7. International Immersive Projection Technologies Workshop
9. Eurographics Workshop on Virtual Environments
J. Deisinger, A. Kunz (Editors)

# An Innovative Design Approach to Build Virtual Environment Systems

M. Oliveira[1], J. Crowcroft[2] and M. Slater[1]

1 Department of Computer Science, University College London, London WC1E BT, UK
2 Computer Laboratory, Cambridge University, Cambridge CB3 OFD, UK

**Abstract**
*A Virtual Environment (VE) presents a complex problem with interesting non-trivial challenges for software development. The majority of existing systems supporting VE are based on monolithic architectures, making maintenance and software reuse difficult at best. When a novel concept or idea requires implementation, it is not possible to extend an existing system by replacing or incrementing the necessary functionality. This leads to a proliferation of VE systems.*

*This paper identifies some of the major problems in the current development trend of VE systems that result in incremental innovation with little overall progress. However, component methodology and other software engineering principles are not widely employed in system design. We present the Java Adaptive Dynamic Environment (JADE) as an innovative design approach to building VE systems.*

*The paper discusses some of the major elements of the JADE component framework, such as the kernel, the namespace, the event model and how to configuration takes place. In addition, a simple maze dungeon game is discussed demonstrating the runtime reconfiguration of the supporting VE system.*

**Keywords:**
*Virtual reality, virtual environments, system design, component frameworks, online games, java*

Categories and Subject Descriptors (according to ACM CSS): D.2.11 [Software Architectures]: Domain Specific Architectures; I.3.7 [Computer Graphics]: Virtual Reality

## 1. Introduction

An exact definition of a Virtual Environment (VE)[*] is di f-ficult to find. However, all VEs share a common vision, which is best described in the works of science fiction literature related to cyberspace, such as Neuromancer [Gibson84] and Snowcrash [Stephenson92]. The achievement of developing the necessary infrastructure to support the vision remains an elusive goal. This is partly due to the fact that innovation in system development has stagnated or is done in barely noticeable increments.

A VE presents a challenging problem with regards to the development of an underlying system. The problem domain presents itself being vast, requiring diverse areas of expertise, which may range from networks to psychology. The daunting complexity makes the development of any existing VE system a difficult task to achieve with a high cost in resources (money, time and people).

The wide applicability of VE, such as scientific visualization, socializing, training, psychological therapy, gaming, produce a set of requirements that make it very difficult or nearly unfeasible to build a single system to fit all needs. Traditionally, the result has been the creation of monolithic systems that are highly optimized to a particular application, without possibility of reusability with a different purpose.

The development trend of the VE community is seriously affected by the "reinventing the wheel,, and "not invented here,, syndromes. These philosophies limit inno-

[*] We consider an online game to be an instance of a virtual environment

vation and impedes the field to mature sufficiently to begin discussion of standards necessary for empowering the web with 3D windows into alternate realities. A brief analysis of the main causes is presented in section 2, along with some of the pioneering work in new directions. We will describe in section 3 the Java Adaptive Dynamic Environment (JADE), an innovative approach to design and build VE systems. Also included in section 3 is the description of a simple VE system evaluating the dynamic capabilities of JADE. Finally some concluding remarks are summarized in section 4.

## 2. Previous Work

The current trend in the VE community has been for a new VE system to be developed every time it was necessary to have one. Another reason that motivates the genesis of a new system is the lack of system flexibility to a particular application. This has lead to a wide proliferation of monolithic systems, such as DIVE [Hagsand96], MASSIVE [Greenhalgh00], NPSNET [Macedonia94], and SPLINE [Anderson95]. Even with the introduction of some modularity, which promoted the emergence of toolkits such as the WorldToolkit [Sense98] and Avocado [Tramberend98], the core problems that plague the current systems continue to persist.

The trend in the games industry is even worse, where the production cycles of each game traditionally involve game design, technology development and content creation. The licensing of game technology is not overly successful since the particular requirements of each game usually involves significant changes in the code, leading in some cases to total redevelopment of the supporting game engine.

