

DEVELOPING GRAPHICS APPLICATIONS IN AN INTERACTIVE ENVIRONMENT

Kjell Arne Barmsnes, Øystein Jakobsen, Terje Johnsen, Hans Olav Randem
Control Room Systems Development Division
OECD Halden Reactor Project
N-1750 HALDEN, NORWAY

ABSTRACT

Picasso-3 is a User Interface Management System (UIMS) supporting object oriented definition of Graphical User Interfaces (GUIs) in a distributed computing environment. In Picasso-3, GUIs are defined using an interactive editor; however GUI components can be created, modified, and deleted at run-time. The graphics editor is designed in a way that allows immediate testing of the interface to verify that dynamic behaviour works as intended. Attributes of GUI components can be dynamically connected to application data in order to reflect some state of an application. The dynamic connection and the behaviour of the interface is described by the user interface designer using the Picasso-3 language, pTALK.

Reusable interface components, or objects, can be created interactively using the graphics editor. Objects can be put in libraries for easy retrieval and reuse.

The Picasso-3 system is designed to operate in a network environment; Picasso-3 can run on one workstation while the application programs can be distributed on several workstations. The Picasso-3 system is based on industry standards.

Application areas where Picasso-3 is being used include development of simulation tools and simulator interfaces, computerised operator support systems, network and accident management systems, and expert system interfaces.

This paper focuses on how the Picasso-3 system can be used by the user interface designer to interactively create graphical user interfaces.

INTRODUCTION

The OECD Halden Reactor Project which carries out an international research and development programme in the man-machine systems area, sponsored by organisations in 14 countries, has for more than 20 years been engaged in development of user interface systems.

Graphical presentation in computer programs (applications) has become more and more sophisticated and

advanced throughout the last years. The Picasso-3 system is a flexible and powerful tool offering system developers an aid in developing and testing of graphical interfaces. Furthermore it is an execution environment for applications. It is the third generation of *User Interface Management Systems* (UIMS) developed as part of the OECD Halden Reactor Project since 1984.

Two main goals in developing the system have been to:

- enhance the quality of graphical user interfaces of applications
- reduce the development and maintenance costs throughout an application's lifetime.

To improve the quality of graphical interfaces, the Picasso-3 system offers an interactive design/test environment that makes development and testing more efficient. The *Graphics Editor* (GED) is used in this process. GED has two modes of operation; edit-mode is for drawing operations and creation of dynamics and dialogues, and the test mode is for testing dynamics on-line. Switching between the two modes is done by selecting from a menu in GED.

In this way, the switching between design and testing of interfaces can be fast and easily done. This makes Picasso-3 well suitable as a prototyping tool and leads to an evolutionary development process. Even though savings in this phase of a software's lifecycle can be considerable, the real potential for savings is found in the maintenance phase, when the software is installed and is used for its aimed purpose. Errors are found and corrected, but the main work for the developers is to improve and enhance the functionality.

Most of the changes in functionality involve presenting information that is already stored in the system in alternative ways. A way of reducing maintenance costs is to separate the user interface from the application code and information storage. This way, a change in the interface would result in minimal changes in an application's code. A major requirement to the Picasso-3 system is to offer a complete separation of the user interface from the application code.

The Picasso-3 system runs under X-Windows on standard UNIX workstations and is developed using the C++ pro-

programming language and object oriented programming techniques.

Picasso-3 operates in a distributed computing environment. This means that the system must be able to cooperate with other running tasks and support a client-server architecture. The central part of the system is the *Run Time Manager* (RTM), realising the user interface. Communication is carried out through a set of high-level interface routines that is linked with every program. These *Application Programmer's Interface* (API) routines give application programs access to all functionality in the RTM.

