THE MULTIPURPOSE PRESENTATION SYSTEM *

Gerald A. Wilson, Eric A. Domeshek, Ellen L. Drascher, Jeffrey S. Dean

Advanced Information & Decision Systems (AI&DS)
201 San Antonio Circle, Suite 286
Mountain View, CA 94040

ABSTRACT

This paper describes a knowledge-based user-oriented interface system which is unique in its combined use of data base, graphics, and AI technologies. All of the data presented to the user, as well as all of the data input by the user, is maintained and used as abstract "objects" using an AI object-oriented knowledge representation approach. However, the system employs a relational data base as both the storage media for all of the object information, and as the interface between the application and interface systems. The object-oriented approach allows both the application system and the user interface to operate upon the data, describe image layouts, manipulate viewing windows, and build display images using high-level commands. The use of the relational data base provides a common interface between many types of application systems and the user interface, and helps to maintain a clean separation between the functions of the application system and the user interactions.

1. INTRODUCTION

Computers are being used more every day by people who have a job to get done but who have little or no desire to be programmers. This has helped to increase the need for "user friendly" interfaces: computer interfaces which are easily used and difficult to confuse. Substantial effort has been expended on developing mechanisms for displaying information in novel and easily perceived manners, and for using a variety of non-keyboard input devices. Good interfaces are time consuming and difficult to develop. Interfaces also tend to become intimately enmeshed in the application systems which they support. A problem which has received little attention is how to design interfaces which have both good human user versatility and are easily integrated into application systems. To truly reduce the time and cost of developing custom user interfaces, an interface must provide display hardware independence, as well as being user and application system friendly. The Multipurpose Presentation System (MPS) described in this paper is a versatile user-oriented interface which directly addresses the problems of rapid system development by an integrated use of graphics, data base management, and artificial intelligence concepts.

The objective of MPS is to provide a utility through which a wide variety of application systems can accept and present data by describing the data as objects and object attributes. One part of this objective is to make the interaction between the application system and MPS clean, well defined, and implementation language independent. At the same time, MPS must have multi-media input and output capabilities, and be as device independent as possible. The other part of this objective is for MPS to also be versatile and easy to use from the user´s viewpoint.

The MPS approach frees the application systems from the details of handling numerous user interface devices, yet allows full use of the capabilities of these devices. MPS allows a natural, object oriented, declaration of the information which is to be transmitted between the application system and the human user, thus facilitating separation of the interface functions from the application system functions. By minimizing and isolating the communication between the interface and the application system, the MPS and the application system are logically

* This work was supported by the Office of Naval Research under contract number N00014-82-C-0602.

distributed, and thus can easily be physically distributed. By incorporating a library of generic layouts, objects, and operators, the application programmer is relieved of much of the usual difficulty in building good data presentations.

A conscious choice of the term "multipurpose" rather than "general purpose" was made in naming MPS. We neither claim nor intend that MPS be an interface system to support all types of application systems. The choice of an object-oriented approach to interaction between the application system and MPS limits somewhat the type of control and low-level graphical device control which can be accomplished rapidly. This means that MPS would not be appropriate for highly interactive tools such as screen editors or animation systems. There are likely to be other large classes of application systems which MPS will have difficulty supporting. However, we believe that the class of application systems which can be supported by MPS is quite large and important.

MPS consists of four major components: a relational DBMS, a presentation surface builder, presentation surface effector, and a library of generic objects. Presentation surfaces are detailed descriptions of information to be entered by a user and/or presented to the user. MPS treats presentation surfaces as blackboards on which both the user and the application system may write and which both may read. Presentation surfaces are represented internally as a collection of relations maintained by a relational DBMS. Communication between the application system and the MPS is done through the relational DBMS. The contents of presentation surfaces are described in terms of combinations of objects. Communication between the human user and the MPS is handled by the presentation surface effector, which adapts to the capabilities of the interface hardware. Each of these is described in more detail below.

2. SYSTEM OVERVIEW

The MPS provides an extension to the normal programming environment through which application systems can achieve a broad variety of interactions with users. Given our objective of an easily used and elegant interface, great care must be taken on both the application system and human user sides of the interface. Our approach is to allow both the human users and the application system to reference and manipulate data items as "objects" and "object attributes", where our notion of an object is analogous to that of object-oriented programming languages such as SMALLTALK[3]. A better understanding of this approach can be gained by examining a brief scenario.

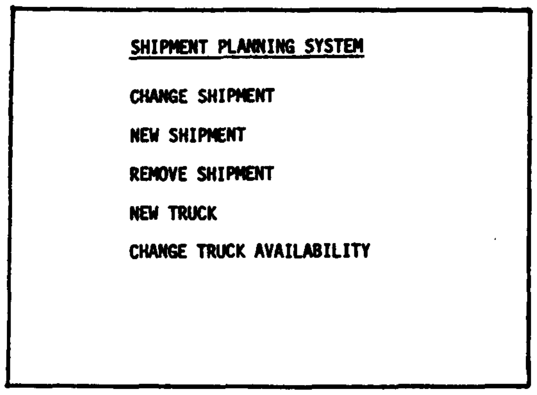

Figure 1: Route Planning Example -- Top Level Menu

2.1 AN EXAMPLE OF MPS USAGE

Suppose that you, our applications expert, have developed a very fast and efficient route planning system for transportation networks. Your system uses a combination of linear programming and heuristic control methods to provide a highly efficient solution which can be dynamically replanned after partial execution (e.g. when a new shipment must be added or an old one deleted the original plan can be altered rather than completely redone.). Now you would like to make your system more usable by adding a convenient user interface. Let us examine a possible scenario of interface behavior from the user's perspective.

To begin you want the system to present a simple menu of operations such as that shown in Figure 1. The user selects an operation from the menu by moving the cursor to the desired operation and pushing the "execute" button on the keyboard. To highlight the operation currently selected, the text on the line pointed to by the cursor is shown in reverse video. Each time the user pushes the space bar on the keyboard the cursor is moved to the next operation on the screen, or moved to the first item if it previously pointed to the last item.

Having selected an operation, "new shipment", the user is given a form to fill out such as the one shown in Figure 2. The user can enter information in the fields in any order, using the right, left, up, and down arrows on the keyboard to move from field to field. Completion of an entry is indicated either by pushing the return key or by moving the cursor outside the field.