### 2.1. Problems

The problems, which constraint innovation, are summarily described as follows:

- **Non-Extensibility**. The design of most VE systems is tightly coupled with the initial requirements, thus resulting in monolithic architectures where any changes or modifications are unfeasible. The architectures of more recent versions of some systems do present a modular design, but continue to make it difficult to extend the core functionality if it was not foreseen in the original design. Thus, any changes require significant, if not total, reengineering of the underlying system. In most cases, the most cost/effective solution is the creation of a new system.
- **Non-Interoperability**. It is not possible to migrate necessary functionality between different systems to obtain the most effective solution customized to a specific problem with minimal resource expenditure. So despite the existence of some toolkits, code migration remains an illusory goal and consequently fomenting the creation of new systems.
- **Non-Evolution**. The current existing VE systems do not evolve at runtime. Every time a modification occurs, no matter how small, it is necessary to shutdown the system to replace the codebase and reinitialize it. This problem does not have a dramatic impact on existing VE systems used within research laboratories since their operation is session based, meaning system availability is limited for small periods of time in well controlled environments. The same does not occur with commercial systems that are required constant availability around the clock.
- **Steep Learning Curve**. The complexity of a VE system, with its tightly coupled nature, makes it difficult for a developer to haul any benefits without becoming an expert. Unfortunately, the learning curve associated to a system is traditionally exponential. This results in a select few being sufficiently proficient with a particular system, normally the creators and maintainers.
- **One Stop Shop**. The complexity of VE involves the operation of several different sub-systems, such as rendering, networking, database, etc. Although modularity may influence the design of each subsystem, their operation remains tightly coupled to each other. Consequently, the result is a monolithic architecture, albeit modular.
- **"Not Invented Here"** and **"Reinventing The Wheel".** These syndromes imply the expenditure of resources on the reemergence of existing technology in building a VE system. Consequently, exploring new approaches becomes quite limited.

One of the fundamental goals of VE is to create alternate realities where people may interact with the environment and each other. This implies the existence of a network infrastructure connecting all the participants together. So in addition to the previous problems, the following become relevant:

- **Poor Scalability**. Most VE systems aimed at collaboration claim to support in theory thousands of users, when in reality all documented experiments in Collaborative Virtual Environments (CVEs) do not go beyond a few dozen. Even considering the military applications based on the Distributed Interactive Simulation (DIS) [DIS93] protocol, where the main design goal is the support of million of simultaneous users, the threshold remains in the few hundreds. With the online game community, the number of user base is reported to be larger at the expense of significant large budgets to increase the network and computational resources. This approach is less than ideal since notoriously the client/server architectures do not scale.

- **Poor Robustness.** The VE systems are essentially distributed systems and current solutions are not sufficiently robust, neither possesses any fault tolerant mechanisms. The situation is aggravated as the number of simultaneous users increases, thus imposing more stress on the system and associated network infrastructure.
- **"BlackBox" Network**. The network constitutes the core of the infrastructure interconnecting all systems, which may or may not be running the same VE. However, both in the VE and online game community, the knowledge of the network and its volatile nature remains naive. This results in viewing the network as a "blackbox„ where the remainder of the system interacts via a simple interface of send and receive messages. This approach results in failure of VE systems to adequately adapt to network problems or reflect behavior that is non-TCP friendly contributing to congestion collapse [Floyd99].

Upon analysis of existing VE systems, the assessment is that a significant overlap in functionality. In most cases, the system excels in particular operational features while performs poorly in others depending on the focus of the targeted set of applications.

The existing variety of different solutions clearly illustrates that there is no single solution to the various problems. This denotes that no system is ideal and that the focus should be interoperability of operational functional blocks between different systems and not enforcing the adoption of a platform, system or toolkit.

