The Cognitive Coprocessor Architecture for Interactive User Interfaces

George G. Robertson, Stuart K. Card, and Jock D. Mackinlay
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract

The graphics capabilities and speed of current hardware systems allow the exploration of 3D and animation in user interfaces, while improving the degree of interaction as well. In order to fully utilize these capabilities, new software architectures must support multiple, asynchronous, interacting agents (the *Multiple Agent Problem*), *and* support smooth interactive animation (the *Animation Problem*). The *Cognitive Coprocessor* is a new user interface architecture designed to solve these two problems, while supporting highly interactive user interfaces that have 2D and 3D animations. This architecture includes *3D Rooms*, a 3D analogy to the Rooms system with Rooms Buttons extended to *Interactive Objects* that deal with 3D, animation, and gestures. This research is being tested in the domain of *Information Visualization*, which uses 2D and 3D animated artifacts to represent the structure of information. A prototype, called the *Information Visualizer*, has been built.

1 Introduction

Rapid improvements in computer and peripheral hardware in recent years have dramatically increased the opportunities for building highly interactive and sophisticated user interfaces for a wide variety of applications. The graphics capabilities and speed of current hardware systems allow the exploration of 3D and animation in user interfaces, while improving the degree of interaction as well. However, in order to fully utilize these capabilities in a systematic way, new software architectures are needed. Two problems, in particular, need to be addressed by these new architectures. First, the degree (and complexity) of interaction increases rapidly as interfaces begin to deal with multiple interacting agents (the *Multiple Agent Problem*). Second, there is a trend toward increased use of interactive animation in interfaces because animation can decrease the cognitive load on the user (the *Animation Problem*). This paper describes a new user interface architecture, called the *Cognitive Coprocessor*, which provides systematic ways of dealing with both the Multiple Agent Problem and the Animation Problem.

After describing the Multiple Agent Problem and Animation Problem in more detail, we describe an application, called *Information Visualization*, which uses 2D and 3D animation to explore information and its structure. We then describe the basis, background, and structure of the Cognitive Coprocessor architecture. Managing collections of visualizations, or *virtual workspaces*, is done with a 3D analogy to the Rooms system [4], called 3D Rooms. Interaction is accomplished through *Interactive Objects*, which are similar to Rooms Buttons, but have been extended to deal with 3D, animation, and gestures. All of these components are being integrated into a prototype system, called the *Information Visualizer*. The paper concludes with a discussion of open implementation issues and the benifits of the architecture.

2 The Multiple Agent Problem

The behavior of an interactive system can be described as the product of the interactions of (at least) three agents [2]: a *user*, a *user discourse machine*, and a task machine or *application* (see Figure 1). The user and the user discourse machine engage in a form of dialogue communication. This dialogue is not, of course, in the form of natural language, but in terms of utterances peculiarly suited to conversation between human

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1989 ACM 0-89791-335-3/89/0011/0010 $1.50

and machine. An interesting feature of such conversations is that, unlike human-human conversations, the utterances are between fundamentally dissimilar agents – a human on one end, and a machine on the other. The machine sends its utterances to the human primarily through graphical presentations (pictures or text) or maybe by sound. The human sends his or her utterances to the machine through manipulations of some sort of input device, such as a keyboard or a mouse. The asymmetries in the conversation reflect (1) the highly developed perceptual abilities of the human compared to the relatively crude perceptual abilities of machines, (2) the ability of many machines to generate (at high speed) graphical information matched to this human perceptual ability, and (3) the greater cognitive resources for semantic processing on the part of humans. In addition to widely dissimilar dialogue languages, these agents operate with very different time constants. For example, a search process in an application and the graphical display of its results may be slow, while the user's perception of displayed results may be quite fast. The user interface must provide a form of impedance matching (dealing with different time constants) between the various agents as well as translate between different languages of interaction.

In addition to the three agents of Figure 1, there are increasing numbers of user interfaces that provide cognitive assistance to the user by adding intelligent agents of various sorts (e.g., an agent to filter and sort your mail [6]). These additional agents have their own time constants and languages of interaction that must be accomodated by the user interface.