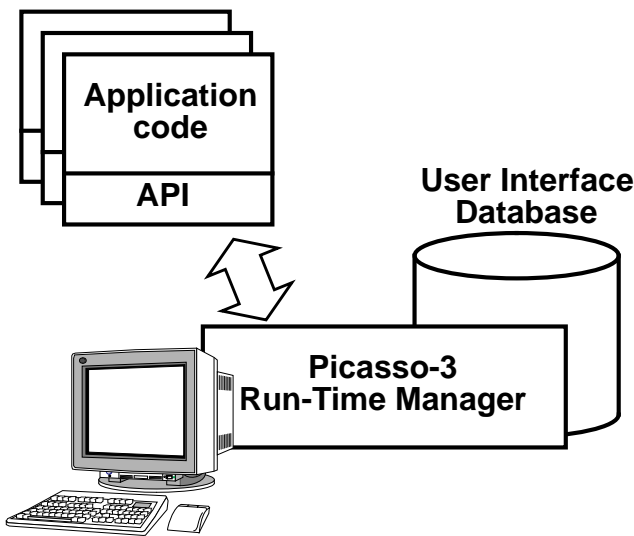


Figure 1. The end-user interacts with the user interface described in the interface database and realised by the RTM. Application programs, communicating through the API library, can be distributed in the network.

Building graphical user interfaces is a time consuming process. A main issue in the Picasso-3 system is to develop high-level user interface components that can be reused. These components are building blocks that represent real world objects such as valves and pumps of different flavours. In Picasso-3 terms we call the representation of these real world objects *classes*. When such a real world object is used in a picture, we say that the class is *instantiated* in the picture. Instantiation of classes is as easy in Picasso-3 as drawing of primitive shapes such as rectangles and circles.

The Graphics Editor is used to create the *look* (appearance) and *feel* (behaviour) of graphical objects. These can be primitive shapes or instances of classes.

INTERACTIVE USER INTERFACE DESIGN

The design and implementation of graphical user interfaces is typically an iterative process. The design tool should therefore provide the designer with methods that support an

integrated design/test environment. Furthermore, a user interface design tool must provide the designer with a high-level way of creating quality graphics.

In Picasso-3 the graphics are defined using an interactive graphics editor. The Picasso-3 Graphics Editor offers general editing functionality similar to that of a general purpose graphics editor, as well as more specialised functionality for creating dynamic graphics.

User interface development is traditionally carried out in two steps. First, to use some off-line tools to specify the interface and somehow compile this specification, and then to run an on-line system that uses the compiled user interface specification. In Picasso-3 there is no distinction between off-line and on-line. All Picasso-3 tools operate on-line and therefore no off-line recompilation and restart of the system is necessary. The edit functionality is an integrated part of the Picasso-3 system, and hence editing operations are available at any time. This is utilised in the graphics editor by allowing the user interface designer to easily switch between design and test mode. In test mode the system is run in its real run-time environment and all parts of the system can be tested. If the actual application is available, it can be connected to the Picasso-3 system while the user interface is being designed, to allow the interface to be tested with real application data.

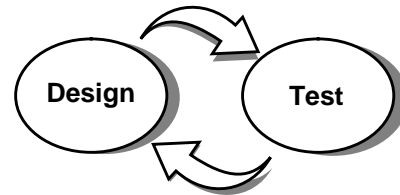


Figure 2. Picasso-3 offers a development environment where the user interface designer easily can switch between design and test mode, without recompiling and restarting the system.

A user interface consists of more than the static graphics. In order to be able to reflect changes in an application program, it must be possible to connect the interface dynamically to application program data. In Picasso-3 the end user can interact with the system through dialogues, describing what actions to be executed as a response to the users input.

Dynamic Graphics

A key functionality in the graphics editor is the creation of dynamic graphics. That is to connect graphics to application data. *Dynamics* are created using Picasso-3's user interface language, pTALK. pTALK is a C/C++ like language with additional constructs for graphics manipulation. The language offers a powerful and flexible way of creating dynamics. The language is compiled at run-time, and hence new pTALK code can be added and tested/executed immediately during the design as well as during the execution of the interface. A typical example of dynamic graphics could

