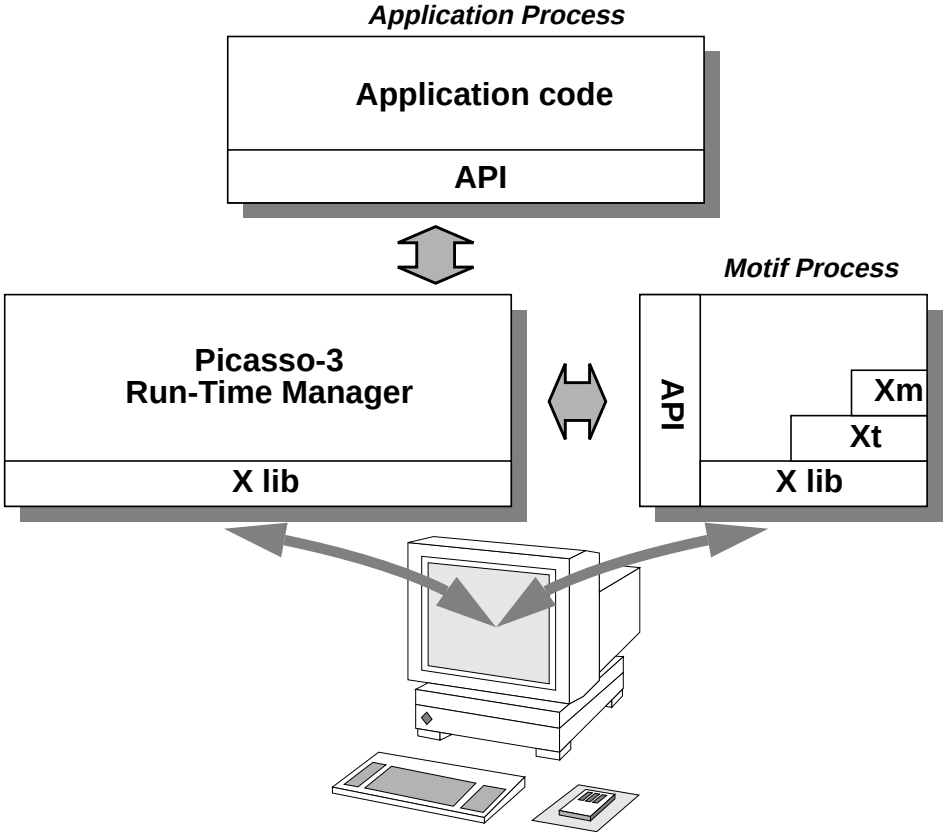

Figure 3 *The Motif part is separated from the Picasso part of the user interface. The application is still a separate process to ensure full separation of the user interface from the rest. In this figure the computer icon also represents the X server.*

The Motif process performs some handshaking upon start-up:

- It connects to the Run-Time Manager by a call to `API_Initialize`. RTM registers the Motif process as it does with any process that connects.

- It “exports” the canvas window(s) to RTM by calling `API_ExportWindow`. RTM will register the window much like its own windows, but the window is flagged as imported. RTM will subscribe to all the usual X events in order to obtain operator input in the window.
- It must instruct Xt to listen for input on the RPC socket used by API in order to handle incoming messages from RTM.

4.2 Handling Dialogues

As described in section 2.3, a dialogue is defined as a pair consisting of a trigger (event type) and action. Whenever an event of the specified type occurs, the action is triggered and performed. However two related challenges arise:

- An event occurring in a Motif widget, e.g. a push button, will probably lead to some action that should be done by the Run-Time Manager, e.g. to display another picture.
- An event occurring in a Picasso picture, e.g. addressing a graphics object, could very well lead to some action in the Motif process, e.g. popping up a Motif form to show some data regarding the selected object.

4.2.1 Remote Execution of Picasso functions

A program connected to the Run-Time Manager has the ability to send function source code to RTM for execution. The code is simply sent as a text string by applying a call to `API_Execute`. Applying `API_Execute( “{cut();}” );` will send the text string “{cut();}” to RTM. RTM will compile and execute the command, which is simply a call to the standard function cut() which deletes all currently selected objects from the picture. Figure 4 illustrates the process.