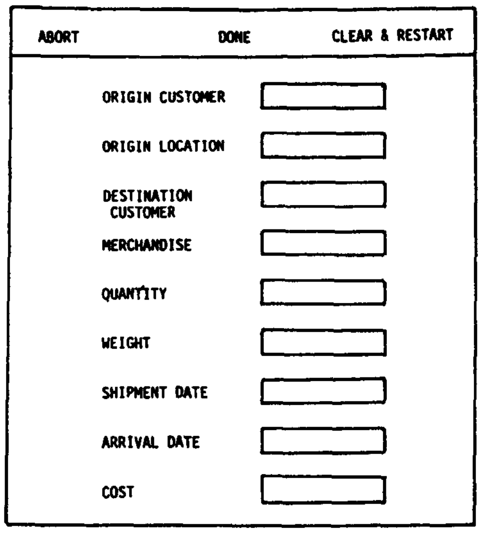

Figure 2: Route Planning Example -- New Shipment Specification

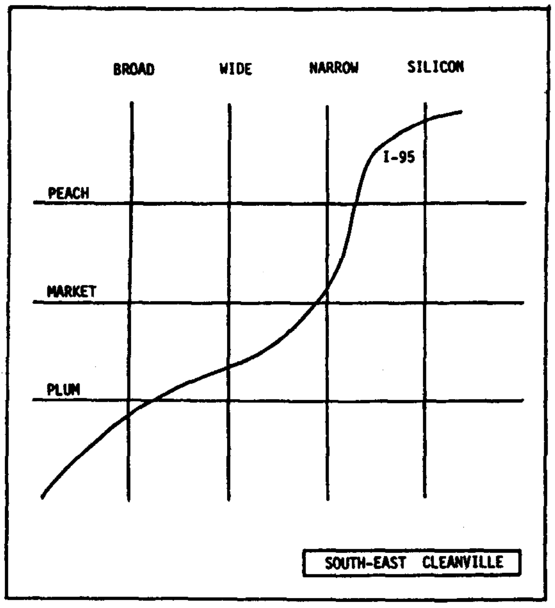

Figure 3: Route Planning Example -- Location Map

Several interesting things are happening as our user is entering data into this form.

Each time data is entered into a field, a collection of associated integrity constraints specified by the application system are checked. These checks vary from simple ones, such as verifying that the data is in the proper form, to complex ones, such as checking that the name of the shipment destination is one known to the application system. If an integrity constraint fails the application system could request either simple actions, such as showing an error message and returning the user to the incorrect field for editing, or complex actions. For example, in response to an integrity constraint failure on the shipment destination entry, MPS might ask the user if this is a destination not previously served. If the answer is "yes", the user would be presented with another form on which the details about the location and nature of this new customer were to be entered. Because the application system can provide these integrity constraint specifications as a part of the definition of the desired user interface characteristics, all of this run-time interaction with the user can happen without intervention from the application system.

When the user enters the shipment destination, the location field is immediately filled in by the interface as long as it is unambiguous (for example, a company with only one location). If the party receiving the shipment needed delivery at another location, the user could move to the location field of the form to modify the default value inserted. Upon moving to that field on the form the user would be presented with another display consisting of a map of the region served overlayed with a grid, as shown in Figure 3. Here we are assuming that the application system needs locations in terms of grid-cell identifiers, rather than street addresses. The user can enter the desired delivery location by moving a cursor on the screen, using a "mouse" type of pointing device, to the correct grid cell and then pushing the button on the top of the mouse to confirm the selection.

If the user moved to the "weight of shipment" field of the form and pushed the "help" key, the interface would present any guidance specified by the application system for that entry. If none had been provided it would present the guidance for the form in general. Similarly, if no help information had been specified for the form, MPS would attempt to present the help information for that operation, as selected at the top menu level.

Throughout this scenario fragment, <u>none</u> of the menus, forms, text, pictures, or constraints were built into MPS. All of that information was provided either from the MPS object library or by the application system. Material to be presented to the user is stored in a relational data base available to both the application system and MPS. Data entered by the user is returned by MPS to the application program by means of appropriate relations in the data base. The means by which this is accomplished is the subject of the sections which follow.

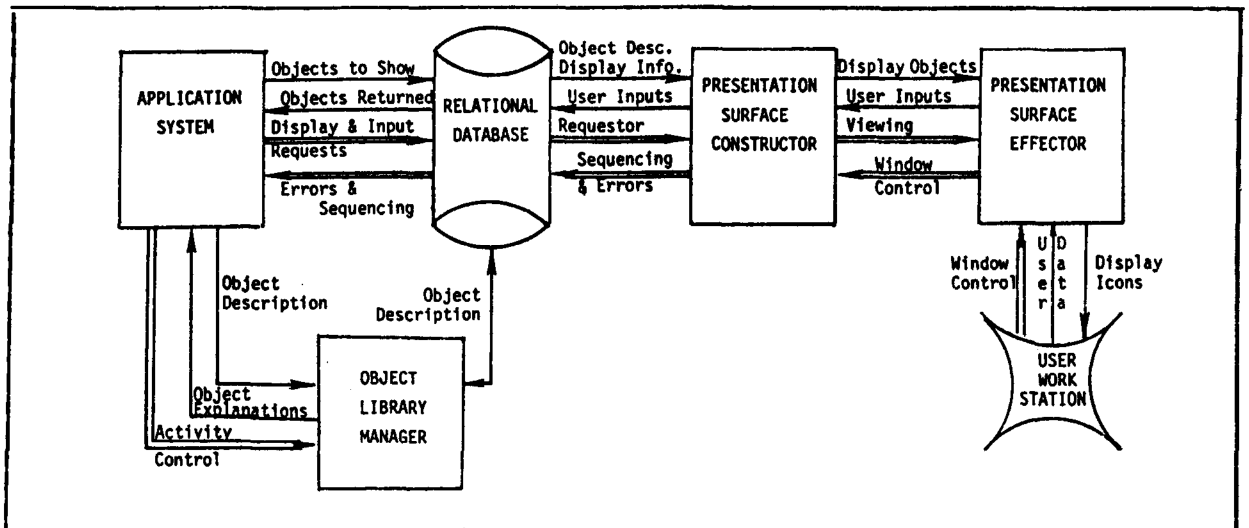

Figure 4: MPS System Architecture

2.2 THE STRUCTURE OF MPS

The major components of MPS, together with their data and control flow relationships, are illustrated in the system diagram of Figure 4. Most of the data flow shown in the Figure deals with objects of various types. There are three types of objects used by MPS: generic, instance, and display. Figure 5 illustrates the relationships amongst these object types. Generic objects are essentially parameterized descriptions of items which might be used to construct images to be shown to and manipulated by the user. Specific instances of generic objects are called instance objects. Generic objects are thus declarations of objects and instance objects are instantiations of those declarations. Display objects are device independent representations of objects in a form appropriate for output. They are the presentable form of instance objects and are created by MPS using special knowledge associated with the corresponding generic object. These can be presented to a user on any given output device using the device dependent knowledge contained in the device context to map the display objects into electronic images.