## 2.2. Component Design Methodology

The Virtual Reality Transport Protocol (vrtp) [Brutzman97] proposal, illustrated in Figure 1, provides insight into the classes of components that are part of a VE system and their relationships.

The proposed components are aggregated into three layers of flexibility:

- **Infrastructure (level 0)**. This layer provides the support to all the remainder components of a VE system. The core of this layer is the Universal Platform, which provides dynamic runtime extension and management of the remaining components.
- **Middleware (level 1)**. This layer aggregates all the components that are not coupled to a particular VE application. The various roles that components may assume may be categorized into client, server and monitor.
- **Application (level 2)**. This layer implies a ready to use system by the end-user, without the necessity to any further development. The main focus is the content production and configuring the underlying components of the system.

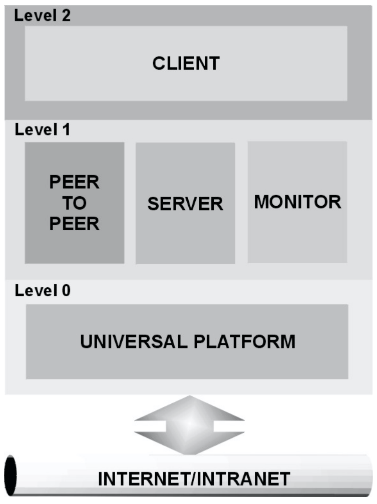

**Figure 1 -** Virtual Reality Transport Protocol (VRTP) layered component framework

## 2.3. New Trends in VE Development

Some pioneering work has been done aimed at exploring systems structured around object-oriented frameworks to address some of the problems previously mentioned.

The **MAVERIK** [Hubbold01] system consists of a small kernel based on object-oriented principles with an elegant call-back mechanism. This decouples the need for a particular internal data structure, thus allowing a particular instantiation of the system to provide a context specific data model. **MAVERIK** itself focuses on a framework, which provides the developer with great flexibility at the cost of productivity. However, to aid the development of a VE system a set of existing base Modules are provided for the purposes of rendering, navigation, spatial management and device integration. The system is for the development of single user systems, without any support for multiple simultaneous users.

The **VRJuggler** [Bierbam01] platform is a toolkit to build VE systems that are portable, flexible and configurable at runtime. The core of the toolkit is the Virtual Platform, which alleviates the developer of considering low-level details regarding the management of system resources such as devices and processes. This allows the developer to develop the system disregarding the target configuration of the hardware and expect the applications to work.

The **Bamboo** [Watsen98] component framework aims to promote reusability and code interoperability, independently of the implementation language. The core of the approach is the micro-kernel that manages at runtime modular blocks of the system encapsulated as blocks that abide the well-defined interface of a Module. Coupled with the set of language loaders, **Bamboo** promotes dynamic extensibility and interoperability. No additional functionality beyond module management is available, although a set of standard extension modules have been announced.

Although all initiatives are either based on component or object-oriented design principles, only **Bamboo** follows the conceptual ideas contained in the vrtp proposal. The **Bamboo** micro-kernel represents the Universal Platform of the infrastructure layer. The remainder two approaches, while extensible and reusable, share the problem of coupling together all the different layers of flexibility: infrastructure, middleware, and application. The results are similar to having a monolithic architecture.

## 3. The Java Adaptive Dynamic Environment

The Java Adaptive Dynamic Environment (JADE) is the core component framework of the Mayhem [Oliveira01] project. JADE permits dynamic runtime management of all components and resources of a VE system. At its core is a light-weight cross-platform micro-kernel, which becomes the foundation to any VE application. Some of the design principles are similar to the Bamboo initiative, but the adoption of the Java™ programming language allowed the focus to be on the design of the framework itself and its functionality.

The JADE reference implementation has six core java packages: Namespace; Kernel; Event; Compiler; Util; Generic. The remainder subsections will describe the most important elements of the JADE framework.