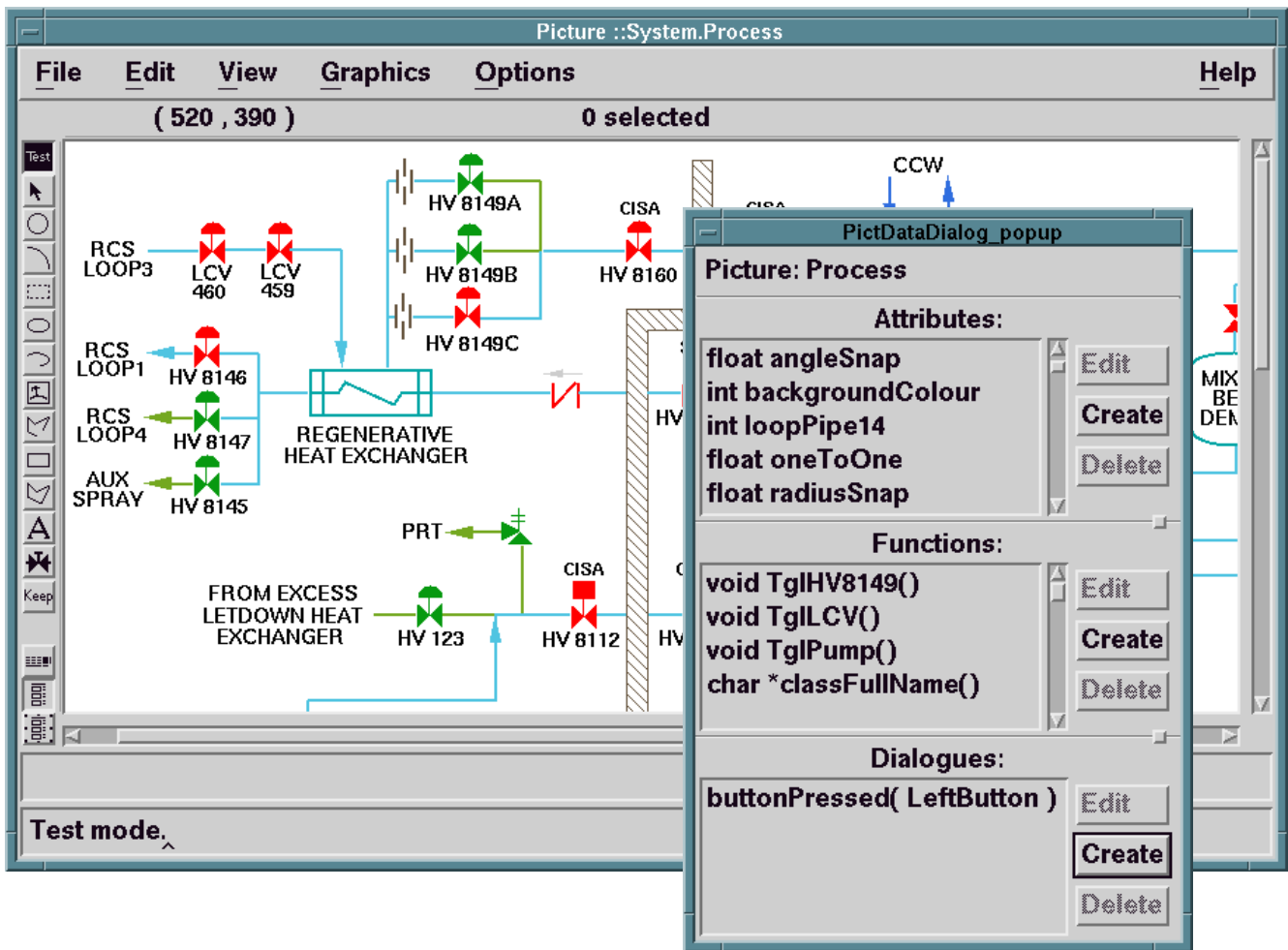


Figure 3. The user interface designer works in the interactive environment constituted by the Picasso-3 Graphics Editor (GED). Attributes, functions and dialogues in the picture, or in selected objects in the picture, can be viewed or changed. Switching between design mode and test mode is done simply by pressing a push button. In test mode, the interface looks and feels exactly as it will do on the end user's screen. The process picture shown in the graphics editor is part of the US NRC's NEWS project.