Impedance matching can be difficult to accomplish architecturally because all agents involved want rapid interaction with no forced waiting on other agents, and the user wants to be able to rapidly change his or her focus of attention as new information becomes available. For example, if a user initiates a long search that provides intermediate results as they become available, the user should be able to abort or redirect the search at any point (e.g., based on perception of the intermediate results), without waiting for a display or search process to complete. Another example, from the domain of text editors, is the ability of an editor to skip all or part of a display update when sequences of commands are typed ahead. The user interface architecture must provide a systematic way to manage the interactions of multiple asynchronous agents that can interrupt and redirect each other's work.

3 The Animation Problem

Over the last sixty-five years [8], animation has come from a primitive art form to a very complex and effective discipline for communication. Walt Disney has been quoted as saying, "Animation can explain whatever the mind can conceive." In fact, when coupled with computers, animation has been used for a wide range of things, from the concrete of simulated reality (like animating the flight of a simulated airplane), through metaphor (like animating the opening of windows on a computer desktop), to the abstract (like animating algorithms [1,3] and Scientific Visualization). Interactive animation is the most difficult of these techniques, because of its extreme computational requirements.

An important property of interactive animation is that it can shift a user's task from cognitive to perceptual activity, freeing cognitive processing capacity for application tasks. For example, interactive animation supports object constancy. Consider an animation of a complex object that represents some complex relationships. When the user rotates this object, or moves around the object, animation of that motion makes it possible (even easy, since it is at the level of perception) for the user to retain the relationships of what is displayed. Without animation, the display would jump from one configuration to another, and the user would have to spend time (and cognitive effort) reassimilating the new display. By providing object constancy, animation significantly reduces the cognitive load on the user.

The *Animation Problem* arrises when building a system that attempts to provide smooth interactive animation *and* solve the Multiple Agent Problem. The difficulty is that smooth animation requires a fixed rate of guaranteed computational resource, while the highly interactive and redirectable support of multiple asynchronous agents with different time constants has widely varying computational requirements. The user interface architecture must balance and protect these very different computational requirements.

4 Application: Information Visualization

In order to put the Cognitive Coprocessor in a framework that addresses the problems set forth above, we has focused on a class of applications that *require* effective solutions to those problems. The application area is *Information Visualization*, analogous to Scientific Visualization. In Information Visualization, 2D and 3D animated objects (or visualizations) are used to represent both information and the structural relationships