### 3.1. Namespace

Within a VE system it is necessary to maintain a numerous amount of resources. This evokes the need of being able to identify successfully a particular resource, thus the requirement of a namespace.

There are two categories of resources depending on the responsibility the VE developer has over them:

- **SystemResource**. These correspond to the resources that are generated by the VE system during its operation. Each time a SystemResource is created, then a global unique SystemID is generated that may be accessed by the developer, but never tampered with. Consequently the JADE framework ensures namespace integrity at system level. The SystemID is a tuple consisting of the current absolute time in milliseconds (with counter modifications to ensure uniqueness) and the local IP of the host machine if connected to the network.
- **Resource**. This is a subclass of SystemResource with the addendum of the means of identification by the VE system developer. The default means are a String for name and an ID, both of which are managed by the developer. Consequently, the responsibility of ensuring namespace integrity belongs to the developer. In the current implementation the ID is an int.

The design of the namespace package represents the foundation of the JADE framework. Figure 2 illustrates a simplified UML class diagram of the design.

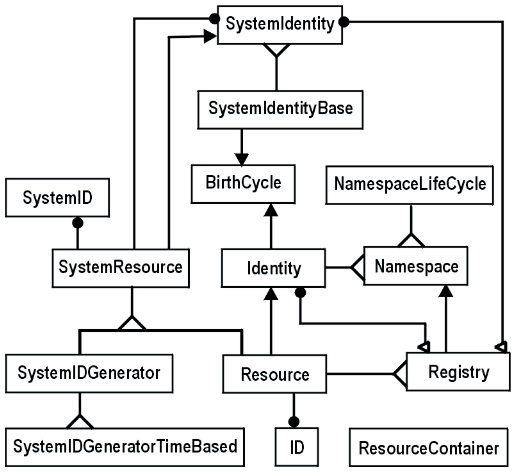

**Figure 2 -** Simplified UML class diagram of the jade.namespace package

The Namespace consists of an interface thus decoupling the implementation from the design. The reference implementation is embodied in the Registry class.

### 3.2. Kernel

The management of the JADE component framework takes place in the JADE kernel. The design of the kernel is highly flexible, allowing for the developer to provide proprietary implementations to replace one or all of the default internal functional blocks. This may be done at runtime or whilst the system is initializing by providing the appropriate configuration file.

Since it is necessary to assure easy accessibility to the JADE kernel from any place in the system, the singleton [Buschmann98] pattern was adopted. As a result, only one micro-kernel per Java Virtual Machine (JVM) exists.

## 3.3. Module

One of the cornerstones of the JADE framework is the Module, which uses the Policy pattern to delegate to the VE developer the implementation of the details. This approach allows the micro-kernel to manage the various Modules that are part of a system without concern of their particular functionality.

The granularity of a Module is entirely the responsibility of the developer. One may encapsulate an entire system within a single Module, or may break down each minimal operation into separate Modules. The developer should aim at a design with balanced flexibility and performance.

```
protected void buildMetaData();
protected void createModule();
protected void initialiseModule();
protected void activateModule();
protected void deactivateModule();
protected void shutdownModule();
protected void absorbModuleState(Module source);
```

**Figure 3 -** Core of the Module interface

Every Module developed is required to implement the policy interface denoted in Figure 3. The buildMetaData is where the developer instantiates the meta data discriminating the particular instance of the Module, this information may be used for both security and reload mechanisms of the kernel. As illustrated in the absorbModuleState method, the developer is responsible for implementing the state transition of one Module instance to another of the same type.

The Module interface reflects its life cycle[*]: *void*, *created*, *ready, activated* and *deactivated*. When a Module is initiated its state becomes void, allowing its member variables to be configured as required if different from default. Once the Module is *created* then it is ready to be loaded into the Kernel and initialized, becoming ready for use. During the remainder of the Module's life cycle, its state oscillates between *ready*, *activated* and *deactivated*, until it the operation shutdown is evoked. The shutdown of the Module returns it to *void*.