be to let the foreground colour of a rectangle be dependent on an application variable. In this example a function that returned the appropriate colour would be created, as shown below:

```
int statusCol( float Status )
{
    if ( Status > 0 )
        return red;
    else
        return green;
}
```

The rectangles foreground colour could then be connected to an application variable 'TankStatus' through this function:

```
foregroundColour = 'statusCol(TankStatus)';
```

Now, the foreground colour of the rectangle is dynamically connected to the application variable 'TankStatus', and the foreground colour will automatically be updated whenever the value of the variable 'TankStatus' is updated from the application program. In this manner all graphics attributes can be made dynamic. Example of attributes are x and y position, **width** and **height**, colour attributes, **visibility**, and for classes, *user defined attributes*.

Dialogues

An important feature of the Picasso-3 system is the ability to create powerful and flexible *dialogues*. In Picasso-3 a dialogue is defined as the interactive part of a graphic element, and is integrated with the graphics element itself. Dialogues can be defined on different levels in the interface. User interfaces developed in Picasso-3 consists of three levels: from top to bottom, the application level, the picture level, and the element level. This hierarchical structure allows the

user interface designer to create local as well as more global dialogues.

An example of a dialogue on a valve object would be to toggle the position of the valve whenever the user clicks on the valve, using the left mouse button. A dialogue consists of two parts: an *event description* and an *action* to be executed whenever the corresponding event occurs. Both the event and the action is described in the pTALK language. The language contains event specific functions, examples of which are:

```
buttonReleased( AnyButton )
buttonDoubleClicked( LeftButton )
keyPressed( CarriageReturn )
cursorMoved()
shapeEntered()
```

An event description can be an arbitrary pTALK expression, containing at least one event specific function:

```
InFreeze() && buttonClicked(LeftButton)
```

The action part of a dialogue is a pTALK statement, and thus can contain calls to standard functions and user defined functions. Since both the event and action descriptions are user defined, highly specialised dialogues can easily be developed utilising the pTALK language.

As edit functionality is an integrated part of the Picasso-3 system, graphical objects can be manipulated at run-time. This can be used in applications where a dynamic data structure is represented graphically, and hence the structure of the graphics should change as the data structure changes. A typical application would be a network application where the number of nodes and the connectivity between nodes change over time. Through the use of standard functions in pTALK, new graphic network nodes and connection lines can easily be created to reflect the changes in the network application. Another application area where run-time manipulation of graphics is needed, is in editor-like applications. The Picasso-3 graphics editor is itself an example of an application utilising the run-time editing functionality provided by the Picasso-3 Run-Time Manager.

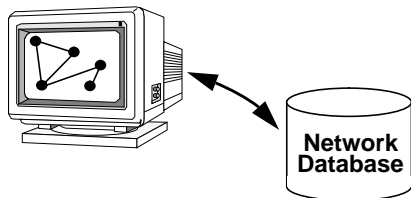


Figure 4. Graphics can be created at run-time to reflect a dynamic data structure. A dynamic network described in a database can be presented graphically in Picasso-3 by creating new graphical elements at run-time, as the network changes.

REUSE OF INTERFACE COMPONENTS

User interfaces are often built to depict real world components. The Picasso-3 *class* concept allows the user interface designer to model these components in the interface. The components (valves, pumps, multiplexers, etc.) are often drawn as a combination of simple shapes. A valve can be represented by a drawing composed of a polygon, a line, and an ellipse.

The design of user interfaces can be optimised by using high-level components instead of drawing each component from scratch each time. The Picasso-3 system gives the designer the ability to design such high-level components, called classes. The actual use of a class in a user interface picture is called an *instance* of the class.

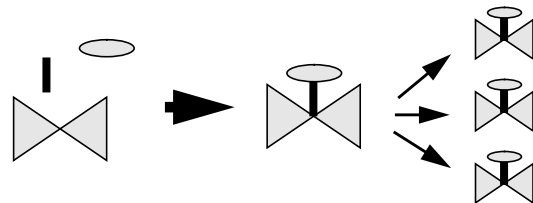


Figure 5. Simple shapes are combined into high-level components, classes, that can be instantiated into the user interface.

