Title
Rapid Prototyping Of Advanced Driver Interface Systems

Authors
Massa, Laura J.
Mendel, Max B.

Publication Date
1993

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Rapid Prototyping of Advanced Driver Interface Systems

Lauren J. Massa
Max B. Mendel

UCB-ITS-PRR-93-8

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

AUGUST 1993

ISSN 1055-1425

Rapid Prototyping of Advanced Driver Interface Systems

Lauren J. Massa and Max B. Mendel

April 30, 1993

1 Introduction

This report describes a computer environment for rapidly prototyping user interfaces for advanced driver information systems or ADIS. The work was carried out in the Systems Integration Laboratory at the Department of Industrial Engineering, UC Berkeley. In addition to the authors, Dr. James Roseborough has contributed significantly to the project.

1.1 Background

Using an environment for rapid prototyping of user interfaces, radically different design ideas can be implemented in prototypical form almost instantaneously at no additional costs, resulting in better designs in shorter time.

Currently, prototyping a user interface is a time consuming and hardware intensive process; it typically takes tens of years for an effective user interface to emerge. For instance, the driver interface in a car has been evolving over 70 years from where the car could only be driven by a trained chauffeur to where it can be driven by almost any adult. A VCR, on the other hand, remains difficult to program, despite being in the market for about ten years. The ability to rapidly prototype high-fidelity user interfaces would provide a means to converge to an effective interface during initial design phases, rather than at incremental steps taken when a new model is brought on the market.

Systems which create user interfaces for computer programs are already being developed. For instance, Next's InterfaceBuilder allows a program developer to rapidly create an interface consisting of buttons, menu items, etc. Development environments also exist for the Motif and OpenLook toolkits. Such systems are called "User Interface Management Systems" or UIMS. Similar advanced prototyping technology is applied to Computer Aided Software Engineering (CASE) and semiconductor CAD/CAE. Early systems provided a more inexpensive way to produce designs, but were focused primarily on producing design artifacts and providing facilities for automated design rules checking. These systems have evolved into sophisticated tools; they incorporate moderate to extensive automation and often use AI and expert systems techniques. However, transfer of rapid prototyping technology into many other areas including human factors has not been realized. Generally, the critical applications incorporate the most advanced tools. For example, the Army-NASA Aircrew/Aircraft Integration (A^3I) research program is developing a prototype of a human factors CAE facility for the design of helicopter cockpits [5]. An automation science research facility at NASA Ames is based on the integration of NASA Ames's human factors and AI capabilities, due to a recognition that both fields are integral to automation science [6]. General Motors is in the progress of transferring the technology for high resolution simulation from Hughes.[2] In addition there have been several academic research projects described in the literature that have concerned rapid prototyping and human factors for areas such as traffic control and information systems [10] , telecommunications systems [9], [4], control panels [7], and generalized human interfaces [11], [8].

1.2 System Overview

Figure 1 is a screendump showing a sampling of the tools available in the rapid prototyping environment in its current state. A designer can simply "drag" around input and output devices such as slides, buttons, dials, alarms, knobs, thumbwheels etc., place them in a window representing the interface, resize them, and connect them to other devices, other applications, or I/O-ports connected to an outside system such as the ADIS system. The environment allows the designer to toggle to a "test interface" mode in which he or she can evaluate the design and also create a running C program. It is also possible to drag in and connect simple sounds like "clicks" and alarms

from a sound kit and to create icons which can be dragged into the design.

The environment is built on Next's InterfaceBuilder, a UIMS bundled with the NextStep operating system. Palettes containing typical software interface devices such as buttons, sliders, and text fields were provided by Next. We extended the library of palettes to contain devices useful for interfacing with ADIS systems such as a rapidly prototypable map and general user-interface devices such as the dials and knobs. Figure 2 displays a screen-dump of custom object palettes. After dragging an object in a window, it can be customized partly using the mouse, partly through an inspector panel. In this way, users with no programming experience may produce new software components to graphically represent instruments and controls in applications.

A graphical object on its own is of little use unless the corresponding code is linked together with other software modules to form a complete program. Therefore a built in facility allows one to connect any instrument or control object to other software modules. For example, a control panel component such as a dial may be connected to a software module that the user has developed to supply input to the dial. That software input module could be a simulation subsystem, or a low level hardware driver used to collect real-world data. In the complete system that we envision the user could integrate alternative simulation components for rapid experimentation.

A classroom setting during two semester-long human-factors courses provided an initial test-bed for empirical observations. Naive users over a short period of time (the sixteen week semester) have been observed using the system successfully, with little or no assistance, to produce new designs for instruments and controls. Thus, the rapid prototyping system has proven to be a valuable instructional tool.

2 An Example of the System in Use

In this section we describe typical usage of the environment from the user's perspective.