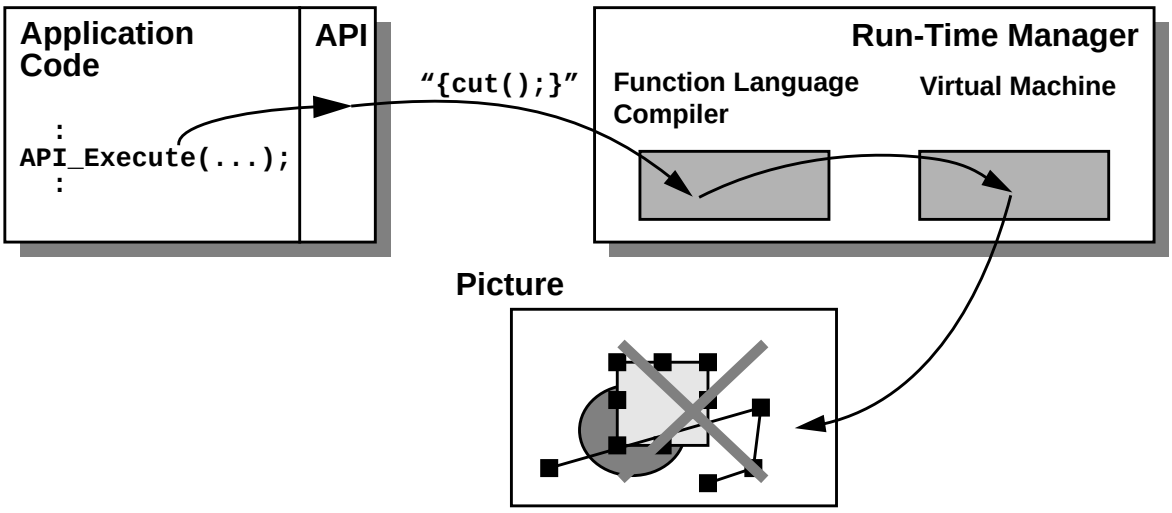

Figure 4 *Picasso-3 function language code can be executed remotely from an application by sending the source code as a text string to the Run-Time Manager for execution.In this case the result of the execution is that the currently selected objects (a rectangle and a line) in a picture are deleted.*

4.2.2 Remote Execution of Application functions

Ordinary C or C++ functions in an application can be made available to RTM by a call to `API_RegisterFunction`. So if the action part of a dialogue (triggered in RTM) is to be performed via the Motif process, then the dialogue should simply be defined in Picasso as a remote call to a function in the Motif process. The following C++ code segment shows an example:

```
REGARGTYPE args[2] = { {Int,1}, {UnsignedChar,32} };
API_RegisterFunction( “f”, &aCfunction, 2, args );
```

Here the C++ function `aCfunction()` is made available for RTM under the name “f”; the functions should take two parameters, a single integer and an array of 32 characters. Then the function can be called from a Picasso function language statement:

```
MainSim.f( 123, aStrVar );
```

Here the remote function f in the application process named `MainSim` is called with parameters 123 and the value of a string variable. Figure 5 illustrates the process.

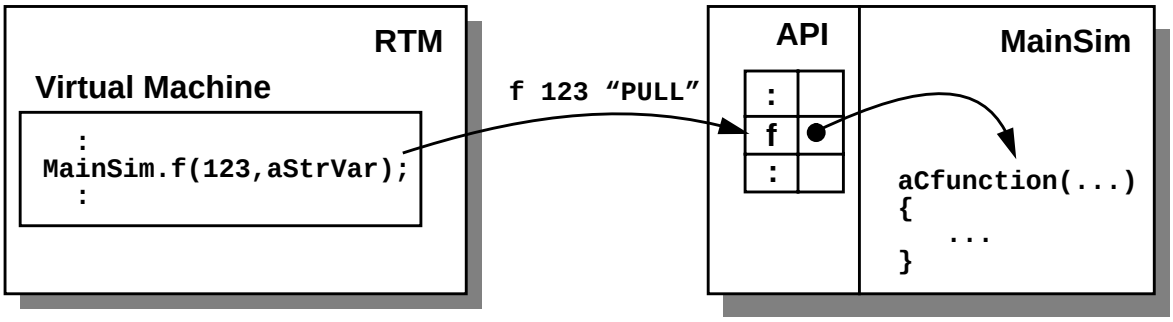

Figure 5 *An application function is called remotely from the Run-Time Manager. The function identifier “f” and the value of the parameters, 123, and “PULL”, are sent to API. API holds a table of all registered functions and calls the functions associated with the identifier “f”.*