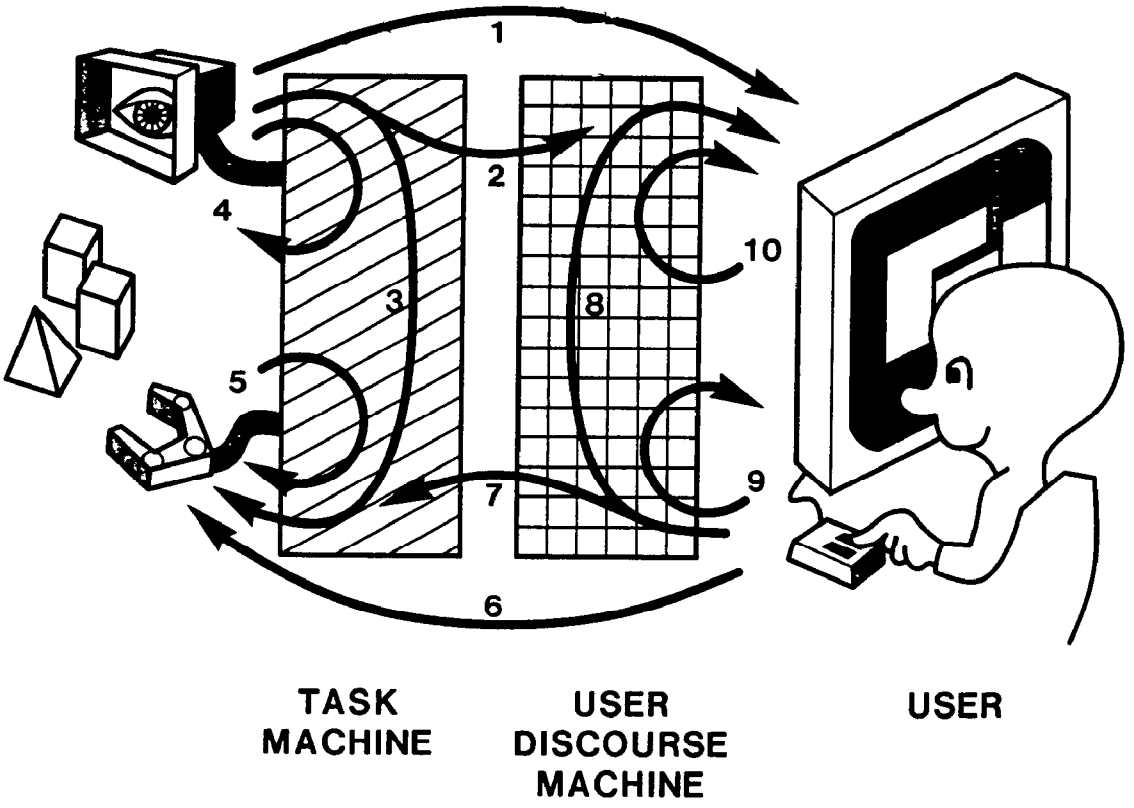

Figure 1: Triple Agent Model of Human-Computer Interaction.

of information. Direct manipulation of these objects causes changes in the actual structure of the information or changes in the actual information. These objects are placed in simulated 3D environments, and the user's task is to move around those environments, interacting with and manipulating the artifacts that he or she finds. Some interactions invoke time consuming operations (like searching a database). Other interactions merely change the user's orientation or some object's orientation in the environment.

Figure 2 shows an example of a simulated 3D environment, called the *Exploratory*, which has a whiteboard on the right wall and a table with some object placed on it. In this simple example, the user's task is to move around the room, interacting with the objects encountered in order to discover their content and function.

Information Visualization can make use of a 3D environment because the aspect ratio and perspective of 3D make much better use of limited screen space when displaying large information structures. It requires a good solution to the Multiple Agent Problem because of the varying time constants and computational requirements of search, manipulation, and movement. And, it requires a good solution to the Animation Problem, because animation is crucial to both movement of the user about the environment and movement of objects in the environment (crucial because the user needs the benefit of object constancy that animation provides in order to make sense out of movement of complex objects representing complex structural relationships).

5 Cognitive Coprocessor

Sheridan [7], in his studies of interactive embedded automation, was the first to arrive at the tri-part division of Figure 1 and to explicate the possible paradigms of interaction. But it seems clear that the three agent model can be adapted and enhanced to serve as a model for human-machine interaction generally [2]. From this point of view, interactions with computer workstations are simply a special case of a more general framework that includes the control of automated airliners, electric power plants, and space probes. Human interfaces to network-based computation, for example, introduce delays, bandwidth limitations, and interactive automation control paradigms previously encountered in embedded automation system control. It has also been pointed out [2] that intelligent agents can be introduced at different sites of the three agent model, creating different classes of intelligent systems.

The Cognitive Coprocessor is a user interface architecture that supports the three agent model, the addi-