[*] The Module expands the Resource's life cycle (BirthCycle) by adding the activate and deactivate operations (LifeCycle).

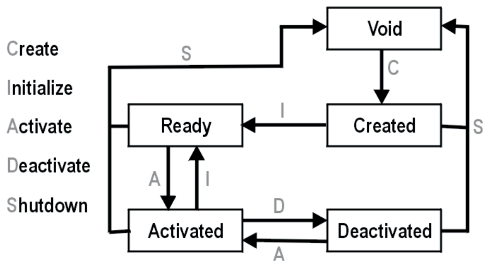

**Figure 4 -** Module life cycle

While in the *ready*, *activated* or *deactivated* state, it is possible for a Module to be reloaded, although due to security restriction the operation is only permissible to the owner of the Module. To assist the reload operation, a Module contains a description that encapsulates its name, version and relevant urls.

The execution of a Module is passive within its owner's context. However, should it be necessary for it to be concurrent, there exists the ModuleRunnable that is contained within a thread context of its own.

## 3.4. ModuleManager

A ModuleManager is a specialized Module that is a container of other Modules and is responsible for their management. Since each Module has a unique ID that identifies it, the resulting namespace in the VE system has a hierarchical nature. The JADE micro-kernel is a derived ModuleManager with functionality beyond just Module management. The operations available are summarized in Figure 5, where (…) contains either the unique ID of the target Module or the pair (owner, module).

The accessibility of the getModule method is intentionally protected, thereby delegating to the derived subclasses the responsibility of exposing the contents of the ModuleManager. The JADE kernel provides such a method retrieveModule(…), which may be either blocking or non-blocking.

In Figure 5, three protected methods exist to enforce the existence of a ReloadPolicy, SecurityPolicy and a Namespace in case none are provided by access methods:

- **ReloadPolicy**. This policy is used to determine how a Module is reloaded. By default the JADE micro-kernel validates version numbers of the Modules, choosing the most recent.
- **SecurityPolicy**. This policy is invoked preceding any operation of a Module and determines its execution.

```
public void activate(…);
public void deactivate(…);
public void shutdown(…);
public void initialize(…);
public void loadModule(…);
public void reloadModule(…);
public void unloadModule(…);
public void unloadAll();
protected Module getModule(…);
protected abstract void buildReloadPolicy();
protected abstract void buildSecurityPolicy();
```

**Figure 5 -** Core of the ModuleManager class

### 3.5. Resource Locator

The ResourceLocator is the fundamental helper class of the JADE kernel. It is responsible for retrieving a resource from a given source and dispatching it to the appropriate sink as illustrated by Figure 6. The ResourceLocator may manage any type of resource provided the appropriate Retriever and Handler exist:

- **Retriever**. The role of this element is to retrieve a resource from its location. By default, there exist three retrievers: local hard disk, Hyper Text Transfer Protocol (HTTP) and the Jini service.
- **Handler**. Once a resource is retrieved, the Retriever delegates it to the respective Handler. The type defined in the resource descriptor determines which handler to evoke. There exist handlers for native system libraries, Modules and Classes.

The process of locating a resource begins by the ResourceLocator receiving a resource descriptor. A caching mechanism is used to validate if the resource has been previously retrieved. This may be overridden when necessary by setting a flag. The retrieval process is determined by the protocol defined in the URL of the resource. If no protocol is identified, then it is assumed that the resource is located on a file system, either local or remote and the system variables for paths are used. Once the resource has been retrieved, it is delegated to the appropriate Handler based on its nature (Module, Library, Class or any content specific resource such as textures, vrml [VRML97], etc). In the case of a Module or a Java class, the ResourceLocator resolves the associated classfile and dynamically links it into the system. The class resolution verifies any dependencies and retrieves all the other necessary classes from the either the local classpath or the same URL of the initial class/Module.

