AVIARY – A Generic Virtual Reality Interface for Real Applications

A.J. West*, T.L.J. Howard, R.J. Hubbold, A.D. Murta, D.N. Snowdon, D.A. Butler

Advanced Interfaces Group
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
United Kingdom

Tel: +44 61 275 6251
Fax: +44 61 275 6236

Keywords: *Virtual Reality, VR, generic virtual world, parallelism, computer graphics, transputer, object-oriented techniques, extensible languages, human-computer interfaces, AVIARY*

Abstract

This paper introduces the work of the Advanced Interfaces Group at the University of Manchester, which is applying recent innovations in the field of human-computer interaction to important real-world applications, whose present human-computer interfaces are difficult and unnatural. We begin with an analysis of the problems of existing interfaces, and present an overview of our proposed solution – AVIARY, the generic, hierarchical, extensible virtual world model. We describe a users' conceptual model for AVIARY, implementation strategies for software and hardware, and the application of the model to specific real-world problems.

1 Introduction

Our motivation for the work described in this paper is that we perceive many of the interfaces to today's complex computer applications to be difficult and unnatural to use. With any computer system it is necessary for users to be trained in using the system if they are to become adept at manipulating information via the system's interface. Indeed, a central problem of designing good user interfaces is that humans are very adaptable, and will often learn to cope with poor interfaces. Historically, the power of the computer to assist with complex tasks has made it so valuable a machine, that the issue of whether the interface is good or bad has almost seemed to be of secondary importance.

For straightforward tasks, such as word processing, the past decade has seen a major improvement in user interfaces based on direct manipulation, as exemplified by the popular 'desktop' paradigm. However, systems are becoming ever more complex, and there is evidence that paradigms such as this do not extend readily into these more complicated areas. We offer two particular examples which illustrate this situation.

The first concerns the problems of three-dimensional design. 3D graphics has been researched and used for thirty years – see, for example Tim Johnson's paper of 1963 [1] – and efforts have continued to improve the interfaces for 3D display and direct manipulation [2, 3]. Much of this has been hampered

*Author for correspondence, ajw@cs.man.ac.uk

by the central problem of interacting with a supposedly 3D world through a flat display screen. An exception to this was the pioneering work by Sutherland on head-mounted displays [4], which was technologically too far ahead of its time to find general acceptance. During the same period there have been very big improvements in the computer's capabilities for storing and manipulating precise geometric descriptions of 3D designs, of which solid modelling systems are but one example. It is evident that the user interfaces to these systems have not kept pace with the advances in modelling. User interface toolkits, based on the X windows system, for example, may give the interface a professional and modern look, but are not fundamentally different from very early CAD systems. Thus, there remain significant problems with the interfaces to 3D systems.

The second example is that of information management, a task which is becoming more complicated. The desktop paradigm begins to break down when we need to simultaneously track large amounts of information. One aspect of this is that the cognitive load on the operator increases. For example, consider a real-time application such as financial dealing, in which there is an enormous amount of information to be visualized and comprehended. Here, change may be frequent, and it is vital that important changes are quickly recognised and acknowledged. Limitations of screen size and flat display surfaces are contributory factors to the difficulty of designing interfaces for such complex systems. In the real world however, humans have a remarkable ability to remember where objects are in 3D space – well, most of the time! In everyday life we are surrounded by very complex environments and we rely on our spatial memory and perceptual capabilities to keep track of things. For example, we can reach out and grasp a telephone almost without looking at it because we remember where the phone is positioned. These issues have been explored in spatial data management research [5, 6], and more recently researchers have begun to consider whether it is possible to transfer some of the cognitive load to the perceptual domain [7].

We may conclude, therefore, that humans have 3D perceptual and spatial skills which are largely ignored in today's user interfaces, partly because of technological limitations, and partly because some of the ideas are untested in real world applications. Recent developments in the technologies of Virtual Reality (VR) have led to major interest in exploring these user interface issues.

2 Work on VR at Manchester

Much of the work on VR currently centres on development of the base technology, which is still too crude to find general acceptance for most real world tasks. There is also a widening interest in developing specific applications of VR although many of these are based on fairly specialised environments. Examples include flight simulations and arcade games.

Our own research focuses on developing a general framework for advanced interfaces, known as AVIARY which we shall describe in this paper. We believe that it is essential to study how we can apply our ideas to real problems, because it is only by doing this that we will discover those ideas which work, and those which do not. An important objective of AVIARY is that it should support a broad range of quite different types of VR environment, and to test this we have selected a number of trial applications. We plan to develop our ideas for these in collaboration with companies and other potential users to ensure a correct understanding of the real problems.

The initial applications we are pursuing are air traffic control, computer-aided design, data visualization and medicine, and we now examine each of these very briefly to give a flavour of how VR might be used to improve their interfaces.

2.1 Air traffic control

Traditionally, air traffic controllers use two-dimensional displays which show aircraft positions and directions, with flight codes and heights displayed as text. These displays are effectively enhanced radar pictures which depict a time-varying 3D situation in a 2D projection. While there may be good historical reasons for this approach it is clear that other forms of presentation can be devised. Indeed, it is striking how the current displays illustrate a recurring feature of user interfaces generally – namely their evolutionary development. In this application, the information to be displayed is multidimensional. Aircraft have a position in 3D space, a direction and speed, flight code, and an intended flight path. There is pressure for controllers in the UK to handle an increasing number of flights. If air traffic does grow substantially it is hard to see how this can be done without improvements to methods currently employed.

How might 3D interfaces be developed for this problem? We are studying methods for displaying flight information in 3D. One idea we are pursuing is to show aircraft positions and flight paths using envelopes in 3D space; these are like 3D tubes through which the aircraft fly. The controller can check that the necessary constraints for horizontal and vertical separation of aircraft are satisfied. In practice the computer applies the checks and indicates any potential violations. The controller can re-route an aircraft by directly manipulating its flight 'tube', with the computer again checking for violations. With a full VR environment, one could envisage the controller surrounded by aircraft in three dimensions, and able to directly manipulate flight paths by pushing and pulling at the envelopes. The different control zones have complex shapes in 3D space, so there is considerable scope for investigating alternative display techniques.

In practice, such a scenario is unlikely – a controller needs to work in an environment in which he or she can see and converse with other colleagues, and is unlikely to wear a VR headset. This is likely to be true of many applications, notwithstanding research into methods which allow a user to move around in a model. However, we believe that large screen, full field of view 3D displays, and suitable input devices which support novel interaction techniques have much to offer with problems of this type.

2.2 Computer-aided design

CAD systems for 3D design have been developed for 25 years. Indeed, CAD is becoming pervasive, and is routinely used for designing microchips, product packaging, cars and aircraft, to name but a few areas. Thus far, a few CAD applications of VR have been highlighted, especially Computer-Aided Architectural Design [8, 9]. In one novel application, a Japanese company has developed a system which allows a purchaser of a kitchen design to experience what the design will be like by moving around inside it, opening cupboard doors and so on.

Fun though this may be, the potential for improving interfaces to CAD systems themselves, as opposed to allowing people to experience the end product, is enormous. Many CAD interfaces, such as solid modellers, are rooted in the modelling techniques used in physical construction. Much of the research in such systems has been concerned with rule checking – making sure that what has been constructed is logically correct and does not violate a variety of design rules. Clearly, such issues are of major practical importance for the engineer or designer if these systems are to be useful for real work. CAD interfaces, like many others, have tended to use standard interface tool kits for interaction, and construction techniques with which engineers are familiar, such as 2D projections for representing three dimensions. Because such practical issues are so important, little attention has been paid to other issues concerning the way we design and manipulate objects in three-dimensional space.