Consider the design of the instrument cluster for a new concept automobile. This advanced vehicle will incorporate several new technological advances including Advanced Transportation Management and Traveler Information Systems (ATMS and ATIS)[10]. First, the designer will produce a rough prototype. Using a mouse to select menu items, a blank page application is selected by clicking on the IB icon, starting the Interface Builder which has been preconfigured to include the environment's extensions. The designer decides to begin with instrumentation, so the DIALS palette is selected. Figure 3 is a screendump of the graphical menu of the environment's palettes along with our example project in progress. DIAL type objects which will represent the odometer and rpm indicator are pointed to with a mouse, and dragged into position in a window that will be displayed by the new application. An LCD map display is required for the ATIS; the MAPS palette is selected and a map object is positioned. Figure 4 depicts the map object class palette, and the map object instance that has been added to the project. Also shown is the inspector, which is a display of the methods and variables the object is composed of. Continuing on in this manner other objects are incorporated into the prototypical simulated instrument panel. In addition, a control panel is created using palettes of templates for control objects such as buttons, knobs, trackballs, etc.

The designer decides to increase the size of the map (the default size is the smallest size readable for the 90 percentile). Using the mouse, he or she can click on the object and then manipulate it to "stretch" it into a larger size. The map class may have a constrained width to length ratio so that it will conform to a particular standard for ATIS/ADIS maps in automobiles. After making a few adjustments to some of the other objects, the design is finalized. The development system then automatically generates the software code to represent the objects. Now the designer may proceed to "connect" the instrument and control panel objects to existing software modules that will simulate the I/O. The new application may now be run immediately by using IB's facility for testing an interface.

Given the complete system that we envision, in addition to being able to design new interfaces and applications and connect the object instances

to simulator or hardware I/O modules, one could also run automated or interactive simulations, analyse the data collected during these test runs, and use the results of the analyze to tune the design. The user would iterate through the these steps as many time as is necessary to optimize the design. At any point in the process, he or she may decide to integrate a new class of instrument or control objects, such as a voice recognition object class, or sound synthesis class or an "earcon"[3], for example. One could also use the envisioned environment to optimize not only the design, but the design process or subprocesses such as simulation of test runs. For example, time on a high resolution simulator is a valuable resource. Environment tools could be used to validate a low resolution version of a test-run, in advance.

3 Design Methodology

In this section we describe the design methodology used to develop the environment. A brief introduction to the object oriented paradigm is presented.

3.1 The System Design: An object oriented approach

The object oriented design paradigm has its roots in a number of areas including computer science, artificial intelligence, and philosophy. During the early 1970's, the term object appeared almost independently in various fields of computer science, simultaneously. [1]

An object oriented paradigm is one in which the system to be represented in software is viewed as a collection of objects which have a one-to-one correspondence with objects in the real world or some other problem space, (an object may be intangible). In our case, real-world objects are classified and positioned in a structure that is a directed graph; a hierarchical tree structure in which the most generalized object classes are near the higher levels, or root of the structure, and more specialized object classes are classified at lower levels. An arc in the graph represents an 'is a type of' or 'isa' relation between nodes (and thus an existence dependency). Each node (i.e. object class) inherits attributes from its predecessor node(s). All attributes of all predecessors on the path to a node are inherited, however attributes may be overridden by a definition of the attribute within the node or its closest predecessor. Attributes are generally methods for performing operations. Given a defined class hierarchy, a programmer will declare instances of the object classes. It is the instance of the class or type that performs operations utilizing the methods available for that class of object, during run time. The environment automates the creation of the software code representing an instance and supports online manipulation of object instances and supporting code modules.

A primary reason why we are following the object oriented paradigm is that a class hierarchy can represent the system in a way that mirrors the way designers and engineers think. In addition, the object oriented representation is isomorphic with the behavior of the system, real world objects, and relationships between objects and data. In this way, related information is encapsulated and hidden within each object. Data encapsulation makes it easier to design for iterative enhancement, and the inherent modularity

supports the integration of heterogeneous subsystems. Because data encapsulation associates data with related methods, data handling and information management can be facilitated.

4 System Architecture

In this section we describe the architecture of the environment.

The software/hardware system architecture is depicted in Figure 5. The hardware platform consists of one or more workstations connected by ethernet. A workstation can be any that will run the NeXTSTEP environment. Platform cluster(s) may contain heterogeneous machine types. The NeXT workstations run the Mach operating system in coexistence with BSD UNIX. However, on other platforms such as HP's, Sun's, PCs's, etc the operating system may be different. The PC platform is scheduled to be released in the first quarter of 1993; releases for the other platforms are scheduled for 1994.

NeXTSTEP comes with a user level layer of source code, applications and GUI components. In Figure 5, this is labeled 'The Integration Layer'. The environment code is integrated with NeXTSTEP through two of the component applications in the user level layer, called Project Builder (PB) and Interface Builder (IB). PB is the vehicle for generating new custom objects. IB is the mechanism for manipulating the existing environment objects and incorporating them into the user's applications.

Like all object classes created using NeXTSTEP, the environment's object classes are specializations on the existing NeXTSTEP classes. Figure 6 depicts the environment's class objects and their relationship to the built-in NeXT classes. The environment's classes correspond to the new nodes added to the NeXTSTEP class hierarchy graph (shaded).