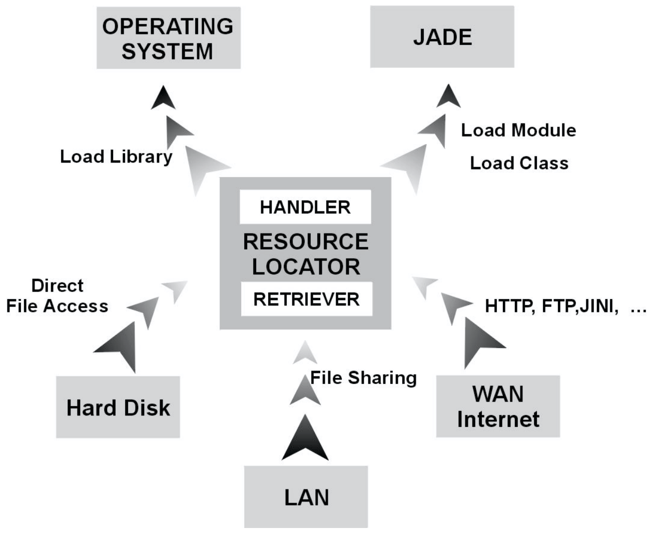

**Figure 6 -** ResourceLocator with sources/sinks

The handling of libraries via the Java Native Interface (JNI) allows the integration of native code (C/C++, Pascal, Assembly, LISP, etc). However, this option will compromise the portability of the VE system, requiring the existence of a different library for each system.

### 3.6. Event Model

The JADE kernel may support any number of Modules, but most likely their operation is not isolated from one another. Therefore it is necessary to provide some means for data to be exchange between Modules and for events to be triggered.

All the existing mechanisms have their advantages and disadvantages. The choice of which mechanism to adopt is delegated to the VE system developer.

The simplest and most efficient form of communication is direct accessibility to a Module. This is possible since each Module is retrievable via the Namespace managed by the JADE kernel. However, this approach requires prior knowledge of the target Module or to use reflection to determine the interface at run-time. The adoption of this method is discouraged since it produces Modules that are tightly coupled to one another.

The JADE framework provides two other forms for communication between Modules, both based on the Subscriber/Publisher pattern using events. An overview of the event package is depicted in the UML class diagram of Figure 7.

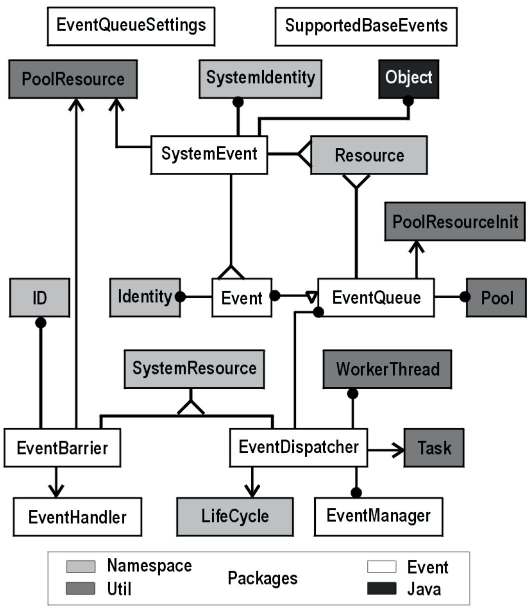

**Figure 7 -** Simplified UML class diagram of the jade.event.package. Only immediate relationships to classes in other packages are illustrated

The base SystemEvent indicates the source, type of event and any arguments necessary to be processed by the target subscribers. In the case the source is beyond the scope of the JADE kernel then it is necessary to use the Event class that tracks the source Identity.

Any Module (or code within the system) may generate events. The way in which an event is disseminated characterizes the corresponding event model being used, either distributed or central.

**Distributed Event Model**

Each Module may accept the direct registration of a subscriber's interest. Whenever an event is generated, the Module informs each of the subscribers that belong to the set of listeners.