There is a fundamental problem here, which is that human beings have very real difficulties in visualizing three-dimensional situations [10]. One of the major potential benefits of VR is that it opens up genuinely new ways to build and interact with designs which are not limited by flat display surfaces. For example, imagine a system in which it is possible to sketch in 3D [11].

As an example of how VR could improve such interfaces we consider chemical plant design, using a piping design program. A number of these systems exist, such as PDMS [12]. Complex networks are assembled from a library of parts, interconnected by varying lengths of piping. With a VR system, one could literally *construct* such a design, using voice input to call down appropriate parts, and plugging them into their correct positions and orientations with a dataglove. Design rules could be applied to each connection, and tolerances, accurate positioning, and lengths of piping could be checked.

2.3 Visualization

Supercomputers are increasingly being used for large scale simulation. Applications include computational fluid dynamics (CFD), finite element analysis, earth sciences, theoretical chemistry, pharmaceutical design, and astrophysics. The numerical models used in these applications are multidimensional and frequently time-varying [13, 14]. Attempting to present such information in two dimensions can be very difficult. 3D rotation is often used to convey geometrical shape, and animation is used to show time-dependent behaviour [15]. With the addition of colour, a variety of symbols and other display techniques, the overall effect becomes too difficult to comprehend, and the result is a cognitive processing overload. Again, with VR we believe that the potential exists to remove part of the cognitive load and replace it with perceptual processing. Within the Department of Computer Science at the University of Manchester we are already studying visualization problems in relation to simulations carried out on parallel computers. We plan to explore the application of VR to some of these problems.

2.4 Medical applications

The concept of virtual surgery has already been suggested as an application of VR. One can envisage the benefits of using a combination of video camera and computer-generated imagery within surgical procedures – for example keyhole surgery. Another major area which we hope to pursue is that of student training. There is a shortage of cadavers for medical students to practice on. Using slides and videos may give students a good appreciation of what a surgical procedure looks like, but cannot convey what it feels like to actually carry it out. Experience from flight and battle simulators has already shown that it is possible to create a very realistic response to emergencies – and it is this degree of 'realism' which it would be especially helpful to capture, without harm to real patients.

Such a simulation can also record what a student actually did – whether correct or incorrect – for subsequent analysis and rehearsal. It would be possible to practise just those parts of a procedure which required improvement – just like rehearsing a passage of music – instead of having to start all over again. A virtual body can be reused (abused?) as often as required in order to perfect a technique, unlike a cadaver which can only be dissected once. Models for these procedures will be captured using CT and NMR scanners.

2.5 Towards a new model for human-computer interaction

Exactly as with graphics systems a few years ago, it is apparent that many applications have common requirements, some of which include the following:

- The ability to structure and name objects and parts.
- The control of display methods, such as wireframe, shaded representations, volume rendering, sound, tactile feedback, and so on.
- Attaching constraints and other, possibly time-varying, application-dependent behaviour to objects.
- Navigation of the viewer around a scene.
- Interaction, in application-specific ways.
- Interaction between multiple users, such as a medical student and teacher in a simulated operation.

Some of these are affected by the practicality of introducing VR into to the workplace. In pilot training, for example, a completely enclosed environment may be acceptable. Equally, for some training tasks in which the user must move around, navigation techniques which permit a reasonable range of movement will prove appropriate [16]. But in other situations, users need to work in a normal office environment in which they can communicate with colleagues, drink coffee, read papers on their desk, and so on. Thus, it will be important to explore input devices and novel interaction techniques which can be employed with more traditional 2D workstations and with large-screen, full field of view stereo displays [17, 18].

It is pointless to support only those features needed for one application area, unless it is a large and very lucrative one. Many existing VR systems are intimately related to particular hardware and software. If we wish to design a system which is capable of supporting a wide range of applications, we require a model which can be implemented on a range of underlying hardware and software. This is especially important when we consider that current VR hardware is in its early stages of development, and in the years to come we can anticipate continuing advances in technology.

In the remainder of this paper we describe such a model. First, we develop a general framework – or in user interface parlance, a users' conceptual model.

3 The generic world model

Having looked at the motivation behind the AVIARY project, we now turn to the development of a model which will satisfy the requirements of advanced human-computer interfaces. We are dealing with a complex model, in that we are concerned with the nature of a *reality*, albeit one which is greatly simplified compared to the reality of the physical world we inhabit. It is therefore important that we begin by having a consistent understanding of the nature of the proposed reality. From this we will go on to examine in more detail the implementation of its various aspects in the software model, and then describe how these are realised on the hardware implementation.

In the following discussion we use the term 'world' to refer to a collection of properties and the laws which operate on them. Here, for example, mass and energy could be attributes, and gravitation or conservation would be laws. 'Objects' are then defined which possess these properties and whose behaviour is constrained by the laws. The term 'virtual environment' refers to the experience presented to the user by the system.

3.1 A conceptual model of reality

If we are to construct a consistent model of a reality, it is essential to develop a coherent philosophy in order to understand the part played by each component of the system, and the nature of their interactions. At this level of exposition, there are essentially four aspects of the reality with which we are concerned. These are as follows:

- The world and its nature.
- Objects which exist in the world.
- Applications which interface to the world through an association with objects in that world.
- Users which interact with applications.

We must define what is meant by each of these, how they interact, and how they should be interpreted in relation to the reality. By so doing, we are defining a conceptual model, which forms the basis of the implementation, and lends coherency to the world. This coherency is important. The powerful perceptual mechanisms of the users that we are trying to engage rely on there being some regularity, or underlying consistency, to their experience. It is important that the users conceptual model we develop has this characteristic if it is to be comprehensible.

The task of developing such an infrastructure is in itself an exciting and challenging problem. Historically, much effort has been expended by philosophers developing systems to explain the nature of the reality we experience in our day-to-day lives. Many systems have been developed, and it might be argued that none has been entirely satisfactory. Some of these ideas are useful when designing the rather specialised (and more limited) realities that are the subject of this paper. At this stage we are not primarily concerned with implementation issues, although of course, that harsher kind of reality must also be borne in mind.

3.2 The requirements of reality

We begin by considering the requirements of reality. The purpose of the AVIARY project is to provide a VR interface to real applications. If our system is to have general application, then the range of features to be supported must be wide. These range from 3D objects with gravity, inertia and collision detection, to notions such as temperature and abstract vector spaces.

It immediately becomes apparent that supporting even a small subset of these requirements 'well', is beyond the scope of current accessible technology. By 'well' here, we note that, for VR, the emphasis must be on maintaining seamless interaction rates, which implies keeping the closed loop feedback times low enough for perception of the world to be convincing. Although we know from our everyday experience that individual interactions may take significant time [7], our overall perception of the world is smooth and immediate. Given this constraint, we conclude that any current VR model cannot simultaneously provide all the facilities desired for the range of applications we wish to support. This is a fact, and is unsurprising. From these considerations we can identify several requirements of the model, which are in some cases conflicting. These are as follows:

- The closed loop feedback time must be small - we must have perceptual continuity if we are to maintain realism.
- A large number of different facilities are required to support the range of applications envisaged.
- The system itself should minimally constrain the nature of the world models that can be generated.
- The system should provide a means of lending consistency to the worlds that are supported.

3.3 A simple model

A straightforward approach is to provide an infrastructure for a basic model of a world, and to require this to be tailored to suit particular requirements. There are two ways in which this tailoring may be done. Firstly, the application writer may extend the VR system's model of the world with the facilities required by the application. Gravitation for example, could be added to the basic world model. While this is useful for limited development, it is not satisfactory as a basis for the general support of advanced interfaces to the range of applications we envisage. If the system's support for applications extends significantly beyond rendering and event handling, then this begins to determine the form of worlds that can be supported. To decide that 'solid' objects are fundamental to a world, for example, precludes some obvious methods for representing vector spaces required in data visualization.