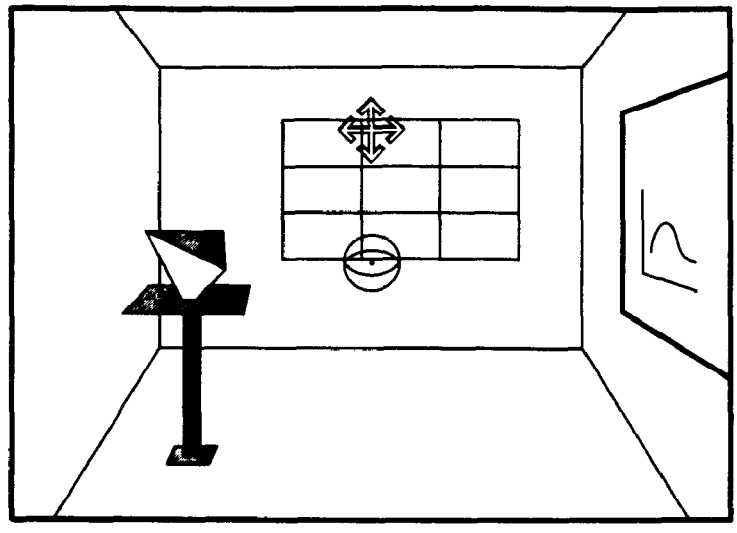

Figure 2: Example 3D-Room: The Exploratory.

tion of intelligent agents, *and* smooth interactive animation. It includes management of multiple asynchronous agents that operate with different time constants and need to interrupt and redirect each other's work. It's name is derived from the cognitive assistance it provides the user and other agents, as well as the coprocessing of multiple interacting agents that it supports.

Figure 3 shows the basic structure of the Cognitive Coprocessor Architecture. The three layers correspond to the agents of the three agent model (Figure 1). The basic control mechanism (inner loop) of the Cognitive Coprocessor is called the *Animation Loop*. It maintains a *Task Queue*, a *Display Queue*, and a *Governor*. Built on top of the Animation Loop is: an information workspace manager (and support for 3D simulated environments), called *3D Rooms*; support for navigating around 3D environments; and support for *Interactive Objects*, which provide basic I/O mechanisms for the user interface. The task machine (which, for the Information Visualization application, is a collection of visualizers) couples with the Cognitive Coprocessor in various ways, as illustrated in Figure 3.

5.1 Animation Loop

The Animation Loop is the basic control mechanism for the Cognitive Coprocessor Architecture, and maintains: (1) a *Task Queue*, which contains pending computational tasks from various agents; (2) a *Display Queue*, which contains pending instructions from various agents about how the screen should be painted on the next animation loop cycle; and (3) a *Governor*, which keeps track of time and allows for adjustments to animations to keep them smooth.

The architecture provides a framework for agents. However, it is clear that agents must be designed carefully to fit in that framework. There are guidelines for defining agents that specify how they should interact with and use the animation loop. For instance, these guidelines indicate the size of tasks and display instructions permitted during any one cycle of the animation loop. Agents must be prepared to be interrupted during a task and redirect or abort their efforts. This can be accomplished by either decomposing tasks into small pieces and allowing one piece to be executed during each animation cycle (like coroutines), or by spawning processes that can be interrupted by interprocess communication (like multiprogrammed systems). In either case, the architecture provides the mechanisms and guidelines to make it easy to construct agents that function well in this environment, but does not force agents to conform to the guidelines.

The *Task Queue* may contain three different kinds of tasks: procedures, light-weight process spawn requests, and heavy-weight process spawn requests. The heavy-weight processes are supported by the underlying operating system (e.g., Unix), share no memory with the rest of the architecture, are communicated with by means of interprocess communication, and may be aborted at any time by the architecture (by killing the process). These are most appropriate for long running tasks or for parts of agents that already exist in some other software package (e.g., a search procedure in a database manager). The light-weight processes are supported by the underlying language (e.g., extended Common Lisp), share memory with the architecture, and are communicated with by means of either shared memory or light-weight interprocess communication. These are most appropriate for medium and long term tasks that are more