Many activities are performed by the MPS in support of the bi-directional interaction between users and application systems. These activities fall into four groups: generic object definitions, instance object specification, display object building, and interaction handling. A general understanding of these activities can be gained by examining what would be required to construct a picture such as Figure 4.

Definition of the generic objects to be used includes both the use of existing generic objects in the MPS library and new objects built upon the library objects. Because a presentation surface

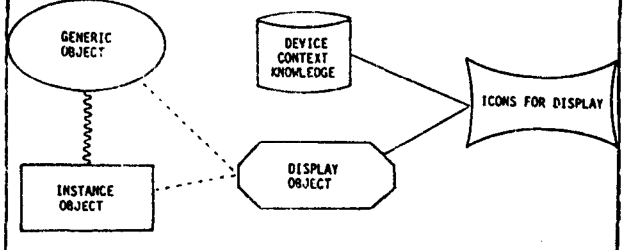

Figure 5: Relationships among MPS Object Types

is just a type of generic object, the definition of generic objects includes the definition of the contents and organization of each of the general presentation surfaces required. In building Figure 4, the generic objects are squares and rectangles and cylinders containing text, two types of arrows, and a presentation surface, which we will call "system-diagram", consisting of boxes and cylinders and arrows which are to be arranged in locations to be specified later. A constraint which might be added to the "system-diagram" object definition would be that no lines should be positioned such that they cross.

The second activity is to create the specific instances of boxes and cylinders, one for each box and cylinder to be shown on the final figure, and the specific arrows connecting those items. This is done by storing the complete specification of each of these instance objects in the data base, and associating each with an instance of the "system-diagram" object. This creates a specific instance of a presentation surface which constitutes a high level description of the desired picture.

The third activity is to create a displayable form of the presentation surface instance in a display device independent manner. This is done by using the instance objects and graphical layout templates associated with the generic objects. These provide the mapping of the required instance objects composing the presentation surface instance into display objects. To actually show this presentation surface instance to a user we must send it to a user-viewable media.

The fourth set of activities, interaction handling, involves the direct, device dependent, I/O with the human users. Pictures are created by mapping the display objects onto the desired device under the guidance of the device context. Inputs from the user are accepted by mapping the device specific actions taken by the user into selections of and modifications to object instances. The results of user inputs are then stored into the data base, thus making them available to the application system.

Generic and instance object definition activities of MPS are described in Section 3. Activities related to presentation surfaces and display objects are discussed in Section 4. Section 5 covers the various aspects of interaction handling.

3. EVERYTHING IS AN OBJECT

The concept of an object is central to MPS. Everything which is presented to a user and everything which a user presents to the interface is an object. The notion of an MPS object thus is modelled after the more generic concept of data abstraction, and employs many of the principals of structured semantic networks.

3.1 GENERIC OBJECTS

Let us consider first the formal definition of an MPS generic object.

A generic object is a data structure which includes: a description of the attributes and components which collectively comprise instances of the object; the structure of display objects which represent the external presentation of the instance objects; and constraining properties which apply to instance and display objects. It must satisfy exactly one of the following:

a. It is a primitive object.

b. It is a compound object composed of two or more generic objects which are combined using well defined operators.

A primitive object is a generic object which is fully defined without reference to other objects.

The slight circularity of these definitions is intentional. While we believe that there is probably a set of primitive objects which could be defined from which all other objects of interest could be built, there does not appear to be any advantage to fixing the set of primitive objects. In fact the advantage of this definition of primitive object lies with the ability to define the most advantageous set of primitive objects for a particular use of the MPS. Note that primitive objects can still have parameters and use operators. This allows us to declare a square to be a primitive object, and yet to leave open the length, color, and line-widths of this object until a specific instance of a square is desired.

Consider the partial definition, illustrated graphically, of the object SHIP shown in Figure 6. There are six portions of this definition: attributes, components, constraints, graphical properties, graphical layout template, and sub-objects. Each portion captures important information used by MPS.

3.1.1 Attributes and Components of Objects

Attributes are properties of the object as a whole. Attributes, pictured in Figure 6 in the single rectangle, are typically what makes one instance of an object different from another. Attributes are also objects, and can thus be addressed, examined, and modified by either the application system or the user. Some attributes will be quite simple objects having only a single numeric or textual value and possibly an associated specification of units. Others may take sets of one or more values. Still others may be complex structures composed of other attribute objects. For example, the location of a ship might be described as a single attribute of the SHIP object, with location being an object composed of a latitude and a longitude component. To simplify our examples we have not used any complex attributes.

It is sometimes useful to think of the collection of attributes which describe an object as one "view" of the object. Another somewhat different view of an object is the description of

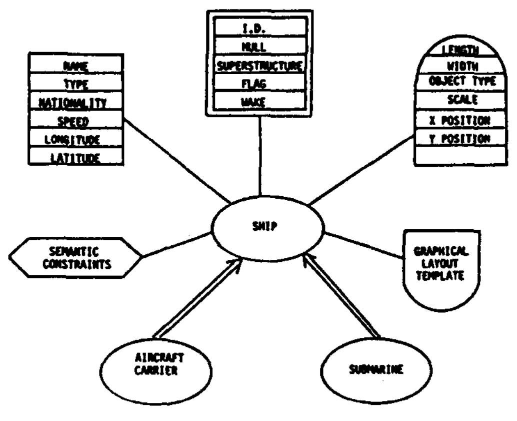

Figure 6: A Conceptual Illustration of MPS Objects

the component parts (shown in Figure 6 by the double walled rectangle) which together compose the object. Each of these component parts is itself an object with its own description. Not all components of a generic object description necessarily correspond directly to physical components of the real-world object. This occurs because each icon presented to the user must have a corresponding object. For example, the component WAKE, though not physically part of a real-world ship, is used to represent the speed and direction of the ship graphically.