A real world component often has some state information and some behaviour related to it. A valve has a state in that it is either open or closed. It also has a behaviour in that it can be opened or closed. In the Picasso-3 system the state information for a class is stored as *attributes* of the class. The component's behaviour is also part of the class, and is realised as pTALK functions and dialogues. The fact that not only the appearance of the real world components, but also the state and behaviour associated with the components are part of the classes, is one of the object oriented features of the Picasso-3 system.

A Picasso-3 class is composed of simple shapes as rectangles, circles and lines, and of attributes, functions, and dialogues. A class can also contain instances of other classes.

A class can be designed independently of its surroundings, making it possible to reuse the components in different user interfaces, and different parts of the user interface. A class can be viewed as a small user interface in itself, and this is taken advantage of in the Picasso-3 system; classes are constructed the same way as ordinary user interface pictures, i.e. using the interactive environment provided by the Picasso-3 Graphics Editor (GED).

Classes can be stored in *libraries* to make reuse easier. Preferably, classes describing related components should be stored in the same library. To make the libraries independent

of context, other interface elements can be stored in the library, such as colours, fonts, and functions.

Another use of libraries, is to easily exchange the look of interface components in an application, simply by switching between libraries. This can be useful if the application should conform to different standards for the look and feel of the components.

An example could be a picture containing instances of two classes Valve and Pump. These classes are stored in a library named 'ISO_lib'. Equivalent components are also stored in the library 'ANSI_lib'. Switching between the two standards can then be done simply by switching libraries.

INTEGRATION

Picasso-3 supports several mechanisms to coexist with other software components in order to integrate into a complete system. Integration mechanisms fall into two categories, namely *integrating by communication*, and *integrating user interfaces*. The first category deals with message passing in a computer network, while the second deals with cooperation in a graphics environment.

Communication

Since Picasso-3 encourages distributed systems, *communication* is a keyword in the integration process. Picasso-3 allows other programs to communicate with the user interface through the *Application Programmer's Interface*. API is a routine library which puts the functionality of Picasso-3 at reach of any C callable program.

Data sharing is important in a distributed system. Data has a source, which is the program that owns the data and controls it. However, the data is typically needed in other programs. The data generated in an acquisition system must for example be available to a separate program for analysis, and to the user interface system for presentation.

A program containing variables, can make these variables available to the user interface by calling selected API routines. Variables can be simple floating point or integer numbers, or complex record structures. When variables are declared to Picasso-3, they can be referenced in any pTALK expression as part of the user interface logic.

The Picasso-3 API hides the error-prone process of packaging and interpreting bytes, something that too often becomes the main activity when programming a message based system. When a variable is declared to Picasso-3 using the appropriate API calls, the API routines will take care of sending and interpreting the necessary messages. Whenever a variable is changed by a pTALK expression, the new value must be sent from the Run-Time Manager to the application program. When a variable is changed by the program itself, the new value must be sent to the Run-Time Manager.

In addition to data sharing, there is often a need to issue commands to other programs. The goal is to make the programs look and feel as one integrated system, although they are separate programs running on different computers. Calling *remote procedures*, i.e. routines in other programs, is one way of achieving this, and Picasso-3 supports this in two directions.

An application program can call a pTALK function, and actually execute any legal pTALK expression or statement. Since pTALK is an interpretative language*, the source code can be built at run-time, and even be entered by the end-user.

Routines in the application program can be called from within a pTALK expression. This requires that the application program declares the routines to Picasso-3 prior to the call, much the same way as with variables. This means that the user interface can easily issue commands to the underlying program, for example tell a simulator to go freeze, or tell a database system to save its data to file:

```
bwrSimulator.freeze( )
dataServer.save( "newdata.dat" )
```

An object-oriented notation is used in the calls, as we for example tell the bwrSimulator (the object) to freeze (the message).