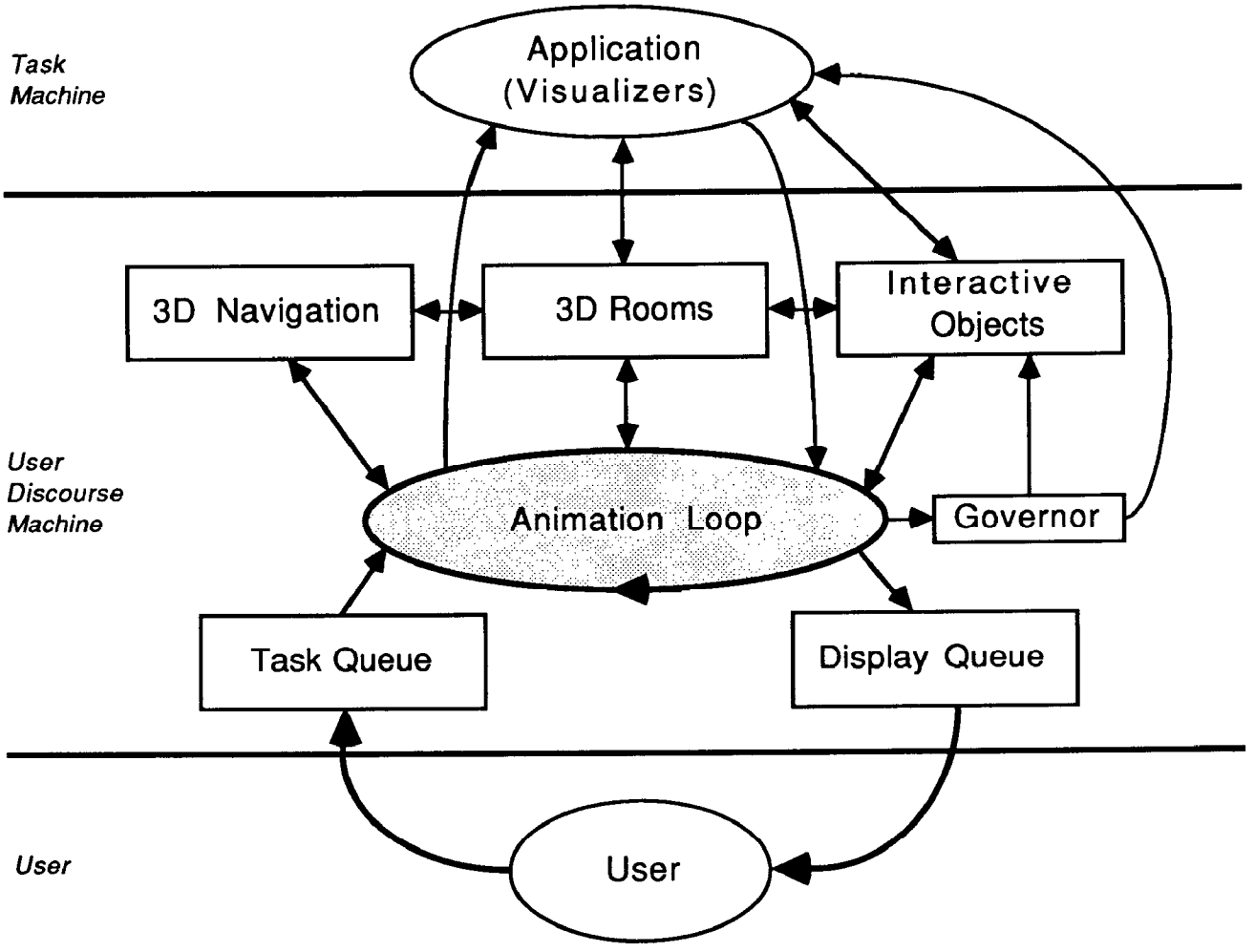

Figure 3: Cognitive Coprocessor Architecture.

fully integrated with the rest of the architecture (e.g., a search procedure over a data structure built by an integrated visualizer). Finally, task procedures are executed to completion, and are expected to be very short. They are most appropriate for monitoring the behavior of spawned processes and for short processing tasks (e.g., checking on the status of a spawned search process).

Conceptually, the *Display Queue* contains procedures for painting the various aspects of the display. In practice, the architecture is object oriented, hence the Display Queue contains all of the objects to be drawn. The architecture is tailored for double-buffered animation (required for smooth animation), hence all objects are redrawn on each animation loop cycle. When single-buffered animation is necessary, the architecture requires application agents to provide a method for drawing only the changes for each object, and to place only changed objects on the Display Queue.