3.1.2 Constraints

To assure that all of the real-world semantics of the object are captured we must add the third portion of the object description, the constraints. Constraints specify conditions of, and relationships between, attributes and/or component parts of an object which must be maintained. MPS attempts to provide good constraint checking capabilities, but not to remove all need for constraint checking from the application system. Constraints not handled, or not desired to be handled, by MPS can always be addressed by the application system.

Although constraints, illustrated in Figure 6 by the hexagon, are logically thought of as a distinct part of an object definition, they are, as shown in Figure 7, directly associated with the attributes and components of an object. MPS currently includes five types of constraint specifications: mode, number, generic object, default, and semantic. These five appear to support the current MPS applications; additional types of constraints can be added if the need arises.

<u>ATTRIBUTE REPRESENTATION</u>

name	mode	#	default	gen.obj.	semantic constraint
SPEED	optional,integer	1	20	SPEED	sc-1
NATIONALITY	required,text	>0		NATIONALITY	sc-2
NAME	required,text	1	nil	NAME	sc-3
TYPE	required,text	1		TYPE	sc-4
LONGITUDE	required,integer	1		LONGITUDE	sc-5
LATITUDE	required,integer	1		LATITUDE	sc-6

<u>COMPONENT REPRESENTATION</u>

name	mode	#	default	gen.obj.	semantic constraint
I.D.	required,protected	1		SHIP-ID	sc-7
HULL	required,protected	1	NULL	SHIP-HULL	sc-8
SUPER STRUCTURE	required,protected	1	nil	SHIP-TOP	sc-9
FLAG	optional,protected	<3		NATIONALITY	
WAKE	optional,updatable	1		SHIP-COURSE	

Figure 7: MPS Attribute and Component Representations for Objects

The "mode" constraint specifies whether a given attribute or component of an object is required or optional, its type (integer, real, textual, complex, ...), and whether or not it may be altered by the user. The "number" constraint specifies the number of values which the attribute may take at any one time. A "generic object" constraint is a pointer to the description of the generic object which fully defines this attribute or component of the current object. In our example, SPEED is an optional attribute which can take on only one value, while NATIONALITY is a required textual attribute which can take on multiple values (i.e. ships may fly more than one flag), but must have at least one value.

The "default" constraint is used primarily as an aid to ease the burden of defining an instance object. When an object instance is described to include a given attribute or component, but with no value given for that attribute or component, MPS can use the "default" constraint to assign an appropriate value. Thus if the nationality of a ship was not given, MPS could use the default value of "US"; if the type of hull was not specified MPS would use the default of "cargo-ship-hull". As shown in Figure 7, the "default" constraint is empty for those components or attributes which are required, but have no possible default and thus must be provided by the application system or the user. When the attribute or component is not required, it is "nil" for those cases for which the default is an empty symbol, thus indicating that a value exists in the real-world, but is unknown at present. An empty entry in default is used in this case to mean that the attribute or component should not be used in creating display objects unless it is explicitly given a value in the instance object.

The constraints discussed above deal only with the specific attribute or component to which they were attached. Semantic constraints deal both with rules about single attribute or component values and with relationships among attributes and components. In our example SHIP object definition we might restrict NATIONALITY to one of the known countries, restrict SPEED to less than 30 knots when not in port and zero otherwise, and require that the HULL component be consistent with the type of the ship. Semantic constraints such as these will require a rich language for their specification. The nature of such a language is too lengthy a topic to discuss in this paper. MPS employs a language based upon an extension of first-order predicate logic which was developed in earlier work by Wilson[11] dealing with semantic integrity of data bases.

3.1.3 Graphical Properties and Layout Templates

The combination of all the attributes plus all the component parts plus all the constraints fully describe an abstract object. They do not, however, describe how that object may be displayed. The description of the graphical representation of the object is specified by the graphical properties of the object. The graphical properties provide a mapping between the attributes of the generic object and the desired collection of device-independent display objects which may be used to represent instances of the generic object. The approach used in MPS is to consider the displayable form of an object as consisting of one or more display icons which are assembled using graphical operators in accordance with the specifications of the graphical layout template, and placed upon a rectangular surface with a clear background.

3.1.4 Object Relationships

The relationships between objects, illustrated in Figure 6 by the double line arrows and the generic object pointers to attributes and components, are both quite important, and are often exploited in MPS by means of inference. The later relationship is that of a generic object appearing as a component or attribute of another generic object. This was discussed above as a form of constraint upon the larger object, where the component of the larger object must be constrained according to the definition of the generic object which fully defines the component. The former relationship is that of "super-object". This is precisely a set relationship where if "A" is the super-object of "B", then all of the instances of "B" are a subset (not necessarily proper) of the instances of "A". MPS requires that immediate sub-objects (i.e. ones with no intervening super-object relationships) be formed from the super-object by restricting the values of one or more of the super-object attributes. The attributes so used are termed the "discriminator" for the sub-objects.

3.1.5 Uses of Implication

Some or all of the attributes, components, and constraints of an object may be inferred from its super-object. Two types of inference are employed in MPS. The first type of inference is that of inheritance. Inheritance is the process of passing down the definition tree, from super-objects to the more restricted objects, all of the attributes and components of the super-object. The only exceptions to inheritance are those attributes or components specifically excluded from inheritance. This is illustrated in Figure 6 by the relationships between "aircraft carrier" and "ship" and between "sub" and "ship". All of the attributes and components which apply to "ship" apply equally as well to both "aircraft carrier" and "sub". The discrimination made in these different objects is that the "type" attribute of "ship" has been restricted in each of the other two objects, thus "ship" is the super-object of "aircraft carrier" and "sub". This use of inference provides a great simplification in the description of generic objects. We need only specify attributes and components of a sub-object when they are different from those of the super-object. Great advantage is taken of this in providing aids to the application system in the definition of new objects.