The second way in which a simple VR model may be matched to particular requirements is to extend the application itself to provide the additional functionality. Gravitation could for example be handled by the application taking on the task of computing its effects. If, in this case, the VR system's support extends no further than managing rendering and events, then a lot of work is left for the application to perform. In addition to compounding the problems of code re-use, different applications may present widely differing world models to the user unnecessarily. The coherency of the

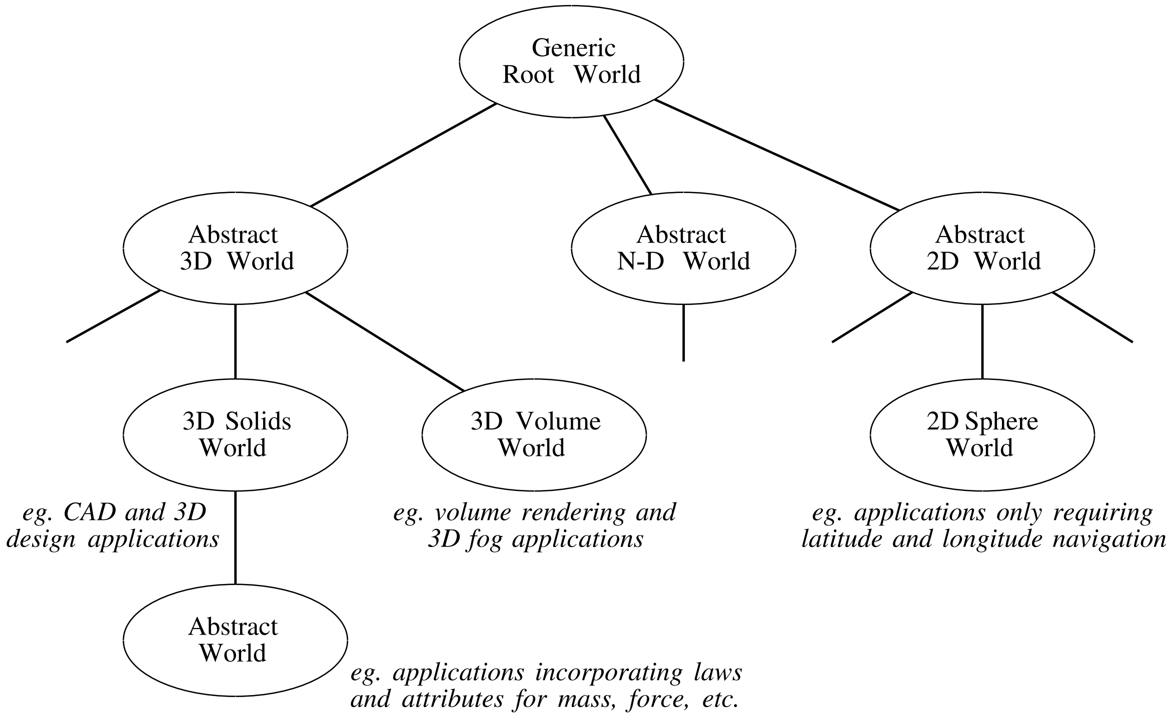

Figure 1: A hierarchy of virtual worlds.

world presented to the user would be left as a problem for the applications interface writers, if it were not in some way intrinsically supported by the system. This appears to conflict with the wish that the system should not constrain the kinds of worlds that can be created.

3.4 Parallel worlds

For the interface requirements of the broad range applications we have been considering, there is much that is in common, but also much that is mutually exclusive, either logically, or for reasons of efficiency as indicated above. The conflicting properties of commonality and diversity may be catered for by postulating that there are many possible worlds, each with its own particular set of laws, and that an application should choose the most appropriate world to map onto.

It is important to consider the relationships which exist between worlds, since it is likely that they cluster into groups with very closely related properties, dependent on the nature of the applications they are intended to support. For example, we would expect to make particular use of worlds with a three-dimensional nature, which contain solid objects that may collide with one another. For some worlds of this kind, supporting mechanical simulations, for example, it would be appropriate to include gravity. For other applications, however, such as data visualization, gravity might not be appropriate.

From a consideration of the kinds of worlds appropriate to particular applications, we are led to propose a model of totality – the set of all possible worlds – which is structured as a hierarchy. This is illustrated in Figure 1. At the top of the hierarchy are the laws and attributes that apply to all possible worlds. The basic support for time, space and causality would be appropriate here [19]. Beneath this we have a number of abstract worlds, which differ according to their fundamental spatial dimensionality. Below these are the broad classes of worlds that cover the most diverse world models (vector space, volume and 3D solid for example). Yet further down the hierarchy are worlds which have successive refinements of these laws. For example some worlds might have gravity, and others may not. Each world may therefore be viewed as a ‘refinement’ of its parent in the hierarchy, which inherits the properties of the parent and augments or amends them as appropriate.

What is the interpretation of the different levels of the hierarchy in our conceptual model of reality?

As there is no constraint on what alterations a child world should make to the properties inherited from its parent, there is no absolute significance to the level of a world in the hierarchy. Furthermore, if we were to treat the highest level as the most abstract, with only the leaf nodes of the hierarchy as fully concrete worlds capable of being inhabited by objects, this would lead to a practical limitation. One can clearly envisage instances where the most appropriate world for an application is not one of the specialised nodes near the leaves, but one of the more general specifications nearer the top. If for example, we wished to construct a world to demonstrate relativistic effects, then a simple world near the top of the hierarchy would provide the most appropriate starting point, as more specialised worlds further down the hierarchy may have too restricted a view of spatial laws to support relativistic effects easily.

3.5 Worlds and objects

A world may contain objects which are governed by the laws of that world. The laws merely express relationships between objects, they have little to say about the objects themselves. An object, on the other hand, is a concrete entity – an instance of something upon which the laws act. Therefore the definition of a world must include a specification of the fundamental (minimum set of) attributes of all objects that may exist within it, and constraints on the values and relationships between these attributes. An object, as an instance, then specifies actual values for these attributes.

In a conventional world for example, with gravity, inertia and collision detection would define mass, extent, location, velocity etc. as fundamental attributes of objects within it. A bouncing ball in this world must therefore possess these attributes. An application associated with the ball may define additional attributes, such as price, age, or the nature of the material from which the ball is made, which define the relationship of the object to the application.

At this point we should stress that we are conscious of the dangers of interpreting this model too literally with respect to the real physical world we inhabit. Clearly, the overhead of continually computing universal laws applying to all objects in a world would be immense. Rather, what is intended is that the system should provide methods for operating and interacting with items in the virtual world, and that operations which are essentially due to the nature of the world model rather than the application, should be automatically enacted and constraint checked by the virtual world system. In this way the system provides the default behaviour of objects and removes much of the burden of world dependent simulation from the applications.

Suppose for example, that a simple 3D world has basic Newtonian mechanics defined, but only perfectly elastic behaviour on collision is supported. This defines the default behaviour of objects in this world. If we wish a particular object such as a bouncing ball to behave in a more complex way, perhaps non-elastically, or with deformation, or a jet motor, then some modification of the default behaviour of this object must be made. This new behaviour need only be invoked in situations where it applies. For a non-elastic model this would be on collision for example. When the new behaviour is not applicable, the object can be left in the hands of the world model.

3.6 An interpretation of applications and users

If objects act merely in passive accord with the laws of the world, then they are rather lifeless, or at least soul-less. We correspond the fact that objects are driven by forces beyond the mechanics of world laws to the notion of volition, or free will. This offers an explanation for the unusually structured or patterned behaviour of the world, that cannot be explained in terms of the world's laws alone.