All of the environment's object classes inherit attributes from a set of NeXT classes unless an attribute has been overridden. One very useful subset of the inherited attributes is called the Inspector. Inspectors are software code modules which empower the user by allowing the non-programmer to perform two tasks otherwise not feasible. These tasks are: viewing and manipulation of the attributes of an object, and connection of an object's input variables to the output variables of other code modules.

In the NeXT Objective C terminology, input/output variables are referred to as inlets and outlets. Typically the user will utilize the IB and an inspector to put together software components by connecting the inlets and outlets of an object which will represent the human-machine interface. To the user, this corresponds to the pointing and clicking on icons and menus, and 'drag and drop' of objects as described in the introduction.

5 Summary, Conclusions, and Future Directions

In this section we present a summary of our accomplishments, our conclusions, and a brief description of future directions that our research may take.

5.1 Summary and Conclusions

In our view a rapid prototyping system is one which supports the research, design and development process by providing facilities for very quickly simulating the full scale or physical prototypes that have been traditionally used in the application area. We have identified the following goals and features for such a framework. It must

1. support the existing design procedures in the application area,
2. promote evolution of the existing design process,
3. employ integration of heterogeneous subsystems,
4. provide automation and visualization tools,
5. integrate data acquisition tools,
6. accommodate the naive computer user.

In addition to determining the necessary features, we have investigated the feasibility of our generic approach to rapid prototyping human-machine interfaces as applied to advanced driver interface systems. We have addressed support for all of the six features to some extent because we have realized a primary enabling technology (CASE for ADIS). However, we have limited the scope of our in-depth investigation to two of the six features. These are numbers 4., automation and visualization tools and number 6., a user interface for naive computer users. We have accomplished this through transfer of existing CASE technology to the problem domain of rapid prototyping advanced driver interface systems. Empirical observation of the interaction of computer users (undergraduate IEOR students) has been carried out over two (16 week) semesters. In our opinion the observations show that the automation and visualization tools we have developed are sufficient for naive

computer users to rapidly explore and produce a variety of designs for human-machine interfaces and ADIS. The implications of our observations are that the environment has the potential for greatly improving the existing methods of design prototyping ADIS, just as it was observed to do in the classroom test-bed. Significant cost reductions, increased reliability, shorter design cycles and more complete investigation of possible design alternatives are some of the benefits that rapid prototyping of ADIS has to offer. Perhaps most importantly, the designer is empowered to use tools that might otherwise not be directly accessible.

5.2 Future Directions

During the process of investigating the feasibility of our approach, we have become aware of additional issues that need to be addressed.

These are:

1. how to support heterogeneous data formats,

2. the need for a generic mechanism for managing data.

Although there is nothing data format specific in any part of the environment, developing the palette for ATIS maps would have been easier if we had some mechanism predefined for handling disparate data formats. We can expect that specifics required to address this issue will be forthcoming in the form of industry standards. A part of our future research should be continuing to track and assess emerging standards.

Another area of future research must be investigation into a generic strategy for management of data. From one view of the system, data is encapsulated and hidden in object instances. However, we have not addressed the high level management of this data; see Section 3. In the current implementation of the environment we have encoded a small subset of the many possible design constraints, and we have not integrated any modular modeling and simulation components. Therefore it has been feasible for the system to function without a data management subsystem. However, in order to increase the scope of the research we must provide a subsystem capable of handling large amounts of data.

Our future work will include the development of additional object classes to represent a larger variety of objects. We would like to span the set of

objects required to investigate the human factors issues inherent to all of the different kinds of human input/output. There are areas of visual i/o as well as the entire area of sound, and most of the tactile issues that have not been addressed. Once we have a set of objects spanning these , we would like to design and run experiments so that we may characterize the ergonomic constants of the objects as predictors of human interaction with the end user product. This will require expanding the scope of the research to include modeling and simulation components.

We would like to build a complete complement of modeling components that we may use to simulate I/O for objects. In connection with this goal, the methodology for incorporating heterogeneous model components needs to be addressed. Again, data management lies at the root of this issue, and it is the next enabling technology we will need to realize.

As a part of the process, we will determine the requirements, specify, and develop the methods and classes required to manage the environment's data. We will develop our own models and those collected from others to validate the approach as a means for integrating heterogeneous components. In connection with the ergonomic issues described above, some of the models we wish to integrate will be human performance models. Human performance models can be used to implement design rules checking and to investigate the feasibility of developing an intelligent designer's assistant as a part of the system.

References

[1] G . Booch. ***Object Oriented Design with Applications.*** Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[2] Scott T. Wood. Charles M. Enderby. Head-up display in automotive/aircraft applications. ***Electronic display technology and information systems,*** 1992.

[3] M. Cohen and N. Koizumi. Audio windows for sound field telecommunication. ***7th Symposium on Human Interface, October 1991 Paper 2433,*** 1991.

[4] J. C. Tolmie D. T. Henskes. Rapid prototyping of man-machine interfaces for telecommunications equipment using interactive animated computer graphics. In B. Shake1 H. J. Bullinger, editor, ***Human-Computer Interaction:*** *INTERACT'87*, pages 1053-1058, Amsterdam, 1987. North Holland.