The second type of inference used by MPS is a slightly different form of inheritance: attribute/component inheritance. This is used to capture the relationships between the attributes and the components of a given object. For example, for our "ship" object its nationality, speed, and location attributes apply both to the ship as a whole and to each of the component parts of the ship. MPS knows not to apply the TYPE attribute directly because the semantic constraints associated with each of the components of the SHIP object explicitly specify the relationship between the ship TYPE and the TYPE of each of the components. As with the first type of inheritance, this approach reduces both the

Unique ID	Speed	Nationality	Name	Type	Long	Lat
31156	30	USA	Kennedy	Aircraft Carrier	30	72
31157	25	USSR	Minsk	Aircraft Carrier	57	139
31158		BRITISH	Kon-Tiki	Raft	245	10

Figure 8: Examples of MPS Instance Objects for the SHIP Object

effort and space required to specify the properties of each of the MPS objects. This type of inference is termed belief satisfaction in MPS.

3.2 INSTANCE OBJECTS

Having fully described generic objects, we must address instance objects.

An <u>instance object</u> is a generic object in which all of the required parameters have been filled with specific values.

Instance objects are thus simply instantiations of generic objects. Instance objects are represented in MPS as entries in the relational table of the relation corresponding to the appropriate generic object. Some examples of instances of ships are shown in Figure 8.

The values of the attributes of any instance object may be specified by either the application system or the user. This unique uniformity in the handling of user and application system inputs is made possible by the object oriented approach together with the use of the data base as the communication medium.

3.3 OBJECT REPRESENTATION

While the type of graphical representations of objects used in Figures 6 and 7 are convenient for paper presentations, they are difficult to capture and use in a computer system. Even more importantly, a cornerstone of the MPS approach is that MPS and the application systems should be equally able to reference and build the generic and instance object descriptions, even if different programming languages and different machines are used. The approach which we have employed to make all of the object descriptions universally available is to represent all of the objects, attributes, components, etc. in a collection of relations managed by a relational DBMS. The collection of relations which capture the object descriptions of Figures 6 and 7 are shown in Figure 9.

<u>ATTRIBUTE Relation:</u>

Attribute	Object	Mode	Default	#	Constraint
SPEED	SPEED	optional,integer	20	1	SC-23
NATIONALITY	NATIONALITY	required,text		>0	SC-24
NAME	NAME	required,text	nil	1	SC-25
TYPE	TYPE	required,text		1	SC-26
LONGITUDE	LONGITUDE	required,integer		1	SC-27
LATITUDE	LATITUDE	required,integer		1	SC-28

<u>COMPONENT Relation:</u>

Compobj	OfObj	Mode	DefaultObj	DefID	#	Constraint
ID	SHIP	required,protect			1	SC-32
HULL	SHIP	required,protect	HULL	31154	1	SC-33
SUPER STRUCTURE	SHIP	required,protect	SHIP-TOP	nil	1	SC-34
FLAG	SHIP	required,protect			<3	SC-35
WAKE	SHIP	required			1	SC-36

OBJECT Relation:

Obj	Type	Template
SHIP	OBJECT	SHIP_TEMP
ID	TEXT	TXT_BLOCK
HULL	ICON	HULL_TEMP
SUPER STRUCTURE	ICON	SS_TEMP
FLAG	ICON	FLAG_TEMP
WAKE	OBJECT	WAKE_TEMP

Figure 9: Example MPS Object Representations

Our approach employs standard relational calculus descriptions of the MPS objects, with the exception of the templates to define the physical icon construction corresponding to an object. The relational tuples describing generic objects are maintained permanently in the data base, maintained in our current implementation by the Troll relational DBMS[6]. The templates used to describe the physical icons and operations for combining icons are maintained in a separate dictionary indexed by the names of the individual object templates. As instances of objects are created, they are also stored in the appropriate relational tables in the data base.

While the bandwidth required for information transfer between the application system and the data base is fairly low, the bandwidth between MPS and the data base must be quite high as the icons to be presented to the user are being constructed or modified. This would place a great strain on the DBMS if all of this communication required disk access to the data base. Thus it is essential that the relevant portions of the data base be maintained in an in-core working area for rapid access. To further improve the efficiency of use of the generic object descriptions, it is important that the implied links, represented in the relations by joins between tuples, be made explicit. Much more rapid traversal of the explicit links can be made than would be possible with a series of joins.

In the current implementation the in-core working area is maintained by an in-core data base facility called PEARL[1], which is an AI knowledge representation system intended to incorporate a full DBMS. We have combined PEARL and Troll into what appears to MPS to be a single system by providing a mapping between the internal structures of PEARL and the external relations defined for Troll. Thus, from the view point of MPS, the combined Troll/PEARL system is an efficient relational DBMS capable of representing limited forms of semantic networks directly. A few special purpose additions to the DBMS have been made to allow the efficient handling of signals passed between the application system and MPS to indicate such conditions as the completion of an update of an object on the screen. Such extensions to standard relational DBMS's would be required to support MPS with reasonable computation times during the building of presentation surfaces. These extensions clearly do not violate the nature of the relational model.

4. PRESENTATION SURFACES AND DISPLAY OBJECTS

Having defined the concepts of generic and instance objects, and described their representation in MPS, we can describe how to construct scenes to be presented to the user. We use "scenes" here to mean logical descriptions of images to be presented on some form of display. Scenes are constructed in MPS in the form of presentation surfaces, which are a special type of generic object.

4.1 DEFINITION OF A PRESENTATION SURFACE

The specification of the potential contents of a presentation surface (in terms of generic objects composing the presentation surface) and organization (in terms of the physical layout of the objects) can be fully described using the representations presented in Section 3. The generic object specifications provide a recipe for constructing a scene for display. To create an actual scene we must add the required types of ingredients and then mix them according to the recipe. Adding the ingredients is the process of creating an instance of the presentation surface plus instances of each of the objects composing the presentation surface. Mixing the ingredients is the process of creating display objects by combining the ingredients in accordance with the instructions provided by the graphical layout template.

Instance objects are created, as described in Section 3, by storing tuples into the relational tables of the appropriate generic object. However, these tuples specify only the values of the attributes of the generic object. For instance objects which are not presentation surfaces, such as instances of SHIP, MPS normally uses default components. This is possible because the default components are typically primitive objects existing in the MPS object library. There cannot be default components for presentation surface components. The components of a presentation surface instance are normally instance objects created during the execution of the application system, often after the creation of the presentation surface instance.