While this approach is more robust than a central architecture, it requires that a subscriber to retrieve the reference of the desired Module to register their interest. Another disadvantage is the impossibility of a subscriber just indicating their interest in just the type of an event independent of its source.

**Central Event Model**

This approach relies on the existence of a central event manager responsible for dispatching events. Whenever an event is generated, it is posted to the event manager, which places it into the appropriate dispatch queue according to its type, priority and source.

Any subscriber interested in receiving events is required to register their interest with the event manager, indicating their interest either in a source or particular type of event independent of its origin.

Upon subscribing, it is possible for the subscriber to denote if the interest is blocking or non-blocking by using EventBarriers. In the case of blocking, the subscriber is blocked until the event occurs once and only once. While in the non-blocking case, the subscriber is notified each time the event occurs or a particular source generates an event. In this case, it is necessary to explicitly deregister when there is no more interest in receiving events.

It is possible to create dispatchers and event queues as necessary, but by default the JADE kernel provides three with varying degrees of priority (high, normal, low).

## 3.7. Configuration

The configuration of the JADE kernel and the resulting system is done either via a command line interpreter or a file. In both cases, the associated mechanisms are based on a syntax parser coupled with a semantic parser. Thus, the semantic parser may be shared across different syntax parsers, such as eXtensible Markup Language (XML) or online commands.

The compiler framelet [Fayad99] is reused throughout other frameworks, such as the TreacleWell (TW) networking module and the Meta Interest Management (MIM).

The compiler, similar to other supporting elements of the kernel, has a default implementation, which may be replaced by the developer provided the corresponding interfaces are respected.

## 3.8. Evaluation

A common drawback of existing VE systems, namely online games, is the cumbersome process of applying upgrades. Traditionally the upgrade operation requires the system to be offline or before the user begins interacting with the VE.

One of the design goals of the JADE framework was the dynamic extensibility of the VE system at runtime. In Figure 8 it is possible to see the evolution of the rendering Module from flat shaded to depth shaded and finally to textured map. All the transitions occurred whilst the user was engaged in the VE, without requiring the system to go offline.

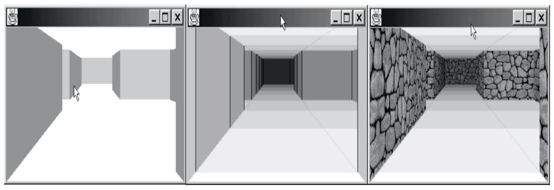

**Figure 8 -** Dynamic replacement of the Renderer Module at runtime in the Maze game test program

## 4. Conclusions

The wide proliferation of existing VE systems clearly demonstrates that the current development trend is costly in terms of manpower, time and monetary investment. The lack of interoperability makes it hard to reuse any code and for rapid knowledge transfer. As a result there are many VE systems that excel in its design objectives whilst performing inadequate, if not miserably, in the remainder operational blocks of the systems. Although this problem has been recognized within the field of VE research and the gaming industry, little progress has been made in finding a solution.

We argue that a possible alternative to the current development trend is the adoption of component design methodologies. The JADE component framework provides an alternative approach that addresses most of the shortcomings of the existing development trend.

In addition to the JADE framework, a set of middleware Modules have been developed, such as:

- **Meta Interest Management (MIM).** This Module is responsible for handling the Area Of Interest Management (AOIM). It consists of a component framework that allows the multiple simulatenous usage of different policies to avoid design constraints on the VE itself (ie: visibility based policy for indoors with grid based for outdoors)
- **TreacleWell** [Oliveira02]. This Module is responsible for the networking of the VE system. It is a three tiered component framework based on the network design principle of micro-protocols.
- **Meta Unified Datamodel (MUD).** This Module is responsible for the data management of a VE system using extensively the Model/View/Controller pattern. This approach allows for users to interact together independently of their devices (PC, PDA, Head Mounted Display (HMD), mobile phone (G3), etc).