4.3 The Picasso-3 Graphics Editor

As usual, the proof of the pudding is in the eating, and the Graphics Editor delivered as part of Picasso-3 hence utilizes the Motif integration concept. Figure 6 shows GED as it appears when the user interface designer edits a process mimic picture.

The graphics editor is a tool covering all aspects of the interface design process, some functions available to the interface designer are listed below:

- Set up basic properties like text fonts, line styles, fill patterns, and colours.
- Define reusable user interface components, known as classes, and connect to already existing libraries of components.
- Draw pictures using available graphic primitives and higher level interface components.
- Define relationships between pictures in order to set up hierarchies, chains or graphs of pictures, classes.
- Set up dialogues to define how the interface should respond to user interaction.
- Connect attributes to process data to define how the user interface is a function of the state of application processes.
- Test the interface to verify that dynamic behaviour and dialogues work as intended. Testing the interface without actually starting the application is an important feature of the editor.

The Graphics Editor consist of a canvas window, and Motif widgets like menus and scrollbars. The canvas window is exported to RTM as described in section 4.1. The Motif menus and Motif dialog boxes are used to obtain input from the interface designer concerning the editing options. When the interface designer selects editing options from the Motif menus, information is sent to RTM which carries out the instructions as described in section 4.2.1 and Figure 4. As an example, when cut is selected from the Edit menu, a call to API_Execute(“{cut();}”) is applied by the Graphics Editor.