We may distinguish two kinds of volitional behaviour: external and internal. When it originates from an application or user, then its source is external to the world model. We also permit objects to have behaviour directly, in which case its source is internal. The latter form enables the creation of autonomous world objects; this has practical implications discussed in the software implementation section which follows. Further, the two kinds of volition may be interrelated: the internal behaviour

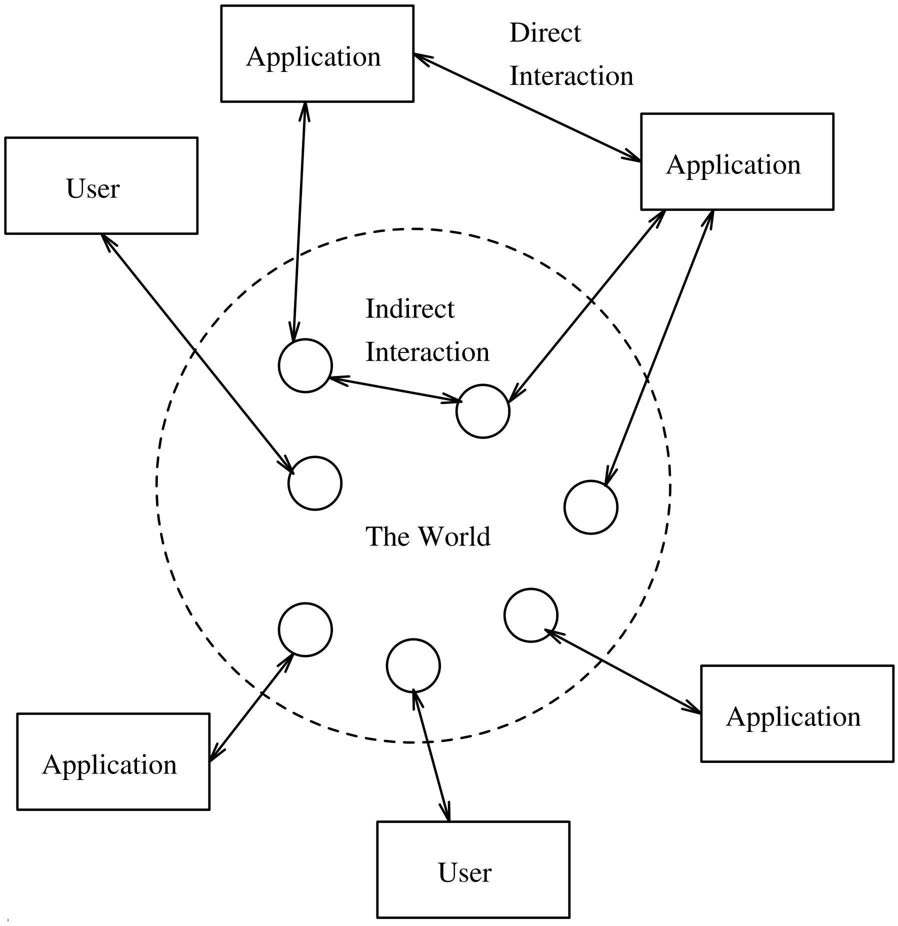

Figure 2: Direct and indirect interactions between applications.

may be dynamically modified by the external sources. This volitional behaviour may appear at the level of individual objects, as in the case of an aeroplane 'icon' showing the position of a real world aeroplane, or it may be a more complex relationship between many objects, as in the representation of a vector space. In all cases however, it is essentially motivated by forces beyond the mundane laws of the world.

The operation of a single application in a world is conceptually straightforward. We are however particularly concerned to allow multiple applications to co-exist within the same world. In this case, if several applications are to interface with the same world model, then clearly their manifestations will be of that world, and must be compatible with it by conforming to its laws. In such a scheme, each application controls its own objects, which, as they affect and are affected by the world, may in turn affect, or be affected by other applications.

To use a trivial example, if two bouncing balls are driven by separate applications, and the balls collide, then one application is affecting another, purely through its representation in the world. Any interactions that take place via the world in this way will be consistent with the operation of the world. Interactions may certainly take place between applications directly without involving the world, in which case they occur beyond the realm of experience. This is illustrated in Figure 2. From the point of view of an inhabitant of the world, there can be no direct knowledge of ultimate reality (the applications), only of an indirect perception of it (the phenomena of the virtual world environment). If actions in the world for example, cause files to be transferred between applications, then the mechanics of the actual transfer are direct communications between the applications involved, and are not experienced in the world unless explicitly manifested by the applications.

3.7 Users

How do users fit into this model? Given we have worlds with objects and applications driving them, what additional features are required to support users? To answer this question we must ask in what way users are special, and in what ways they differ from other aspects of the world.

A user may be an object in the sense of having an objective manifestation either to him/herself in the case of looking at one's body, or to other users if they exist. A user also has the qualities of an application in that they imbue the objects they control with volition beyond the background physical laws of the world. The fact that a user may perceive the world (whether visually, aurally, or with tactile feedback, and so on) and may interact with it, does not fundamentally distinguish a user from an application, which also 'perceives' the world, albeit in a rather different fashion. In the same way that an application needs to be able to interface with the world, a user requires suitable perceptual and effectual apparatus. For a human user the interface can be quite complex, as it involves rendering task and needs a great deal of support. This is nonetheless viewed as merely the interface associated with the user application, and is not intrinsically central to the world model.

There will, of course, be more to be said on the subject of the perceptual apparatus available to the user when we come to address the implementation. At this level of exposition however, we are lead to treat users in the same way as applications.

3.8 Model summary

Our conceptual model comprises a hierarchy of extensible alternate worlds. A world defines a set of attributes, which all things that exist within the world must have, and laws which act on those attributes. Objects are concrete instances of things in the world. They have actual values for their attributes, and obey the laws of the world. Applications interact with the world through the objects, which may be regarded as the manifestation of the application within the world. Users are not considered as being fundamentally distinct from applications. Multiple applications and users are supported. This is the conceptual model to which users relate, and which the implementation supports.

4 Software implementation

Any implementation of the AVIARY model should be fast, without compromising the philosophical model which underpins the system. The implementation therefore requires a large amount of computational power in order to provide the level of support needed in VR application development. Advantage may be taken of the natural parallelism present within the model.

To allow for the use of parallel hardware, the implementation is segmented into loosely connected processes which can execute concurrently. This is illustrated in Figure 3. In order not to place too many restrictions on the nature of any particular kind of parallel hardware, the implementation assumes little about its capabilities. Specifically, there is no requirement for shared memory. The only interactions between processes occur explicitly via a communication system, which given a message will deliver it to a specified process. This strategy should allow the software to map onto a variety of hardware architectures. The components of the implementation are as follows:

- Input processes which monitor external hardware, such as a user's dataglove or a speech recognition system, and produce input for interested processes.

- Output processes which respond to the state of the virtual environment and produce output to some external system. For example, an output process may monitor the state of the world from a given view-point and produce a graphical display on a user's headset, or may provide 3D audio output.