[5] et al. Elkind, Jerome I. Use and integration of models. In Julian Hochberg Jerome I. Elkind, Stuart K. Card, editor, ***Human Performance Models for Computer-Aided Engineering,*** page Chapter 3, San Diego, California, 1990. Academic Press, Inc.

[6] Dwain A. Deets Eugene. L. Duke, Victoria A Regenie. Rapid prototyping facility for flight research in artificial-intelligence-based flight systems concepts. ***NASA Technical Memorandum; 88268, 1986,*** 1986.

[7] K. Freburger. Rapid: Prototyping control panel interfaces. In *Proc.* ***of the OOPSLA-87: Conj erence on Object-Oriented Programming Systems, Languages and Applications,*** pages 416-422, Orlando, Florida, 1987.

[8] D. W. Parker J. R. Harris. Evaluation of rapid prototyping methodology in a human interface. In ***Human-Computer Interaction:*** *INTERACT'87*, pages 1059-1063, Amsterdam, 1987. North Holland.

[9] J. D. Gould J. T. Richards, S. J. Boies. Rapid prototyping and system development: Examination of an interface toolkit for voice and telephony applications. *Proc.CHI-86,216-220*, 1986.

[10] P. Jovanis and R. Kitamura. User perceptions and safety implications of in-vehicle navigation systems. In Dept. of Civil Engineering and Davis Transportation Research Group, University of California, editors, ***Vehicle Navigation and Information Systems Conference,*** *1st*, Toronto, Ontario, 1989. VNIS '89 New York, NY: IEEE.

[11] J. Ebert R. Gimnich. Constructive formal specifications for rapid prototyping. In ***Human-Computer Interaction:*** *INTERACT'87*, pages 1047-1052, Amsterdam, 1987. North Holland.