Therefore, to complete the specification of the presentation surface instance, each of the instance objects composing the presentation surface instance must be linked to the appropriate presentation surface instance component. This is done by storing tuples in the Presentation Instance Component (PIC) relation in the data base. PIC is defined as:

PIC (pid, pres-obj, goid, gen-obj, protection)

where:

pid	=The unique id of the presentation object instance being built.
pres-obj	=The name of the generic presentation object.
goid	=The unique id of the instance object to be included in this presentation object instance.
gen-obj	=The name of the generic object of which goid is an instance.
protection	=The flag indicating whether the user may modify this object at this time.

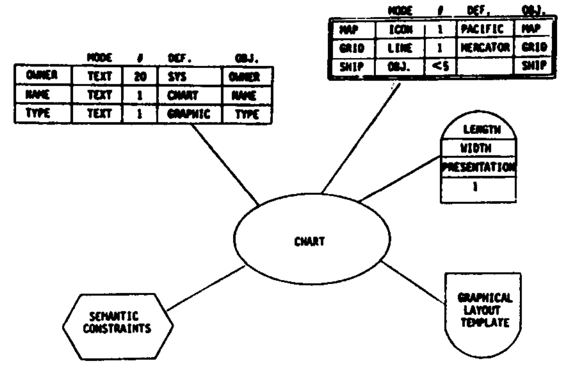

Figure 10: The CHART Presentation Object

Before specification of the ingredients is complete, we must add some information about the way in which the instance objects are to be displayed. Instance objects are created by instantiating the attributes of the generic object. This leaves the graphical attributes unspecified, except for defaults. The graphical attributes for all instance objects are represented in a single relation named "DISPLAY" in the Troll data base. The DISPLAY relation is defined as:

DISPLAY (unique-id,property,value)

where:

unique-id=The unique internal identifier of the instance object.

property=The name of the display property being addressed.

value =The value to be assigned.

Using the unique id of the instance object, MPS uses the description of the property associated with the corresponding generic object to check that the property is appropriate, and the value acceptable. Some of the display properties which are currently used by MPS objects are:

x-pos =The x coordinate of the instance object within the next higher level object of which it is a component. Usually not updated by the application system.

y-pos =The y coordinate of the instance object within the next higher level object of which it is a component. Usually not updated by the application system.

OBJECT:

Obj	Layout Template	Type
CHART	CHART_TEMP	PRESENTATION
MAP	MAP_TEMP	ICON
GRID	GRID_TEMP	LINE
SHIP	SHIP_TEMP	OBJ.

COMPONENT:

CompObj	OfObj	Mode	#	DefaultObj	DefaultID	Constraint
MAP	CHART	ICON	1	MAP	024569	SC-54
GRID	CHART	LINE	1	GRID	004231	SC-55
SHIP	CHART	OBJ.	<5	SHIP	002846	SC-56

ATTRIBUTES:

Att	OfObj	Mode	#	Default	Constraint
OWNER	CHART	TEXT	20	SYS	SC-95
NAME	CHART	TEXT	1	NAME	SC-96
SHIP	CHART	TEXT	1	TYPE	SC-97

CHART:

ID	Owner	Name	Type
31507	SYS	ARABIAN SEA	PRESENTATION

INSTANCE_COMPONENTS:

CompID	CompObj	OfID	OfObj
31508	SHIP	31507	CHART
31509	SHIP	31507	CHART
31510	SHIP	31507	CHART
31511	MAP	31507	CHART
31512	GRID	31507	CHART

Figure 11: Relational Representation of CHART Generic and Instance Objects

x-size =The width of this display object in basic MPS units.

y-size =The height of this display object in basic MPS units.

color =The central color value to be used in creating this display icon.

MPS automatically creates a set of DISPLAY tuples for each instance object with default parameter values when the instance object is created. The application system has the obligation to alter any values required to achieve the desired display object.

```
SHIP Layout Template:

        ONTOP(SUPERSTRUCTURE,HULL)
        ONTOP(SUPERSTRUCTURE,FLAG)
        BEHIND(SUPERSTRUCTURE,FLAG)
        BEHIND(HULL,WAKE)
        ATTACH(BOTTOM(HULL),WAKE)
        EQUAL(NATIONALITY,"US") ==>
            COLOR(FLAG,"BLUE")
        EQUAL(NATIONALITY,"USSR") ==>
            COLOR(FLAG,"RED")

CHART Layout Template:

        OVERLAY(GRID,MAP)
        OVERLAY(SHIP,GRID)
        MERGE(DETECTION-PROBABILITY,MAP)
```

Figure 12: A Graphical Layout Template

An example of the definition of a presentation surface called CHART is shown in graphical form in Figure 10. This is a surface which displays a partial map of the world including longitude and latitude lines, and shows the location of various SHIP objects on that map, together with some high-level information about the status of each of those ships. Figure 11 shows the relational tables of MPS corresponding to the CHART object after all of the desired instance objects have been specified by the application system. Note that the generic description of the CHART object has not changed, we have simply attached information about the specific object instances which are to be used in creating the display for the user.

Although the object CHART contains the specification of only objects of type SHIP, all of the generic definitions associated with SHIP objects apply. Thus when the application system stores the description of the Kennedy in the relation AIRCRAFT-CARRIER and specifies that it is to be included in the presentation surface object CHART, the MPS can derive the proper association from its knowledge of the super-object relationship between SHIP and AIRCRAFT-CARRIER.

4.2 DISPLAY OBJECTS

Given the description of an instance of a presentation surface, MPS can be requested to create a display object and show it to the user. This is a two step process. A device independent display object is first created for the presentation surface instance. This is then passed to the presentation surface effector (as described in Section 5) for actual presentation to the user. Creation of the device independent display object is the processes of "mixing the ingredients" mentioned above.

The ingredient mixing is done in accordance with a recipe, one specified by the graphical layout template. MPS uses a simple but powerful language called the High-level Positional Language (HPL) to capture the layout recipe. The details of this language, its representation, and its use in MPS are the subject of a separate paper[12]. Because of the complexity of performing graphical layout using high-level positional specifications, we illustrate here only the general nature of HPL by example.