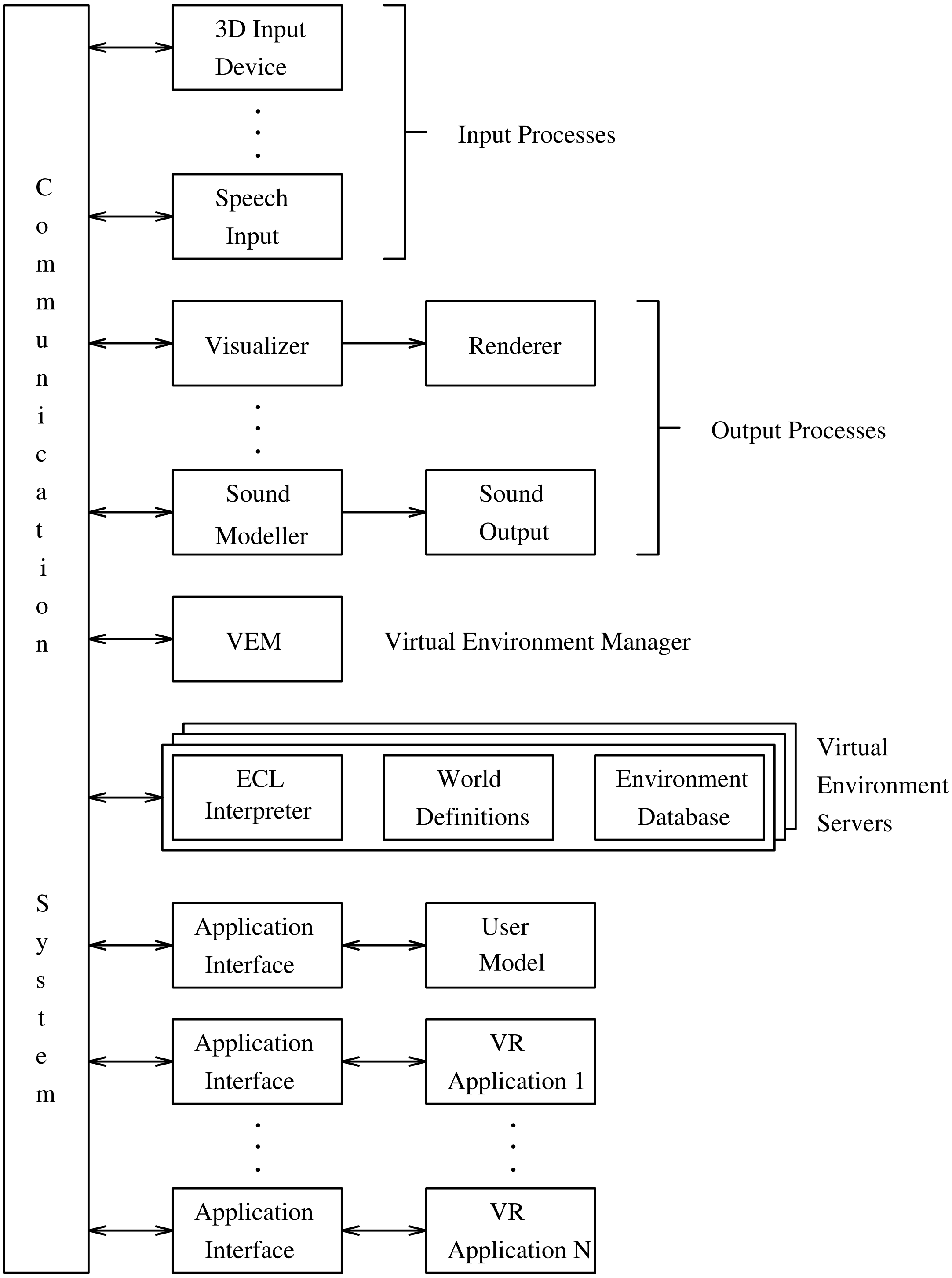

Figure 3: The software architecture.

- A process, the Virtual Environment Manager (VEM), which maintains the consistency of the distributed database of virtual objects.
- One or more server processes which contain an interpreter for a language, called the Environment Control Language (ECL), which is used to describe the virtual worlds and objects, class definitions for the virtual world and a distributed database which contains the objects present in the virtual environment.
- One or more applications which translate user input into actions in the virtual environment.
- One or more applications which use the virtual environment to gather input and to represent their output. A module is supplied which provides functions such as connecting the application to the communication system.

The remainder of this section describes issues affecting applications and users, and the reasons for their explicit separation; how worlds and objects are defined and the means of enforcing the universal laws; the mechanism by which the environment is distributed across loosely connected processors; and the use of an interpreted (or incrementally compiled) language to construct worlds and provide a flexible communications protocol.

4.1 Applications and users

In VR systems which cater only for a single user and a single application, it is possible to have the user's actions modelled by the application. In this case, the application directly reads input from devices attached to the user, such as a dataglove and a headset, and updates its own state and the state of the user, which might be represented, for example, by a displayed hand. This approach is also suitable for supporting multiple users, although it is likely that the processing burden placed on the application would be large. For multiple applications however, this approach is impractical, since applications must now communicate with one another in order to share information about the state of the user.

The approach taken in the AVIARY model is to make an explicit separation between the modelling of the user, and the application. Each user is modelled by a separate process which translates the actions of the user into actions in the virtual environment, and which provides the user with feedback of the state of the virtual environment. Communication between user and application is now indirect, and is achieved by manipulating objects in the virtual environment.

Although users and applications have different reasons for existence, they are no different as far as the rest of the system in concerned and are connected to it by identical interfaces. A module known as the 'Application Interface' provides the necessary facilities to allow an application to connect to the VR system, to access a particular world and manipulate objects in that world. Each application specifies which types of information it wishes to receive from which input processes. The applications receive messages from the input processes when an event in which they have registered interest occurs. Each application is then responsible for its own event processing based on this input and the current state of the world.

4.2 Worlds and their objects

The information that must be stored in order to represent an object in the virtual environment will vary, according to what the object itself is intended to represent. For example, an object which is visible will need to contain graphical information which can be used to render an image of that object. It may also need to store a bounding volume which could be used to help implement collision checking, and it may require additional data to describe its tactile properties, and so on. Clearly the nature of objects also depends on the world in which they exist: if the world models momentum (so that realistic collision effects may be achieved), then the objects will need to have particular masses. Because of these considerations it is not acceptable to simply provide a fixed number of object types and rely on the application writer to store any extra information.

As we saw in Section 3.4, the conceptual model defines a hierarchy of worlds which provide the basic properties and laws governing objects in the worlds. The worlds are implemented as abstract classes – that is, they have no instances. Laws are encoded as program code in the methods of the world classes. For example a method 'move' for a Newtonian world would move the world object after first checking that the new position would not cause the object to occupy the same space as another object. Objects in the world are defined using sub-classes of the world classes. For example, a 'pipe' object might be defined using a sub-class of the 'CAD' class which would thus have a position, and be subject to the laws operating in the world of 3D solids. In addition, the 'pipe' class would define those attributes that describe real pipes such as length, inner and outer radii, material properties, and a graphical description. Creating a new pipe in the world is realised by creating an instance of the 'pipe' class.

The representation of objects can also be designated as 'active' or 'passive'. A passive object contains only data, and is operated on by applications. An active object has code as well as data associated with it, and can execute this code concurrently with the application which created it. The standard object-oriented notion of an 'object' actually falls somewhere between these two categories, in that the object will have code (in the form of methods) associated with it but will execute this code serially and in the same context as the application which invokes the method. With an active object it is possible for the object to reside in a different process from the application. This allows an application to create objects in a virtual environment which will then respond to stimuli in the environment without intervention by the application itself. In this way, the application writer can easily express the parallelism inherent in the environment. For example, in the real world a person is not limited to moving one item at a time, and as it is moved the object may touch several other objects, which may need to respond.

4.3 Distributing the virtual environment

Introducing parallelism into the implementation also creates problems, the most significant of which is the possibility of several concurrent processes requiring simultaneous access to the same data. A common situation might be that the application wishes to manipulate an object, at the same time that a renderer process needs to interrogate the object in order to display it. Storing the contents of the virtual environment in a single centralised database would result in a large amount of inefficiency due to the volume of communication needed between the database and the processes requiring access to the state of the environment. One solution is to allow multiple copies of each object to exist. However, this requires that consistency be maintained between the copies of each object, a situation which has much in common with the cache coherency problems experienced by shared memory multi-processors. To cope with this, the Virtual Environment Manager (VEM) maintains information regarding the allocation of all copies of objects to specific processors. When an object is modified in some way, an appropriate message is sent to the VEM which distributes the update to the relevant processors. In some machines the hardware and operating system manage this coherency, so this becomes trivial.