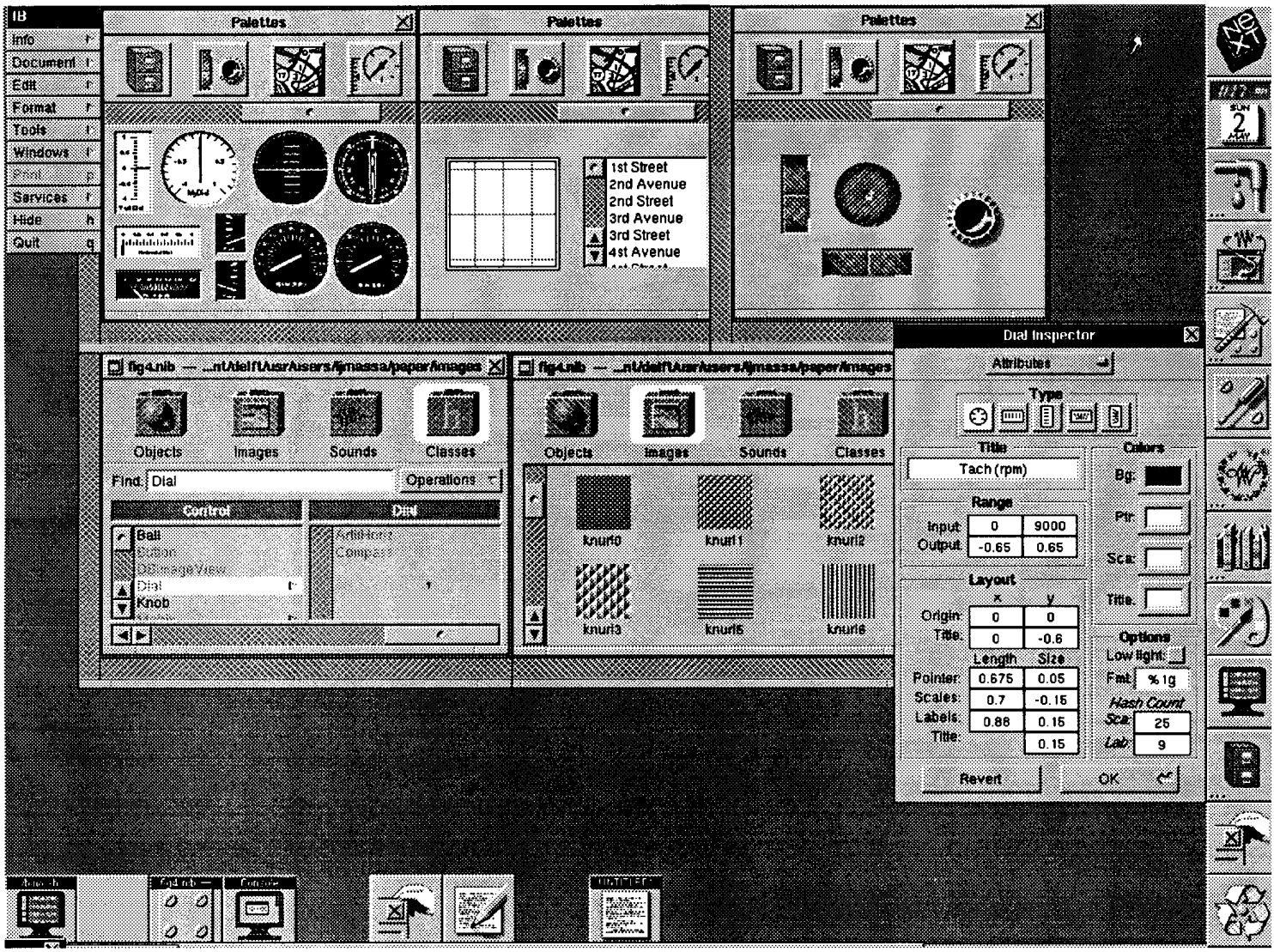

Figure 1 rapid prototyping environment

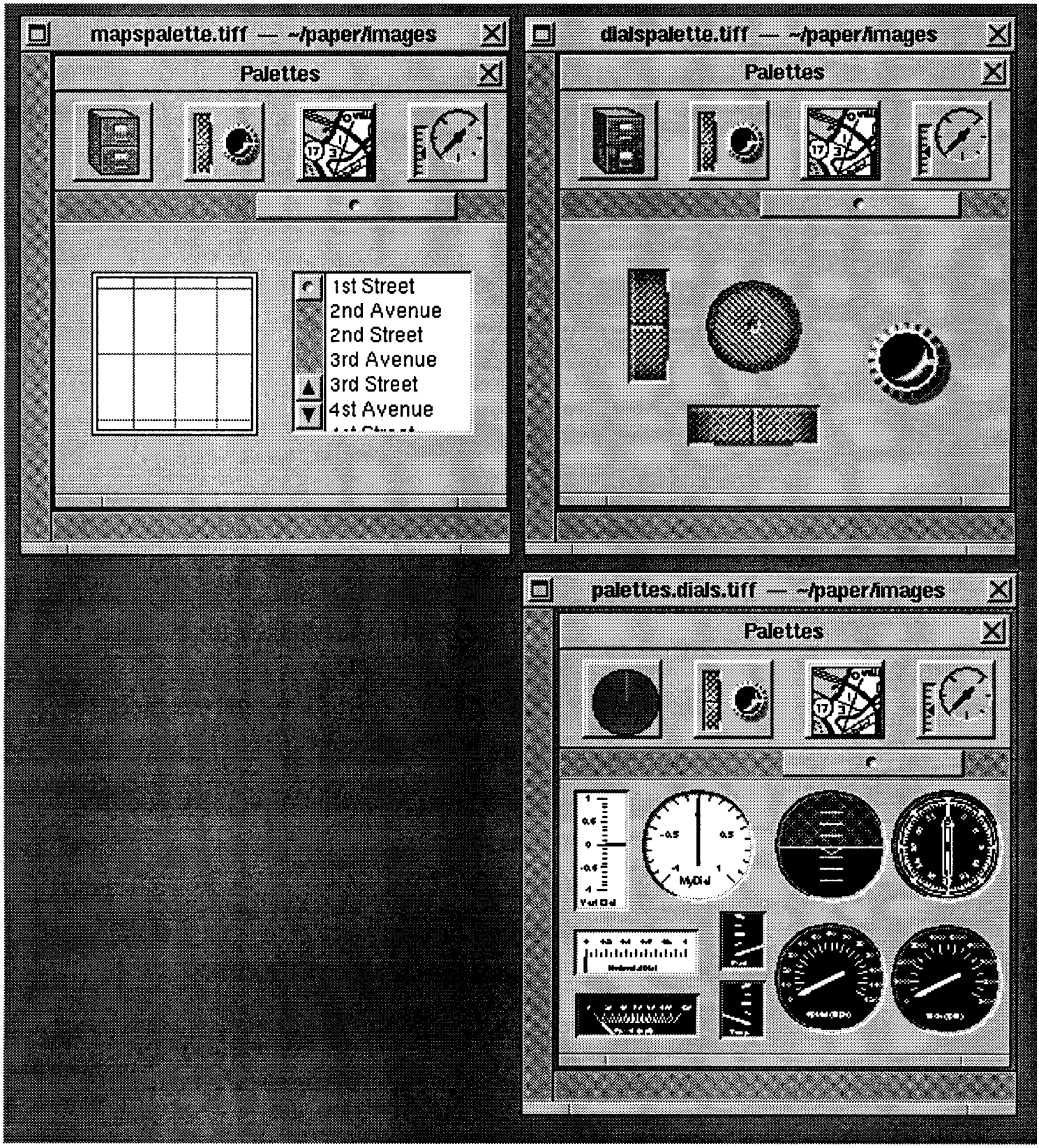