Calling API routines is simple as long as the application program's source code is available. When we want to integrate with a commercial product, we probably need a different strategy. Unfortunately there is no accepted standard for high-level inter-process communication within the UNIX world, and a program cannot possibly support the protocols of all other available systems. The solution is to put a small filter program between the two systems, a program that will translate between the two protocols. The filter program will typically include the two system's different API libraries. This strategy can be used to integrate Picasso-3 with commercial products, for example database systems, spreadsheets, or *Geographical Information Systems* (GIS).

Picasso-3 communicates using Remote Procedure Call (RPC), and supports both synchronous and asynchronous message passing.

Integrating User Interfaces

Some parts of a complete user interface set special requirements that cannot easily be fulfilled by a general purpose interface tool. Needs for easy development and high

*. Although pTALK is interpretative, the source code is still compiled transparently within the Run-Time Manager to increase execution speed.

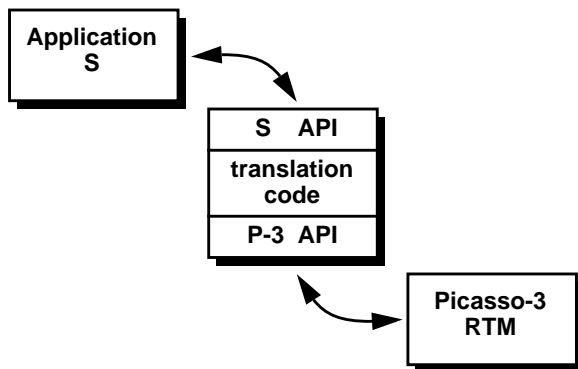


Figure 6. Communication between two applications with their own protocols. A filter process is used to translate between the protocols. In this example, Picasso-3 is connected to another system S.

performance imply that for example the following interface parts should be realised using more dedicated tools:

- Digital pictures and maps
- Complex graphs, charts, and plots
- Application frameworks (menus, push buttons, lists etc.)

Since a complete user interface can typically contain both maps and graphs as well as more traditional elements, the key to success lies in making the different tools work together. Each tool will be responsible for its own part of the user interface, but to the end user it will look as one uniform system. To realise this, the different tools must conform to the following requirements:

- All tools must be able to communicate with each other, i.e. send commands and messages.
- Tools must be able to “mix” graphics, which means that a window (target) on the screen could possibly contain graphics presented by more than one tool (source). Picasso-3 is for example able to display graphics in windows provided by an OSF/Motif application.

In this way each different part of the user interface can be built with the tool best suited for that part, and each part integrates with the complete system.

CONCLUSION

Picasso-3 assists system developers in designing and testing graphical user interfaces. The main objectives of the system are to enhance the quality of the interface and to reduce the development and maintenance costs.

Graphical interfaces are defined in an interactive graphics editor, allowing the designer to easily switch between design and test. Graphics are dynamically connected to application program data through the pTALK language.

Interaction with the end-user is specified in dialogues configurable by the interface designer using pTALK.

High-level, reusable interface components, or classes, can be created interactively in the graphics editor. The classes together with their resources can be stored in libraries for later reuse.

Communication and integration are not the sole responsibilities of a general purpose interface tool. No single tool exists that can take care of all typical user interfaces. Computing today takes place in more and more distributed hardware systems. Tomorrow’s applications will not be huge computer programs; they will consist of smaller building blocks running together and cooperating in order to solve the end user’s needs. Picasso-3 is one such building block, and there will also be GIS building blocks and others.

Picasso-3 is being used in several application areas. Examples of these are the development of simulation tools, simulator interfaces, telecommunication network management systems, accident management systems, expert system interfaces, and computerised operator support systems.

One current use of the Picasso-3 system is in the US NRC’s Nuclear Engineering Workstation Simulator (NEWS) project.

BIBLIOGRAPHY

Barmsnes, K.A.; A. Hornæs; Ø. Jakobsen; R. Storkås. 1992. “PICASSO: A User Interface Management System for Real-Time Applications” In *Collected Papers from the 2nd SGES International Workshop* (London, Mar. 11-12). British Computer Society.

Ellis, M.A; B. Stroustrup. 1990. *The Annotated C++ Reference Manual*. Addison-Wesley.

Meyer, B. 1988. *Object Oriented Software Construction*. Prentice Hall.