The Cognitive Coprocessor and agents should be designed so that the animation loop runs at ten cycles per second or faster (thirty is ideal), to provide smooth animation. The *Governor* provides a timing mechanism so that agents can tell when the animation loop cycle is taking longer than desirable for smooth animation. Agents are expected to tailor their animations to run at a fixed real-time rate, hence should use the Governor to determine how much distance to cover in an animation for each animation loop cycle. This prevents an animation from running either too fast or too slow. For example, a ball bouncing in a room should be designed to bounce at a fixed rate (e.g., one bounce per second). The distance the ball moves on each animation step depends on how fast the system as a whole is running (i.e., the ball must move further in one step if the system is running slower than normal).

Given the other components of the architecture, the *Animation Loop* is quite simple. It has two major states: the idle state and the animation state. In the idle state, nothing is changing on the display, and the task queue is empty, so the animation loop simply suspends, waiting for input from the user or one of the spawned agent task processes. In the animation state, the following steps are taken:

1. Update the Governor's clock;
2. Process any pending input from the user or spawned task processes;
3. Process any pending tasks on the Task Queue;
4. Draw the new display:
    - Clear the next display buffer (double-buffered animation),
    - Draw each of the objects on the Display Queue,
    - Swap the display buffers;
5. Loop (go to step 1).

This basic architecture with its guidelines for agent construction provides the support we need for managing smooth interactive animation and multiple asynchronous agents.

5.2 Virtual Workspaces: 3D Rooms

Most of the Information Visualization application described earlier can be supported by the Cognitive Coprocessor Architecture described above. However, two additions are needed to fully support the application. First, we need support for simulated 3D environments in which to place our 2D and 3D visualizations. And second, we need a way to manage large collections of visualizations. Both of these are provided by a system we call *3D Rooms*.

The problem of managing large collections of information visualizations is very similar to the problem of managing large numbers of windows on a computer desktop. The Rooms system [4] takes advantage of the fact that a person's dynamic access to information tends to exhibit locality of reference, accessing small clusters of information at a time. Henderson and Card use this fact as the basis for managing windows in clusters, or virtual workspaces called *Rooms*, to significantly reduce space contention in window-based user interfaces. In our Information Visualization application, the virtual workspaces are small clusters of 2D and 3D visualizations.

Since we wish to place our visualizations in 3D environments, it seems appropriate to use a metaphor of a physical room, and make each 3D environment be a three-dimensional simulated room. The extension of the Rooms system to the 3D Rooms system is then natural. The 3D Rooms system manages collections of 3D artifacts (representing abstractions or simulations of information or information structures) instead of windows, and keeps each collection in a simulated 3D room.

5.3 3D Rooms Class Hierarchy

The 3D Rooms system is object oriented. The basic object class hierarchy is shown in Figure 4. There is a basic class, called *Room*, and two major sub-classes, *2D-Room* and *3D-Room*. Sub-classes are created for each widely used collection of visualizations. From each visualization-collection class, individual instances of rooms are created.