Figure 2 Custom object palettes

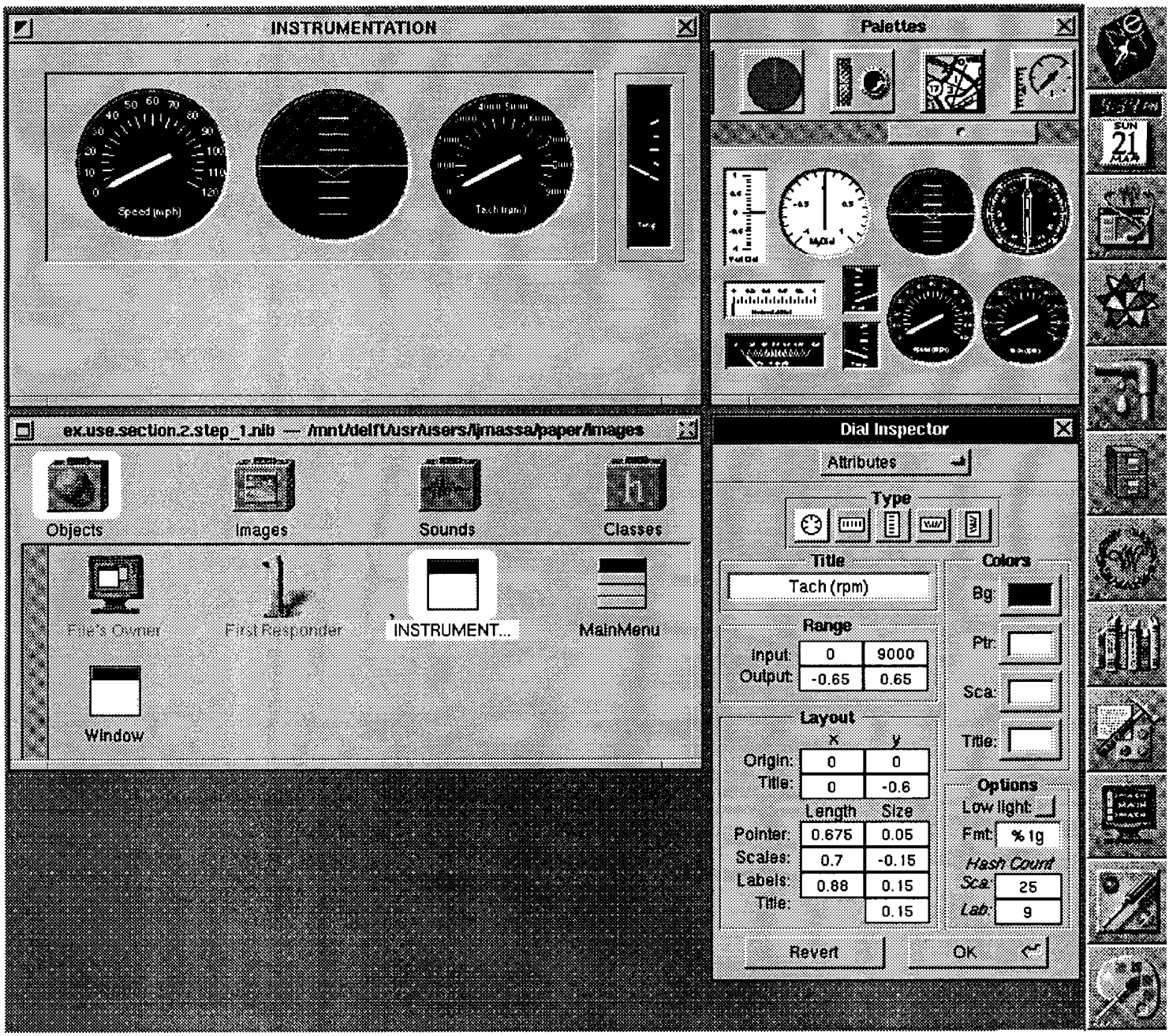

Figure 3 An instrumentation project

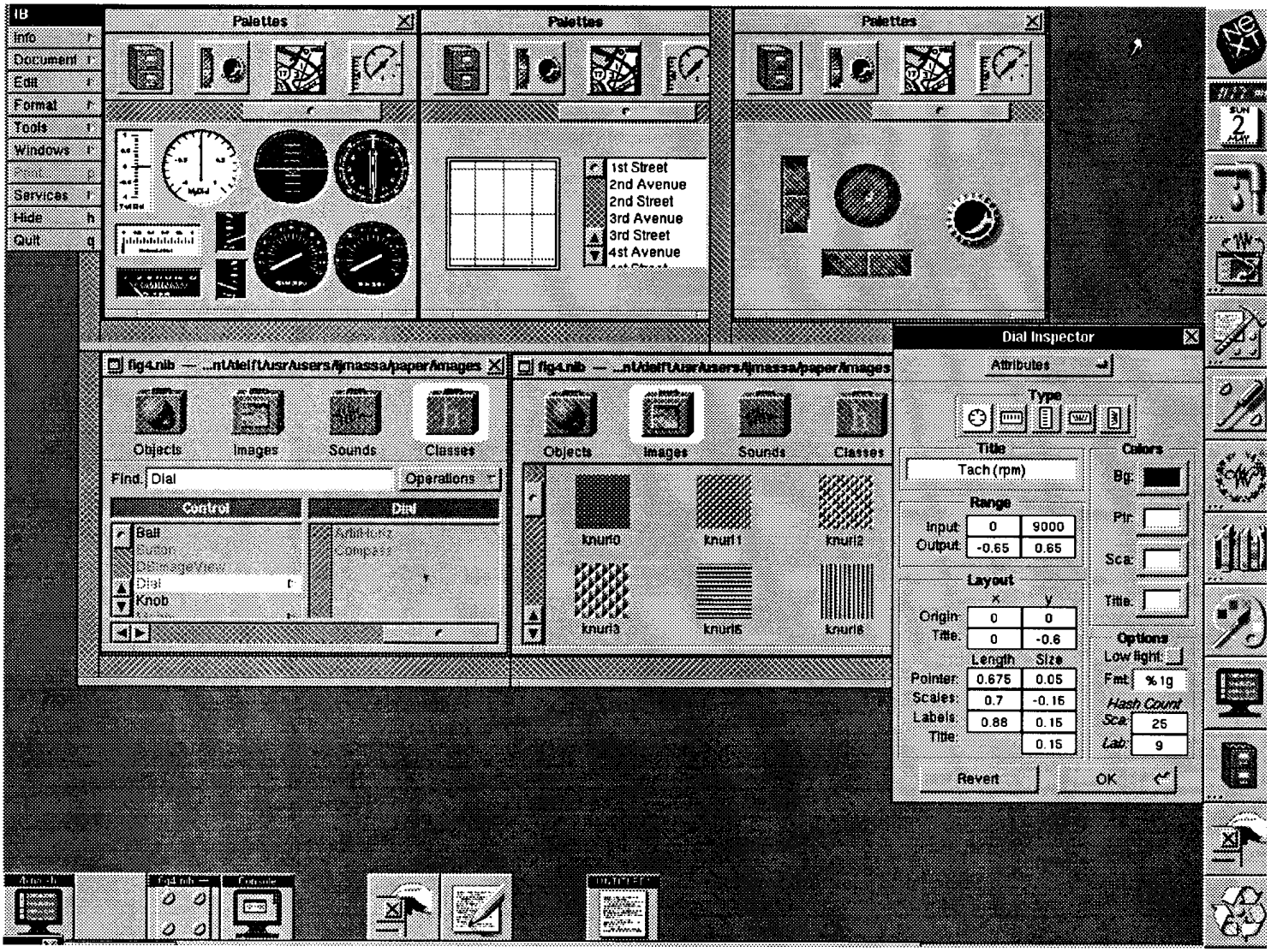