A further problem caused by the distributed nature of the system is that of synchronisation. For example, several balls are arranged in a line on a flat surface so that each ball is touching its neighbour. An impulse is applied to a ball at the end of the line. An optimistic system might start the first ball moving and transmit a collision event to the next ball in line, and this would then propagate down the line of balls. If the ball at the other end is resting against an immovable obstruction, then an 'undo' message would have to be propagated back down the line restoring the balls to their original state as no movement could occur. A cautious system might propagate a message down the line to see if there was the possibility of movement before setting the balls in motion. In this case the cautious algorithm degrades performance in every case, while the optimistic algorithm will make mistakes and will subsequently need to correct them. The distributed nature of the implementation may make this problem worse, due to the inefficiency of inter-process communication. This type of problem occurs in distributed discrete event simulation, and is particularly difficult to solve for parallel machines. However, approaches to it do exist, such as the 'Time Warp' mechanism [20], and this is a central aspect of our research.

4.4 The Environment Control Language

To get the necessary speed the core functions of the system should be written in an efficient compiled language. We would also like to be able to write the definitions for both the worlds, and the objects within them, in a language which provides support for object-oriented constructs. To ease prototyping it would be convenient if we could alter methods associated with worlds and objects at run-time. Also, we would not want to have to re-compile portions of the core system in order to change the definitions of worlds and their objects.

As stated earlier, the only communication between processes is by messages. Therefore, every time a new feature or class is added to the system, the interface which maps messages to 'procedure calls' must be extended. Also, if objects are to be passed between processors then the receiver needs some method of determining the type of an object at run-time. For these reasons, the implementation makes use of two languages: the 'Implementation Language' (IL), and the 'Environment Control Language' (ECL). The IL is the language which is used to code the VR system itself. The ECL is the language in which virtual worlds and objects are encoded. It is an interpreted (or incrementally compiled) language specially designed to support operations required by worlds and objects in the virtual environment. Since the ECL interpreter provides the interface which translates messages into actions, no explicit interface needs to be written when new classes are added. In fact, fragments of ECL code can actually be used as a message protocol, and this provides great flexibility since we are no longer constrained to having a fixed repertoire of message types. This also provides a mechanism for distributing code updates as a code fragment which redefines one of an object's methods can be sent as an ordinary communication.

Using an interpreted language would seem to sacrifice performance in order to gain flexibility. However, this need not be so. Common functions are coded in the IL and made available as primitive operations. ECL code then serves to 'glue' together these efficient building blocks which implement computationally expensive operations. This is an approach which has been used successfully in several areas. For example, the NeWS windowing system [21] is based on extensions to the interpreted language POSTSCRIPT [22], and systems such as GNU Emacs [23], WINTERP [24] and MusE [25] use versions of LISP combined with efficient primitive operations.

The ECL interpreter with its class hierarchy of worlds and objects and the environment database is packaged into a component of the system called the Virtual Environment Server. This component may be replicated as many times as required, in order to exploit parallelism. Its purpose is to provide a resource for storing the state of the world in the database and for executing the methods associated with each object in the virtual environment.

The section has presented an overview of our proposed implementation of our conceptual model for VR. The implementation is segmented into processes with explicit communication using code fragments of an interpreted language to provide a flexible communications protocol and the state of the system is contained in a distributed spatially oriented database.

5 A hardware platform for the AVIARY model

The main focus of the AVIARY project is to develop a software environment based upon a consistent philosophical model, which will support the use of advanced interaction techniques within application domains. In this regard, the hardware on which the virtual world model is based, although essential in the realisation of our goals, is not regarded as a primary research area. Although we have plans to implement our generic VR environment on a specific hardware architecture, the AVIARY model itself is platform-independent, and may be mapped onto any system with sufficiently high input/output and

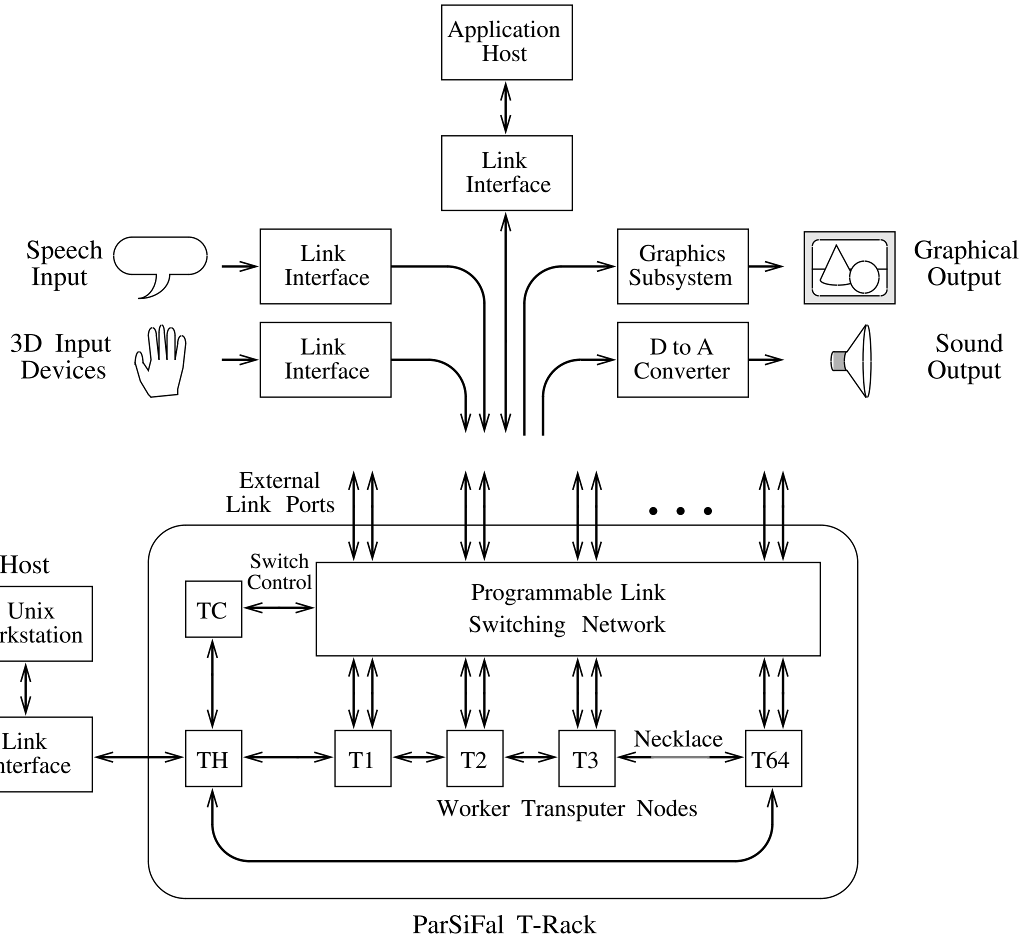

Figure 4: An overview of the hardware architecture.

processing capabilities.

This section describes the computer system on which we initially intend to implement the AVIARY model. We have chosen a high performance, general purpose multicomputer – the ParSiFal 'T-Rack' – as the central processing resource.

5.1 The T-Rack

The T-Rack was designed and built in the Department of Computer Science at the University of Manchester as part of the Alvey sponsored ParSiFal (Parallel Simulation Facility) project [26]. It is a distributed memory machine, based upon the Inmos transputer [27]. The architecture of the T-Rack is shown in Figure 4. The T-Rack contains 64 T800 transputer processors (numbered T1 to T64), on which user applications – in this case the AVIARY software – may be implemented. These are collectively known as the 'worker' transputer nodes. All inter-node communications within the T-Rack are performed via transputer links. Two links from each worker node are used to form a processor chain, known as the 'necklace'. The ends of the necklace are connected to a host interface transputer (TH), through which all code loading, and host-related input/output is performed.

