# VEOS DESIGN GOALS

William Bricken
August 1992



As a technology matures, the demands on the performance of key components increase. In the case of computer technology, we have passed through massive mainframes to personal computers to powerful personal workstations. A growth in complexity of software tasks has accompanied the growth of hardware capabilities. We have gone from command lines to windows to life-like simulation. Virtual reality applications present the most difficult software performance expectations to date. VR challenges us to synthesize and integrate our knowledge of sensor fusion, databases, modeling, communications, interface, interactivity, and autonomy, and to do it in real-time.

The Virtual Environment Operating System (VEOS) is currently under development at the Human Interface Technology Lab at the University of Washington. The VEOS project is the responsibility of Dr. William Bricken and a team of several graduate students lead by Geoffrey Coco. VEOS is designed to integrate the diverse components of a virtual environment.

VEOS consists of several software subsystems. The kernel manages processes, memory, and communication. The entity interface permits modeling objects in the environment, and the environment itself, in a consistent, object-oriented manner. The interaction tools empower a participant within the virtual environment.

The design of VEOS reflects multiple objectives, many practical constraints, and some compromises. Most importantly, VEOS is a research prototype, intended to show the way rather than to be sold for profit. VEOS is supported by a consortium of industrial partners and is developed in the high turnover environment of a university research lab. Thus it is constantly undergoing revision and iterative refinement.

As a research vehicle, VEOS emphasizes functionality at the expense of performance. Premature optimization is the most common source of difficulty in software research. We believe that code is best improved after it is functioning. So our efforts are directed toward demonstrating that a thing can be done at all rather than demonstrating how well we can do it. Since a research prototype must prepare us for the future, VEOS is designed to be as generic as possible; it places very little mechanism in the way of exploring diverse and unexpected design options.

A central consideration for the VEOS project was to respect and integrate with existing and about-to-be-released commercial software. As a research organization, we sought to avoid duplicating what could be bought. VEOS looks five to ten years ahead. It sets an expectation that can be met by the commercial software marketplace through diligent refinement of basic principles.

However, a VR system is far too ambitious an undertaking to begin from scratch. We have conceptualized VEOS as primarily a synthesis of known and understood techniques. Since VEOS is designed for technology transfer, we engage in very little basic research. Instead, we assemble disparate software ideas into a tightly integrated system with new functionality.

| |
|---|
| Research prototype, 5-10 years ahead of marketplace |
| Functionality rather than efficiency |
| Incorporate commercially available software |
| Synthesis of known software technologies |
| Rapidly reconfigurable |

**TABLE I: VEOS Practical Design Decisions**

VR is characterized by a rapid generation of applications ideas; it is the potential of VR that people find exciting. However, complex VR systems take too much time to reconfigure. VEOS was designed for rapid prototyping. The VEOS interface is interactive, so that a programmer can enter a new command or world state at the terminal, and on the next frame update, the virtual world display will change. Any real-time interactive system requires immediate accessibility. VR systems must avoid hardwired configurations, because a participant in the virtual world is free to engage in almost any behavior. For this reason, VEOS is reactive, it permits the world to respond immediately to the participant (and the programmer).

The broad-bandwidth display and the multisensory interaction of VR systems create severe demands for sensor integration. Visual, audio, tactile, and kinesthetic display requires the VR database to handle multiple data formats and massive data transactions. Position sensors, voice recognition, and high dimensional input devices such as the Spaceball, the Flying Mouse, and the Wand overload traditional serial input ports. A VR i/o architecture must incorporate asynchronous communication between dedicated device processors in a distributed computational environment. The database must accommodate update from multiple processors. In VEOS, we have adopted a Linda-like communication model which cleanly partitions communication between processes from the computational threads within a process.

The characteristics of the virtual world impose several design considerations and performance requirements on a VR system.

| |
|---|
| Coordination between distributed, heterogeneous resources |
| Interactive rapid prototyping and reconfiguration |
| Entity-based modeling |
| Multiple participants |
| Concurrent divergent worlds |

**TABLE II: VEOS Functionality**

The design of the virtual world could readily overwhelm a programmer if the programmer were responsible for all objects and interactions. Virtual worlds are simply too complex for monolithic programming. Entities within the virtual world must be modular and self-contained. The designer should be able to conceive of an entity, say a cow, independently of all other aspects of the world. VEOS is structured so that each entity is designed to be independent and autonomous. The system itself takes care of the lower level details of inter-entity communication, coordination, and data management.

In VEOS, all entities are organizationally identical. Only their structure, or internal detail, differs. This means that a designer needs only one metaphor, the entity, for developing all aspects of the world. Changing the graphical image, or the behavioral rules, or even the attached sensors, is a modular activity.

Entity modularity is particularly important when one recognizes that hardware sensors, displays, and computational resources are themselves first class entities. The entity model provides integration modularity for any new components to the VR system, whether they are graphical images, added cpus, or new input devices. Entities can be run independently, as worlds in themselves, or they can be combined into complex worlds. This means that devices and models can be tested and debugged modularly.

Because entities consist of both data and operating system processes, an entity can use other software modules available within the larger operating system. An entity could, for instance, initiate and call a statistical analysis package to analyze the content of its memory for recurrent patterns. The capability of entities to link to other systems software make VEOS particularly appealing as a software testing and integration environment.

Entity autonomy is best modeled by assigning a separate processor to each entity. This approach makes VEOS essentially a distributed operating system.

Distributed resources arise naturally in VR, since the virtual environment is a social place, accommodating multiple concurrent participants.

When more than one person inhabits a virtual world, the perspective of each participant is different. This can be reflected by different views on the same graphical database. But in the virtual world, divergent perspectives can be embodied in divergent databases as well as divergent viewpoints. That each participant can occupy a unique, personalized world makes VR essentially different than physical reality.

Software development is often rocky. In a graduate school environment where the programming staff is part-time and transient, a software project must exhibit both modularity and redundancy. The MOSES (Meta-Operating System and Entity Shell) project is a self-contained and independent duplicate of VEOS. The MOSES team has focused on detailed specification of system functionality, scalability of the conceptual design, fault tolerant shared memory, and heterogeneous communication. MOSES also brings hybrid vigor to the VEOS project.

In summary, VEOS is a significant effort to provide transparent low-level database, process, and communications management for arbitrary sensor suites, software resources, and virtual world designs. VEOS is the glue under VR. As such, it provides a strong integration environment for any team wishing to construct, experiment with, and extend VR systems.