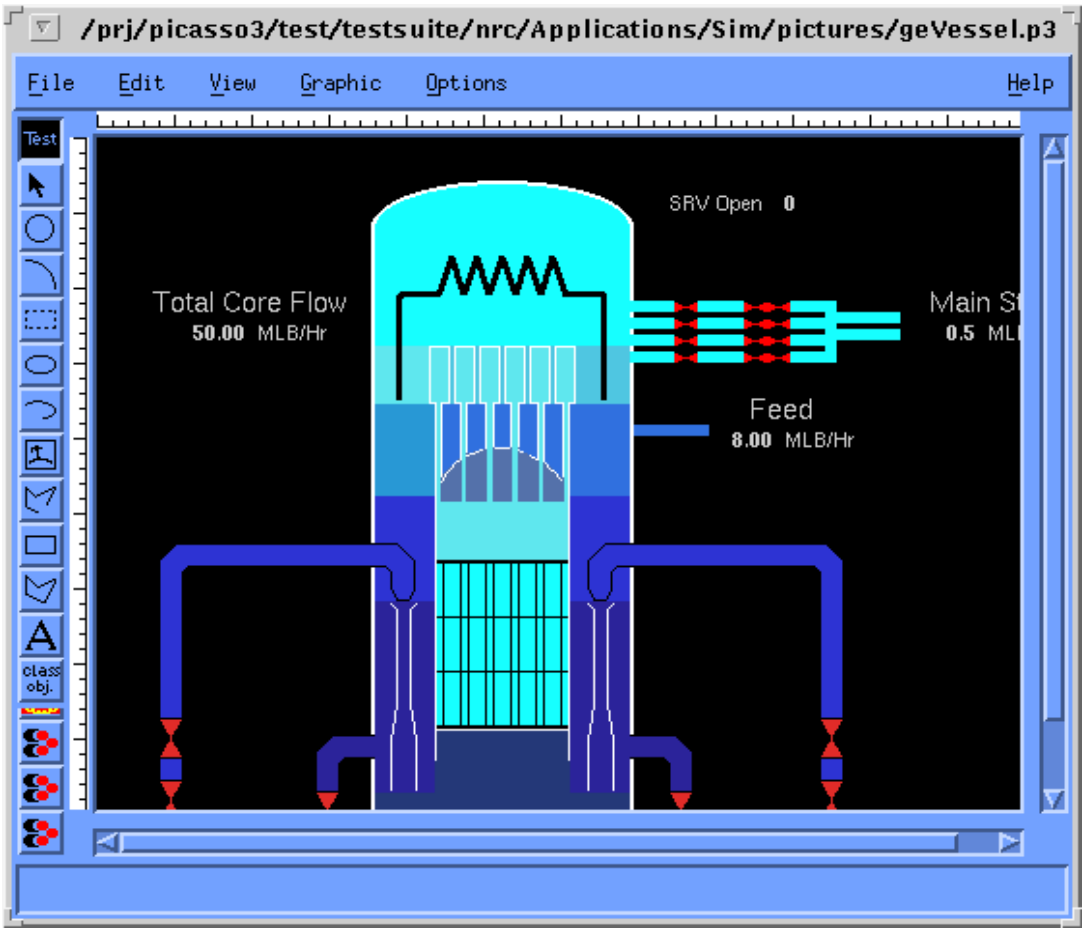

Figure 6 *The Picasso-3 Graphics Editor. The editor process is responsible for drawing all OSF/Motif widgets, i.e. the menus, push-buttons, and scrollbars, while the Run-Time Manager displays the canvas window where the reactor vessel picture being edited is shown.*

GED has the options that you expect to find in a graphics editor. The editing is done in WYSIWYG style (that is when a change is done in the picture the interface designer can immediately see how the change has affected it). Important here is that the interface designer can immediately test the interface, without leaving the editor. During the testing and editing the designer has full access to the Picasso-3 function language. This enables the designer to do much of the testing without connecting to the application that the interface is developed for.

5. THE DEVELOPMENT PROCESS

Picasso-3 is written in C++, the object-oriented extention of the standard C language [2]. The techniques of object-orientation were used both in the design and implementation phases. Basically, object orientation offers:

- *Classes* — The problem domain is modelled as a set of *objects*. Objects of the same “kind” are generalized as *classes*. An object has a state, modelled as data, and a behaviour, modelled as functions.
- *Encapsulation* — Separating clearly *what* a certain code segment does from *how* it actually does it, results in code that is easier to reuse, maintain, and extend. A class should always have a public interface part (*what*) and a private implementation part (*how*).
- *Inheritance* — While most languages offer mechanisms for expressing *is-part-of* relations, few languages offer effective means for expressing *is-kind-of* relations.

C++ supports the concept of classes, objects, and inheritance. Since the problem domain can also be modelled in this way, then the *semantic gap* between the *design model* and the *implementation model* is reduced. Being able to work out an intuitive design model and to transfer that model directly into computer source code is clearly the main advantage of object-oriented languages and indeed C++.

Portability was a key issue in the development process. Some claim that “there is no such thing as a portable program, only programs that are actually ported”. However, a system can be more or less portable, and great efforts have been made to make Picasso-3 as portable as possible. Porting software is best done as a continuous process in parallel with the development, so the staff used four different platforms[1] when developing Picasso-3. Developing software on different platforms also has another effect aside from portability: Each computer typically reacts to bugs in different ways. While one system is robust and “obscures” a certain bug, another system may break immediately, so the bug is caught. Another bug may cause a behaviour much the other way around, so testing software on several platforms clearly has advantages.

The Picasso-3 development process also resulted in code that can be directly reused in other projects. A library of C++ classes to handle dats structures were developed, and contains classes for most common data structure building blocks like stacks, queues, dictionaries, hash tables, etc. The communication system CS is also reusable and can be used independently of Picasso-3.

6. MIGRATING FROM PICASSO-2

Backward compatibility is a key issue in any software upgrade. Since Picasso-3 is a full redesign and reimplementation of the Picasso system, compatibility was a major concern. The problem can be divided into two tasks:

- *Configuration compatibility*, i.e. old pictures, object libraries, can be reused.
- *Communication compatibility*, i.e. old application processes can connect to the Run-Time Manager and run without modification.