The worker transputer links which do not form part of the necklace are attached to a large programmable switching network. This may be used to install additional connections between worker node pairs. Alternatively, some (or all) of the 128 off-necklace links may be routed through the switch for connection to external devices. The arrangement of communication routes within the switch network is managed by a control transputer (TC).

The Advanced Interfaces Group has a second T-Rack, which may be connected to the first rack via switched transputer links, should additional processing or input/output capabilities be required by the AVIARY implementation itself, or in support of the user applications interfaced to it.

5.2 Concurrency

One of the obvious benefits of using a parallel computer system such as the T-Rack, is the ability to take advantage of the inherently concurrent nature of the virtual world model. Concurrency is exhibited by the model at a number of levels, since there is support for multiple users interacting with multiple applications. Additionally, it can accept input from a number of active sources, and simultaneously drive several output devices. A virtual world may also contain within it many concurrently active objects, which respond to stimuli from external applications, users or other objects.

Consideration of the behavioural requirements of the virtual world model suggests an initial decomposition into two basic activities. The first is internal state maintenance, as performed by the Virtual Environment Manager and Virtual Environment Server processes of Section 4.3. These allow objects to respond to stimuli, while checking that their behaviour is consistent with the universal laws which have been imposed. The second requirement is to interface with the real world. This involves communicating with users and external applications. This decomposition will be reflected in the way the AVIARY model is mapped onto the T-Rack worker processors. Some proportion of the processing nodes will be assigned the task of internal state maintenance (using a distributed state representation), and others will perform interface tasks. The precise number of transputers used for each of these basic activities need not be fixed, and will depend upon the complexity of the world being modelled, or the configuration of peripheral devices which are to be connected.

5.3 Connectivity and consistency

The communication of state information is a central issue in the implementation of the AVIARY model. As well as the inter-processor exchanges required to maintain internal consistency and propagate causality within the virtual world, the state of external peripherals such as 3D locators must be read and acted upon. At the same time, the state of the system as a whole must be presented to output devices so that it may be observed by users. The costs and complexity of communication within distributed systems must not be underestimated.

Within the T-Rack, transputer links provide the sole means of conveying state information between processors. The programmable link switching network of the T-Rack offers considerable flexibility in the installation of communication paths between different parts of the system. Thus networks may be constructed in which peripherals (connected to the T-Rack's external link ports) communicate with the state maintenance nodes via intermediate transputers which perform dedicated device interface activities.

The strategy of connecting peripherals to the T-Rack via the external link ports has two practical benefits. First, it is possible to attach a large number of external devices to the rack. In the current configuration up to 128 link ports may be made available for the attachment of peripherals. Second, the construction of transputer link interfaces which allow the connection of arbitrary hardware subsystems is relatively straightforward.

Maintaining consistency within a distributed state representation can be a problem in real-time applications such as VR. For example, the forwarding of state data across several nodes within a transputer network may result in the information being out of date by the time it reaches its destination. Problems such as this are alleviated by the ability of the T-Rack to dynamically re-arrange link connections at run-time, under the control of the distributed user application. This feature enables state data to be passed directly between arbitrary nodes in the network, wherever synchronism and latency are important issues [28].

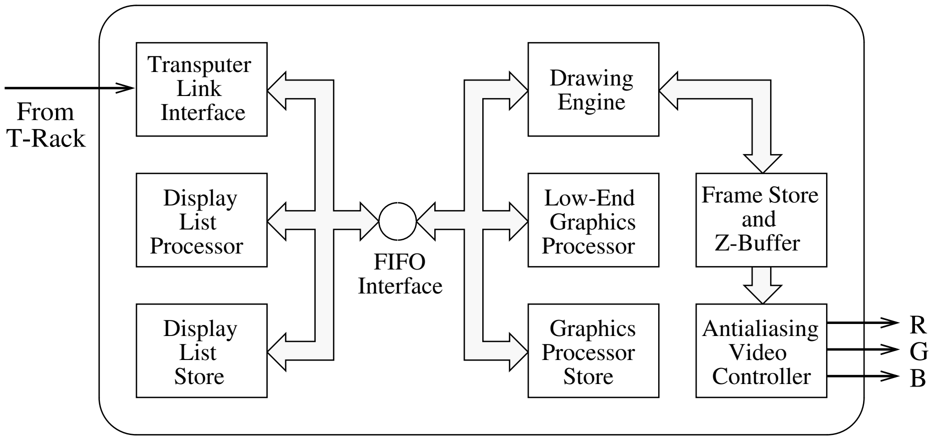

Figure 5: The AVIARY graphics subsystem.

5.4 Peripherals

Our virtual world model defines three fundamental types of peripheral: input devices (accepting data from users), output devices (supplying data to users) and external processing resources (supporting interaction with external applications).

Input devices will typically include spatial locators, such as datagloves [29] and other 3D sensors [30, 31, 32]. Speech input provides another useful means of interaction [33]. In this realm, we hope to draw upon our previous research experience in using speech-based control within graphical environments [34].

Output to users may be presented through a variety of different media. For example, graphical representations of the virtual world may be generated for output to head-mounted, or other stereoscopic displays. Although individual transputer links do not have sufficient bandwidth to convey full screen pixel data at real-time rates, several such links may be used in parallel to write into a shared frame buffer [35]. An alternative solution is to use an off-board rendering subsystem, to which high-level graphical primitives are passed. We have opted for this second strategy, because it reduces the number of external link ports needed to support graphical output, and allows special purpose hardware to be used for low-level rendering operations. This approach is similar to that of Division's ProVision system [36].

A custom graphics subsystem for the T-Rack is currently under development; its structure is illustrated in Figure 5. The hardware, as shown, provides stereoscopic output for a single user, and must therefore be duplicated if multiple users are to be supported. A display list is used to drive the rendering activities. It contains three types of information: geometrical object descriptions (each defined within a local coordinate system), transformations which specify the position and orientation of the objects in the virtual world, and a pair of user viewing transformations (one for each eye). The display list is updated by the T-Rack as the user navigates within the world, and as objects change position. Thus, for the majority of interactions, the traffic between the T-Rack and the graphics subsystem is limited to relatively low bandwidth transformation editing operations.

The display list processor traverses the display list, applying the required transformations to the object descriptions. The transformed primitives are then passed on to a secondary low-end graphics processor which prepares them for scan conversion by a custom drawing engine. Separate eye views may then be extracted from the frame buffer for display on a stereo viewing device.

As well as graphical output, we also consider the use of sound within virtual environments to be essential. It may be used, for example, to relate directional cues corresponding to virtual world events, or to provide audio feedback in response to user interactions. The T-Rack is equipped with a high quality digital to analogue converter which accepts data from an outgoing transputer link, and generates a stereo line signal. This facility was used in a recent research project which modelled the acoustic properties of 3D environments in real time [37]; such modelling techniques should be useful within a VR context.

If the AVIARY system is to provide VR style interfaces for user applications, the machines on which these applications are hosted must be attached to the T-Rack in some way. Several such hosts may be connected to the T-Rack using the external link ports. Alternatively, the T-Rack host may itself be used to run external application programs, with rack access via the host interface transputer (TH, in Figure 4).

6 Summary

In this paper we have highlighted the ways in which some applications are constrained by the inadequacies of interfaces between human beings and computer systems. We have presented AVIARY, a novel model for human-computer interactions, which is underpinned by a coherent philosophy. Our approach provides a generic extensible environment, onto which a range of applications, both existing and anticipated, can be mapped. We intend to evaluate the design by implementing AVIARY on a parallel architecture and applying it to the applications discussed in the paper.