Figure 4 The instrumentation ATIS project

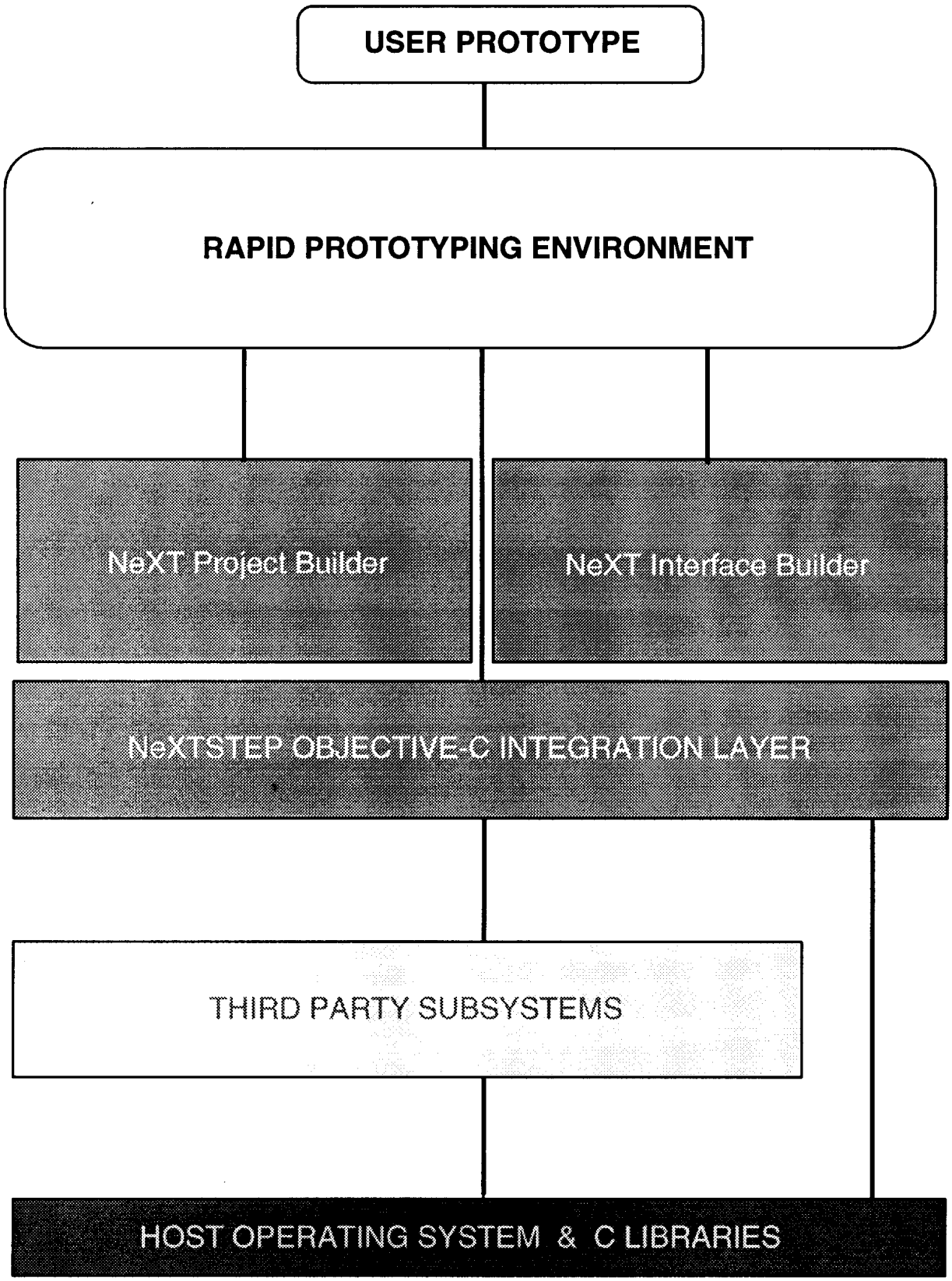

FIGURE 5 SYSTEM COMPONENTS AND RELATIONSHIPS