Picasso-3 is not binary compatible with Picasso-2, so old pictures etc. cannot be directly displayed in Picasso-3. An old configuration can however be *converted* to Picasso-3 format using a conversion tool delivered as part of Picasso-3. This tool converts:

- Picasso-2 pictures and object libraries.
- Picasso-2 evaluation functions and dialogue functions. These are source code converted to the Picasso-3 function language.
- Colours, fonts, linestyles, patterns and window as given in the picasso2.cnf file are converted.

Picasso-3 uses a different message protocol than Picasso-2, so existing Comix[2]-based application processes cannot connect directly to the Run-Time Manager. Two possibilities exist:

- The application process can be rewritten to fit the Picasso-3 API routines. The major differences are in the initial messages passed upon start-up. A structured application process can be rewritten fairly easily.
- A filter process can be used between the application process and the Run-Time Manager. This filter will convert Comix messages from the application process to API calls to RTM, and vice versa. A filter process is planned to be developed and delivered as part of Picasso-3.

1. Solaris on SPARC, SCO ODT on Intel 80x86, HP-UX on Motorola 68K and PA-RISC.
2. Comix is the Berkeley socket-based communication system used in Picasso-2. Picasso-3 uses CS which is based on Sun’s Remote Procedure Call protocol.

7. CONCLUSION

Picasso-3 offers network distribution as the application itself can be spread out in a network while the user interface can be managed by the Run-Time Manager on a different workstation. This strategy also comforms with the idea of separating the user interface from the application code.

Even though the work done on Picasso-3 started out from scratch, the work gained much from what was done in earlier versions. This experience together with more and improved possibilities for managing dynamic domains will allow Picasso-3 to be used in a wide range of applications. Picasso-3 was designed and implemented using object-oriented techniques, and will in itself offer an object-oriented enviromnent for creating intuitive and reusable user interfaces. More advanced user interfaces can be defined because interface components can be created, modified, and deleted on-line by the Run-Time Manager.

Picasso-3 is open in the sense that it can run in close connection with other graphical applications, as the OSF/Motif based editor GED examplifies. Great effort has been put into making Picasso-3 a well-behaved X client when it comes to pasting data and interacting with the window manager.

8. FURTHER WORK

As Picasso-3 will operate in real-time environments, strict requirements exist regarding stability, speed, and memory usage. Substantial effort must be put into testing, debugging, and optimizing the Run-Time Manager.

Functionality to allow access to relational databases through SQL must be added, and the now rudimentary trend log module must be developed further. Full transformations, i.e. scaling and rotation, will be included for all user interface objects.

The object-oriented features of Picasso-3 will be extended further by allowing user interface components to inherit from others. Then general and highly reusable interface components can be sub-classed to make new and more specific components.

In the future the field of multimedia will be explored to be able to integrate video and sound as part of user interfaces generated by Picasso-3.

9. REFERENCES

[1] D.S.H.Rosenthal: *Inter-Client Communication Conformance Manual,* MIT X Consortium, 1989.

[2] M.Ellis, B.Stroustrup: *The Annotated C++ Reference Manual*. Addison-Wesley, 1990.

[3] A.V.Aho, R.Sethi, J.D.Ullman: *Compilers, Principles, Techniques, and Tools,* Addison-Wesley, 1986.

[4] D.E.Knuth: *The Art of Computer Programming, Volume 3,* Addison-Wesley, 1973.

[5] B.Meyer: *Object-Oriented Software Construction*. Prentice Hall, 1988

[6] P.Wegner: *Concepts and Paradigms of Object-Oriented Programming*. OOPS Messenger, ACM Press, Vol.1 No.1 August 1990.

[7] R.Wirfs-Brock et al: *Designing Object-Oriented Software*. Prentice Hall, 1990.

[8] K.A.Barmsnes, A.Hornæs, Ø.Jakobsen, R.Storkås: *Picasso-3 Functional Specification*. Work Report HWR-278, 1991.

[9] K.A.Barmsnes, A.Hornæs, Ø.Jakobsen, R.Storkås: *Picasso-3 System Design*. Work Report HWR-288, 1991.

[10] K.A.Barmsnes, A.Hornæs, Ø.Jakobsen, R.Storkås: *PICASSO: A User Interface Management System for Real-Time Applications*. BCS workshop on User Interfaces for Expert Systems, London, 1991.