Although the adoption of the JADE framework (not its reference implementation) is a requirement, the modules mentioned above are not. The developer gains productivity by employing them, but at the possible cost of flexibility. The main reason for a layered approach is the impossibility of building the ideal system that addresses all the functional requirements of any user.

The **NPSNET-V** [Kapolka02] project shares common design principles and concepts with JADE, including terminology. An objective of future work is to analyze the independent code implementations and synchronize the two designs to obtain a single component framework for future VE system development..

**Acknowledgments**

This work is sponsored by Alfamicro and the Portuguese Foundation for Science and Technology. We wish to thank the many design discussions with Anthony Steed that helped refine the JADE framework

## References

1. D. Anderson, J. Barrus, J. Howard, C. Rich and R. Waters, “Building Multi-User Interactive Environments at MERL„, IEEE Multimedia, Winter 1995.
2. A. Bierbam, C. Just, P. Hartling, K. Meinert, Baker, and C. Cruz-Neira, “VR Juggler: A Virtual Platform for Virtual Reality Application Development„, Proc. IEEE VR’2001, Yokahama, March, 2001.
3. D. Brutzman, M. Zyda, K. Watsen and M. Macedonia, “Virtual Reality Transfer Protocol (vrtp) Design Rationale„, Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality, MIT, Cambridge Massachusetts, June, 1997.
4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, “A System of Patterns„, John Wiley & Sons, 1998.
5. “Standard for Information Technology – Protocols for Distributed Interactive Simulation Applications„, Institute for Simulation and Training, Technical Report IST-CR-93-15, May 1993.
6. M. Fayad, D. Schmidt and R. Johnson, “Building Application Frameworks„, Wiley computer publishing, 1999.
7. S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet„, IEEE/ACM Transactions On Networking, Vol 7, N. 4, August 1999.
8. W. Gibson, “Neuromancer„, Ace Books, New York, 1984.
9. C. Greenhalgh, J. Purbrick and D. Snowdon, “Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring„, Proc. ACM CVE’00, San

Francisco, September, 2000.
10. O. Hagsand, "Interactive Multiuser VEs in the DIVE System„, IEEE Multimedia, Spring 1996, Vol. 3, N. 1, IEEE Computer Society.
11. R. Hubbold, J. Cook, M. Kaetes, S. Gibson, T. Howard, A. Murta, A. West and S. Pettifer, "GNU/MAVERIK A Micro-Kernel for Large-Scale Virtual Environments„, Presence, Vol. 10, N.1, 2001.
12. A. Kapolka, D. McGregor and M. Capps, "A Unified Component Framework for Dynamically Extensible Virtual Environments„, Proc. ACM CVE'02, Bonn, September, 2002.
13. M. Macedonia, M. Zyda, D. Pratt, P. Barham and S. Zestwitz, "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments„, Presence: Teleoperators and Virtual Environments, Vol. 3, N. 4, 1994.
14. M. Oliveira, J. Crowcroft and M. Slater, "Mayhem in Online Game and Virtual Environment Development„, short paper, Eurographics'01, Manchester, September, 2001.
15. M. Oliveira, J. Crowcroft and M. Slater, "TreacleWell: Unraveling the Magic "Black Box" of the Network„, Proc. SCI'02, Orlando, July, 2002
16. Sense 8, "WorldToolkit Technical Overview„, 1998.
17. N. Stephenson, "Snow Crash„, Roc (Penguin Books), New York, 1992.
18. H. Tramberend, "Avocado: A Distributed Virtual Reality Framework„, Proc. IEEE VR'1998, 1998
19. "Virtual Reality Modeling Language„, ISO/IEC DIS 14772-1, April 1997.
20. K. Watsen and M. Zyda, "Bamboo – A Portable System for Dynamically Extensible, Real-Time, Networked, Virtual Environments„, Proc. IEEE Virtual Reality Annual International Symposium (VRAIS'98), Atlanta, March 1998.