Figure 12 contains the graphical layout template for the SHIP and CHART objects. Note that the statements in HPL consist of three types: attribute mappings, positional operators, and combinational operators. The attribute mappings specify what changes to the icon templates are to be made to capture the values of the attributes of the instance object. These are used to cause, for example, the selection of the USSR flag when

-----

the NATIONALITY attribute of the SHIP instance is USSR. Positional operators specify where, within the clear surface upon which the display object is constructed, the icon for each component is to be placed. The required knowledge is built into MPS so that it properly handles the intended relationships, such as placing the SUPERSTRUCTURE of the SHIP centered directly above the HULL.

The combinational operators specify the way in which the component display objects are combined to form the desired effect. For example, the SHIP objects are laid on top of the underlying MAP so that they obliterate the MAP portions underneath, while the DETECTION-PROBABILITY icons are combined with the rest of the scene as a transparent overlay, thus allowing the underlying icons to be seen through a "filter".

In our example the use of the super-object relationship between SHIP and AIRCRAFT-CARRIER allowed MPS to properly determine that the graphical layout template to use for construction the icon to represent the "Kennedy" were those associated with the generic object AIRCRAFT-CARRIER. This same type of inference allowed MPS to determine that the actual location of each SHIP object could be computed by mapping the latitude and longitude position of the SHIP into the coordinates of the CHART object.

5. INTERACTING WITH A PRESENTATION SURFACE

Having described how MPS builds an instance of a presentation surface and the associated device independent display object, we can now describe how the display object is shown to a user and how the user may modify the presentation surface. This is all handled by the presentation surface effector, which is responsible for adapting the display objects for viewing and modification on given devices. The key concepts involved are viewing windows and display contexts.

We have used the name "presentation surface effector" to capture the fact that logically it is the presentation surface instance which is being displayed and with which the user can interact. The display object is only representation from which actual icons on a display device can be created. These icons simply represent the objects of the presentation surface in a convenient medium. The objects and their associated attributes are the only things which carry meaning for either the application system or the user. This is the reason we started with the objective of allowing both the users and the application system to converse in terms of objects, and why we have worked so hard to carry this through all portions of the MPS.

MPS places no restrictions on the logical or physical size of a presentation surface, nor on the number of presentation surfaces which can exist at any one time. However, there is a limited size to both the physical devices for displaying graphical images and a limited number of such devices available to the application system. Therefore MPS treats a display device as a "window" which looks out onto the presentation surface, in the same way that earlier systems like SDMS[3,13] use windows. Principal control of the positioning of the viewing window lies with the application system, but control can be given to the user to support panning and zooming. Thus a viewing window in MPS is simply a means by which information from a part of a presentation surface can be visually shared with the user. Like Star[7] and Lisa[10], the MPS window does not necessarily cover the entire screen of the display device. Thus there can be multiple windows displayed on a single device, and each is associated with a presentation surface. Windows can also share presentation surfaces.

Two parameters are needed to determine the relation of the window to its contents: its location and the portion of the presentation surface to be shown. These parameters are established using the WINDOW relation as follows:

WINDOW(surface-id,object-id)

where:

surface-id	=The unique id of the presentation surface instance.
object-id	=The unique id of the instance object associated with that presentation surface.

The tuples of this relation thus specify which instance objects are to be included in the window for a given instance of a presentation surface. MPS is responsible for determining the scaling required to include all of the required object instances. It similarly determines the positioning of the window. The final scaling must of course be determined by the properties of the display device.

User inputs and movement of the window are passed by the presentation surface effector to the application system through the data base. The application system can move the window by changing the tuples of the WINDOW relation. The user can change it through any of a variety of input device (joystick, mouse, tablet, touch surfaces, etc.) supported by the display device. User movements (panning and zooming) are reflected by appropriate changes to the WINDOW relation tuples by MPS.

Any attribute or component of an object which is to be user updatable is given an appropriate "mode" using the mode constraint. When the display object corresponding to the instance object is created, the value of such an attribute or component is displayed as given by the application program, or default value from the generic object description. To display nothing, the appropriate value need only be left unspecified (that is left with a default of "empty"). When the user (either at his own initiative or in response to an application system request) provides an input starting at a given cursor location, the object being changed is identified by the MPS through its mapping of the display object locations on the presentation surface to the window coordinates. The data input is stored in the appropriate argument of the tuple for the object that was changed. When the user selects an instance object, such as a menu item, by moving the cursor, MPS informs the application system of the object which was selected by storing the unique id of the object instance in the appropriate tuple of the data base. Thus user inputs are uniformly reported to the applications system in terms of the objects pointed at or altered.

The representation of the instance object in the data base can also be used to save the responses of the user for later reference, as when the user provides the application program with a number of parameter values to try. Redisplay of old values requires only the appropriate reference to the desired object instance to request its display.

The actual creation and modification of the scene corresponding to the presentation surface instance is handled by the display context mechanism. A display context is a description of the device dependent characteristics of a given display medium, including all of the information required to form a viewable collection of icons from the device independent display objects.

The mechanism to create displays is straightforward. Given an instance of a presentation surface, the application system requests that the presentation surface instance be displayed on a given device. This is done by storing a tuple into the SHOW relation as follows:

SHOW(generic-object,pid,device)

where:

generic-object=The name of the presentation surface of which an instance is to be shown.

pid =The unique identifier of the presentation surface instance (created by MPS if not given).

device =The specific display device context to be used. An error return will result if the specified device is not compatible with the requirements of the generic presentation object.

MPS then uses the device dependent characteristics specified in the device context to map the display objects corresponding to the presentation surface instance into the appropriate icons for the display. For example, if the presentation instance was defined using multiple colors and the device context specified that the device is monochrome, MPS would appropriately create the icons to correspond to a monochrome image.

6. SUMMARY AND CONCLUSIONS

We have described a powerful user interface system which has several unique features. The system utilizes object oriented descriptions of the information to be exchanged between the human user and the application system. This allows a clean demarcation between the functions of the application system and those of the user interface. The MPS interface uses a relational data base as the medium of communication. In addition to the data about specific objects to be displayed, MPS maintains generic descriptions of objects. These are used directly in the verification of the construction of instance objects, in the construction of objects to be displayed, and in the description of the contents of presentation surfaces. Thus MPS is maintaining and using a high level semantic model of the objects to be displayed. This semantic model is stored directly in the relational data base, and is thus available to both the application system and MPS. Because the interactions between the application system and MPS are cleanly defined and use high level object descriptions, relatively low bandwidth communications are needed between the two systems for many types of applications. Therefore MPS can be used in both single processor and multiple distributed processor architectures.