Acknowledgements

It is a pleasure to be able to thank our colleagues, who have generously shared with us their insights and enthusiasm. We are grateful to Jim Garside, Alan Knowles and Pete Jinks of the Department of Computer Science, and to John Churcher of the Department of Psychology.

Author details

Dr Adrian West is a Lecturer in Computer Science at the University of Manchester. He has worked in the automated test equipment industry, and on parallel implementations of secure networks. He has published and presented work on the monitoring of parallel systems and the development of high level graphical environments for parallel programming. His current research interests are in programming environments for parallel systems, and the application of VR.

Toby Howard is a Lecturer in Computer Science at the University of Manchester. He has published widely in the field of graphics standards, particularly PHIGS, and his research interests include VR, high-level programming interfaces to graphics systems, electronic typography and publishing, and computer art.

Dr Roger J. Hubbold is a Senior Lecturer in Computer Science at the University of Manchester, and Associate Director, responsible for Visualization, in the Centre for Novel Computing. His primary research interest is software architectures for interactive visualization using parallel computing systems.

Dr Alan Murta is a Lecturer in Computer Science at the University of Manchester. His recent activities have included research into novel connection and communication strategies for transputer based systems. His current interests include the exploitation of parallelism in real-time graphical and acoustic modelling.

Dave Snowdon is a Research Student in the Department of Computer Science at the University of Manchester. In 1990 he obtained a 1st Class Honours degree in Computer Engineering from the University of Manchester, and in 1991 he obtained an M.Sc. following research into an extensible object-oriented environment for the representation and manipulation of music. He is now working on a Ph.D. in VR.

Alex Butler is a Research Associate with the Centre for Novel Computing at the University of Manchester. In 1987 he received a 1st Class Honours degree in Computer Engineering from the University of Manchester, and in 1989 he obtained an M.Sc. by research, studying optical computing and holographic optical interconnects. He is currently completing a Ph.D. thesis on the subject of 'Visualization and Parallelism in Computer Graphics'.

References

[1] Timothy E. Johnson. Sketchpad III: A computer program for drawing in three dimensions. In *Proceedings Spring Joint Computer Conference*, Montvale, New Jersey, USA, 1963. AFIPS Press.

[2] R.J. Hubbold. TDD – an interactive program for three-dimensional drawing with graphical display and lightpen. In *Proceedings International Symposium Computer Graphics '70*. Brunel University, April 1970.

[3] J. Encarnação and W. Giloi. PRADIS – an advanced programming system for 3D display. In *Proceedings Spring Joint Computer Conference*, pages 985–998, Montvale, New Jersey, USA, 1972. AFIPS Press.

[4] I.E. Sutherland. A head-mounted three-dimensional display. In *Proceedings Spring Joint Computer Conference*, pages 757–764, Washington DC, USA, 1968. Thompson Books.

[5] A. Lippman. Movie maps: an application of the optical videodisc to computer graphics. *ACM Computer Graphics*, 14(3):39–42, July 1980.

[6] A. Lippman. And seeing through your hand. In *Proceedings SID 22(2) Computer Graphics*, pages 103–113, 1981.

[7] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: Animated 3D visualizations of hierarchical information. *Communications of the ACM*, 34(2):189–194, February 1991.

[8] John M. Airey, John H. Rohlf, and Frederick P. Brooks Jr. Towards image realism with interactive update rates in complex virtual building environments. *ACM Computer Graphics*, 24(2):41–50, March 1990.

[9] James H. Clark. Designing surfaces in 3D. *Communications of the ACM*, 19(8):454–460, August 1976.

[10] Robert Parslow. Why is everyone 3D blind? In *EUROGRAPHICS '91 Proceedings*, pages 367–370, Amsterdam, August 1991. North Holland.

[11] Christopher Schmandt. Spatial input/display correspondence in a stereoscopic graphic work station. *ACM Computer Graphics*, 17(3):253–261, July 1983.

[12] CADCentre Ltd, High Cross, Madingley Road, Cambridge CB3 0HB. *PDMS, Piping Design Management System*.

[13] K.W. Brodlie, L. Carpenter, R.A. Earnshaw, J.R. Gallop, R.J. Hubbold, A.M. Mumford, C.D. Osland, and P. Quarendon, editors. *Scientific Visualization: techniques and applications*. Springer-Verlag, 1992.

[14] Special issue on visualization, May 1991. IEEE Computer Graphics and Applications 11(3).

[15] Steven Feiner and Clifford Beshers. Visualizing n-dimensional virtual worlds with n-vision. *ACM Computer Graphics*, 24(2):37–38, March 1990.

[16] Jih-fang Wang, Vernon Chi, and Henry Fuchs. A real-time optical 3D tracker for head-mounted display systems. *ACM Computer Graphics*, 24(2):205–215, March 1990.

[17] Colin Ware and Steven Osbourne. Exploration and virtual camera control in virtual three dimensional environments. *ACM Computer Graphics*, 24(2):175–183, March 1990.

[18] D. Stoops E. Sachs, A. Roberts. 3-Draw: A tool for designing 3D shapes. *IEEE Computer Graphics and Applications*, pages 18–26, November 1991.

[19] Immanual Kant (Translation by J.M.D. Meiklejohn). *Critique of Pure Reason*. Everyman series. Charles E. Tuttle & Co., Rutland, Vermont, USA, 1934.

[20] David Jefferson and Henry Sowizral. Fast concurrent simulation using the time warp mechanism. In *Proceedings of Distributed Simulation '85*, pages 63–69, 1985.

[21] Sun Microsystems, Mountain View, California, USA. *NeWS Preliminary Technical Overview*, October 1986.

[22] Adobe Systems. POSTSCRIPT *Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, USA, 1985.

[23] Richard M. Stallman. EMACS: The extensible, customizable self-documenting display editor. *SIGPLAN Notices*, 16(6):147–156, June 1981.

[24] Niels P. Mayer. *WINTERP – The OSF/MOTIF Widget Interpreter*. Hewlett-Packard Laboratories, March 1991. Included in the 'contrib' section of the X11 distribution.

[25] David N. Snowdon. MusE: An environment for music. Master's thesis, Department of Computer Science, University of Manchester, 1991.

[26] P.C. Capon, J.R. Gurd, and A.E. Knowles. ParSiFal: a parallel simulation facility. *IEE Colloquium Digest 1986/91*, pages 2/1–2/3, May 1986.

[27] Inmos Limited. *Transputer Reference Manual*. Prentice Hall International, 1988. Number 72 TRN 006 04.

[28] Alan D. Murta. *Support for Transputer Based Program Development via Run-Time Link Reconfiguration*. PhD thesis, Department of Computer Science, University of Manchester, October 1991.

[29] James D. Foley. Interfaces for advanced computing. *Scientific American*, 257(4):82–90, October 1987.

[30] Colin Ware and Danny R. Jessome. Using the bat: A six-dimensional mouse for object placement. *IEEE Computer Graphics and Applications*, pages 65–70, November 1988.

[31] M. J. Prime. Human factors assessment of input devices for EWS. Technical Report RAL-91-033, Rutherford Appleton Laboratory, Didcot, April 1991.

[32] Polhemus. 3 space isotrak. Polhemus, Colchester, Vermont, USA, 1991.

[33] Richard A. Bolt. Put-that-there: Voice and gesture at the graphics interface. *ACM Computer Graphics*, 14(3):262–270, July 1980.

[34] Chi-Kam Yeung. Speakeasy. Project report, Department of Computer Science, University of Manchester, 1989.

[35] H.C. Cheung. A high performance graphics system for transputer arrays. Master's thesis, Department of Computer Science, University of Manchester, 1989.

[36] Dick Pountain. ProVision: The packaging of Virtual Reality. *Byte*, pages 53–64, October 1991.

[37] Graham Binns. A virtual sound environment using geometrical ray tracing. Project report, Department of Computer Science, University of Manchester, July 1991.