LJM '93

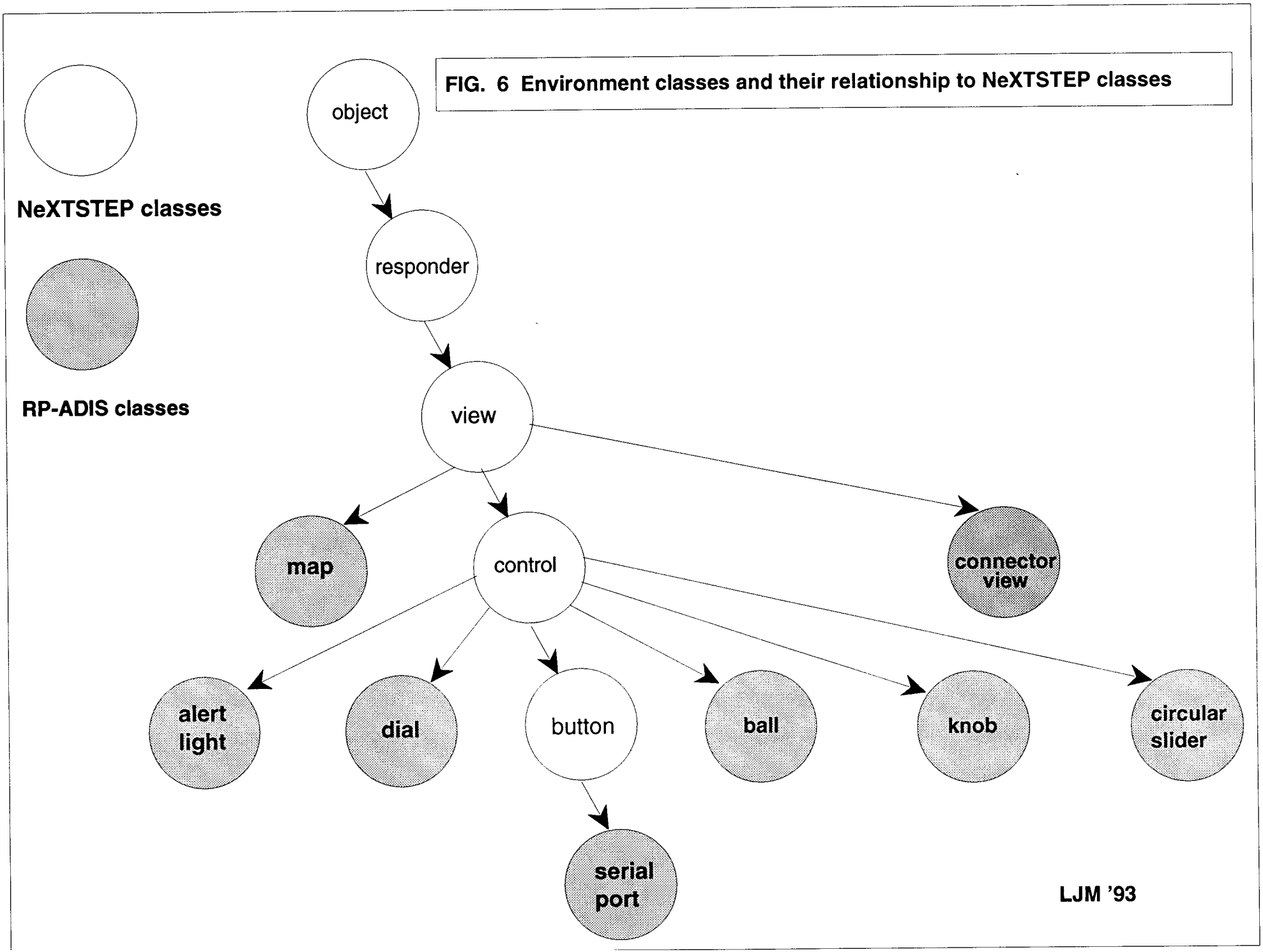

FIG. 6 Environment classes and their relationship to NeXTSTEP classes