There are two key differences between the object oriented approach employed in the Star[7] and those of MPS. First, Star requires that the application system be implemented within the Star/Mesa environment. While this is a powerful environment, it requires many operating system and programming language facilities not available or implementable on general purpose machines. MPS attempts to relax many of these constraints so that it may be used in a much broader collection of applications. Second, Star does not provide a clean division between the interactions of the interface and the computations of the application system. We believe that the clean definition of the interaction medium greatly strengthens the power of the object oriented approach. It is difficult to tell from the early descriptions of Lisa[10], but it appears that their modifications to the Star object oriented approach lie in the hardware capabilities and not it the areas of interaction between the application system and the interface tools.

Like MPS, the TIMBER[9] system uses a data base to contain all of the information on the screen. An interface to the application programs is provided to allow the specification of the items to be presented by storing them in the data base. However, TIMBER has no built in semantics about the objects to be displayed, and thus forces the application system to work at a fairly low level, and does not provide the nice inheritance features of the object oriented approach. MPS is not simply a browsing tool or an access tool for data bases, such as GUIDE[14]. It is also not just an exercise in knowledge based representations of graphical descriptions, such as APE[16]. Like the FLAIR[15] system, the approach in MPS emphasizes the need for high level tools to aid the development and use of interfaces. The FLAIR effort, however, has concentrated on a language for the specification of interface dialogues, and provides very little built-in knowledge to aid the developer in transforming the concepts used by the application system into appropriate interface activities.

More attention was paid to the interaction between the application system and the interface in the TIGER[5] system work, but TIGER concentrated on a programming language to allow the application system to build dialogs and interact in a device independent way. This still requires the application system to embed all of the user interface activities and code within the application system code, rather than making the clean division used in MPS. The technique of utilizing the data base to maintain the data to be presented and also as the interaction medium with the application system was employed by Foley and

Garrett[2]. Their work dealt only with very low level specifications of the icons to be presented, and the associated problems of correspondence between the icons and the data base tuples. They did not provide any general means for two way interactions.

MPS is currently being implemented and used at AI&DS in several different applications. The implementation of the first version with a subset of the capabilities described above has been completed. More powerful versions will be completed during the next few years. The unique features of MPS allow us to utilize it in a number of fairly diverse applications, including menu-based controls, geographic maps with overlays, radar images with overlays, and interactive AI expert systems using both text and graphics.

7. REFERENCES

1. Deering, M., Faletti, J. and R. Wilensky "Using the PEARL AI Package," Technical Report, Department of EECS, University of California, Berkeley, Berkeley, California, February 1982.

2. Garrett, M.T. and J.D. Foley "Graphics Programming Using a Database System with Dependency Declarations," ACM Transactions on Graphics, vol. 1, no. 2 (April 1982), pp. 109-128.

3. Herot, C.F., "Spatial Management of Data," ACM Transactions on Database Management, Association for Computing Machinery, 1980.

4. Ingalls, D. "The Smalltalk-76 Programming System," Fifth Annual ACM Symposium on Principles of Programming Languages, January 1978, pp. 9-16.

5. Kasik, D.J. "A User Interface Management System," SigGraph-82, Computer Graphics, Association for Computing Machinery, vol. 16, no. 3 (July 1982), pp. 99-106.

6. Kersten, M.L. and A.I. Wasserman, "The Architecture of the PLAIN Data Base Handler," <u>Software</u> -- <u>Practice</u> <u>and</u> <u>Experiences</u>, <u>vol</u>. <u>11</u>, <u>no</u>. <u>2</u> (<u>February</u>, 1981), pp. 175-186.

7. Lipkie, D.E., Evans, S.R., Newlin, J.K., Weissman, R.L. "Star Graphics: An Object-Oriented Implementation," SigGraph-82, Computer Graphics, Association for Computing Machinery, vol. 16, no. 3 (July 1982), pp. 115-124.

8. Rosenthal, D.S.H., Michener, J.C., Pfaff, G., Kessener, R., Sabin, M. "The Detailed Semantics of Graphics Input Devices," SigGraph-82, Computer Graphics, Association for Computing Machiner, vol. 16, no. 3 (July 1982), pp. 33-38.

9. Stonebraker, M. and J. Kalash "TIMBER: A Sophisticated Relation Browser," Proceedings of the Eight International Conference on Very Large Data Bases, Mexico City, September, 1982, pp. 1-10.

10. Williams, G. "The Lisa Computer System," BYTE, BYTE Publications Inc., vol. 8, no. 2 (February 1983), pp. 33-50

11. Wilson, G.A. "A Conceptual Model for Semantic Integrity Checking," Proceedings of the Sixth International Conference on Very Large Data Bases, Montreal, Canada, October, 1980.

12. Wilson, G.A., Domeshek, E.A., Dean, J.S., Drascher, E.L., "HPL -- A High-level Language for Describing Graphical Layout of Objects," Advanced Information & Decision Systems, paper in preparation.

13. Wilson, G.A. and C.F. Herot, "Semantic vs. Graphics -- To Show or Not to Show," Proceedings of the Sixth International Conference on Very Large Data Bases, Montreal, Canada, October, 1980.

14. Wong, K.T. and Ivy Kuo, "GUIDE: Graphical User Interface for Database Exploration," Proceedings of the Eight International Conference on Very Large Data Bases, Mexico City, Mexico, September, 1982,pp. 22-32.

15. Wong, P.C.S. and E.R. Reid "FLAIR -- User Interface Dialog Design Tool," SigGraph-82, Computer Graphics, Association for Computing Machinery, vol. 16, no. 3 (July 1982), pp. 87-97.

16. Zdybel, F., Greenfeld, N., Yonke, M. "Application of Symbolic Processing to Command and Control: An Advanced Information Presentation System," BBN Technical Report 4371, Bolt Beranek and Newman, Inc., April, 1980.