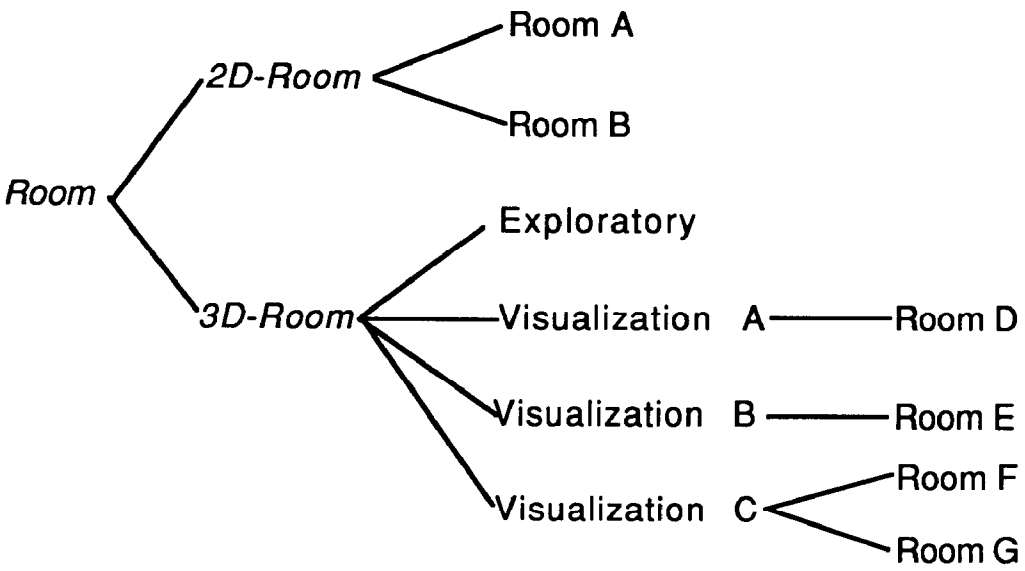

Figure 4: Example 3D Rooms Object Class Hierarchy.

A basic Room has the following characteristics (object slots): a name, a list of interactive objects (described in the next section), and a mapping from keystrokes to selectable interactive objects. A basic Room has methods for:

- Initialization,
- Enter-Room,
- Exit-Room,
- Draw-Room,
- Draw-Contents-Of-Room,
- Draw-Interactive-Objects,
- Handle-Mouse-Selection, and
- Handle-Menu-Selection.

A 2D-Room adds some characteristics: a size (x and y) and a background color. A 3D-Room adds: a size (x, y, and z), colors for four walls, the floor, and the ceiling, a focal angle (for 3D perspective control), and home positions for the body (see navigation discussion below), direction the body is facing, and direction the gaze is facing. Visualization classes add their own characteristics and shadow the basic methods to achieve the desired results.

The Exploratory in Figure 2 is an example of a 3D-Room. In addition to the whiteboard and table with some object on it, there is a door on the back wall (not shown), which leads to another room. The user moves around this room interacting with the objects (the whiteboard, the object on the table, and the door).

5.4 Navigation in 3D Rooms

In order to make interaction with 3D artifacts in 3D Rooms effective, we need a graceful way for the user to move about a simulated 3D room. We have chosen to use the metaphor of walking around a room, and have kept room sizes and egocentric motion controls as close to familiar physical settings and movement as possible. The particular motion controls were partially derived from a theory of input devices [5]. They provide three virtual joysticks with one mouse button. The default joystick controls body movement forward and backward, and body rotation left and right. The other two virtual joysticks are tied to icons on the screen (see Figure 2). One icon represents the direction the body is facing, and its joystick controls body movement up, down, left, and right. Clicking on the body icon will return the body to its normal standing position. The second icon represents the direction the gaze is facing (the orientation of the head), and its joystick controls head movement up and down and head rotation left and right, with natural human limitations (e.g., you cannot turn your head more than about 80 degrees). Clicking on the gaze icon will bring the head to a face forward orientation.

These navigational controls are uniform across the entire Cognitive Coprocessor Architecture. In order to provide proper feedback and smooth animation of egocentric movement in a 3D-Room, the navigation mechanism is integrated into the basic animation loop in the "draw" step (step (4) above).

5.5 Interactive Objects (Generalized Buttons)

The Rooms system [4] has devices, called *Buttons*, which are used for a variety of purposes, such as movement, new interface building blocks, and task assistance. A Button has an appearance (typically, a bitmap) and a selection action (a procedure to execute when the Button is "pressed"). The most typical Button in Rooms is a door – when selected, the user passes from one Room to another. Buttons are abstractions that can be passed from one Room to another, and from one user to another via electronic mail. In our 3D Rooms system, we provide a generalization of Rooms Buttons, which we call *Interactive Objects*. Interactive objects are extended to deal with gestures, animation, 2D or 3D appearance, and an extensible set of types.

An interactive object has an appearance; but has a draw method instead of a bitmap, since it may appear as a 2D or 3D object. The notion of selection is generalized to allow mouse-based gestural input in addition to simple "pressing". Whenever a user gestures at an interactive object, a gesture parser is invoked that interprets mouse movement and classifies it as one of a small set of easily differentiated gestures (e.g., press, rubout, check, and throw left, right, up or down). Once a gesture has been identified, the interactive object dispatches to the appropriate method (i.e., there is a method for each of the gestures). These gesture methods are specified when the interactive object is created. The gesture parser can be easily extended to allow additional gestures and gesture methods, as long as the new gestures are easily differentiated from all other gestures.

Interactive objects can be animated by specifying a home position and a selected position. When the object is selected, it flys to its selected position (which may involve a number of rotations and translations). One example of an animated interactive object is an editable text input object which is at home on the floor of the room, but flys to a vertical orientation when selected so that editing is easier. Another example of an animated interactive object is a door that opens when the doorknob is touched.

There are a number of types of interactive objects. In the current implementation, these include static text, editable text, date entry, number entry, set selection, checkmark, simple button, doors, and thermometers (for feedback and progress indicators). The set of types supported for interactive objects can be easily extended.

6 An Information Visualizer: Discussion

The *Information Visualizer* is a prototype system that incorporates all of the architectural features described for solving the Multiple Agent Problem and Animation Problem, in the context of the Information Visualization application. It provides the integration of: 2D and 3D animated visualizations; 3D Rooms to manage the virtual workspaces and provide a 3D environment; interactive objects to provide generalized extensions to Buttons for user interaction; and the Cognitive Coprocessor Architecture as the basic underlying architecture to support multiple asynchronous agents and smooth interactive animation.

To date, we have contructed six 3D rooms in this system, providing various 2D and 3D representations of information and the structure of information. The Exploratory in Figure 2 is an example of one of those rooms.

The system we have built runs on a hardware base (Silicon Graphics Iris) that supports 3D graphics and double-buffered animation. There are a number of implementation issues surrounding the problem of porting the Information Visualizer to other hardware bases. One major problem has to do with 3D graphics standards. It is desirable to use a standard to ease the porting problems, but undesirable if the standard imposes implementation layers that slow the system down and hender smooth animation (with its real-time constraints). One solution that we are exploring is the use of a portable 3D graphics layer that compiles out intermediate implementation layers. Another major problem has to do with support for both single-buffered and double-buffered animation. As mentioned earlier, we are exploring the required extensions to 3D Rooms draw methods to support single-buffered animation. Whether it can be done effectively enough to support smooth animation is still an open question.

The basic Cognitive Coprocessor Architecture has proven to be very effective in providing a good framework for working on the Information Visualization application. It has been easy to define new visualizations once they were conceived (conceiving new visualizations is quite a difficult task, and is not addressed here). And, it has been easy to achieve the desired kind of responsive redirectable interaction with multiple agents, while also achieving the desired smooth interactive animation.

References

[1] Brown, M.H. (1987). Algorithm animation. MIT Press.

[2] Card, S.K. (1989). Human Factors and Artificial Intelligence. In P.A. Hanccock & M.H. Chignell (Eds.), *Intelligent interfaces: Theory, research and design.* Elsevier Science Publishers B.V. (North-Holland).

[3] Duisberg, R.A. (1988). Animation using temporal constraints: an overview of the animus system. *Human-Computer Interaction, 3,* 275-307, Lawrence Erlbaum.

[4] Henderson, D.A., & Card, S.K. (1986). Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics, 5, 3,* 211-243, July, 1986.

[5] Mackinlay, J.D., Card, S.K., & Robertson, G.G. (in preparation) A semantic analysis and taxonomy of input devices. To appear in *Human-Computer Interaction*, Lawrence Erlbaum.

[6] Malone, T.W., Grant K.R., Turbak, F.A., Brobst, S.A., & Cohen, M.D. (1987). Intelligent information-sharing systems. *Communications ACM, 30, 5,* May 1987, 390-402.

[7] Sheridan, T.B. (1984). Supervisory control of remote manipulators, vehicles and dynamic processes: experiments in command and display aiding. *Advances in Man-Machine System Research, 1,* 49-137, JAI Press.

[8] Thomas, F., & Johnston, O. (1981). Disney animation – the illusion of life. Abbeville Press (New York).