Extensible Virtual Environment Systems Using System of Systems Engineering Approach

Manuel Oliveira
University College London
m.oliveira@cs.ucl.ac.uk

Joao Pereira
INESC-ID
jap@inesc.pt

Abstract

The development of Virtual Environment (VE) systems is a challenging endeavor with a complex problem domain. The experience in the past decade has helped contribute significantly to various measures of software quality of the resulting VE systems. However, the resulting solutions remain monolithic in nature without addressing successfully the issue of system interoperability and software aging. This paper argues that the problem resides in the traditional system centric approach and that an alternative approach based on system of systems engineering is necessary. As a result, the paper presents a reference architecture based on layers, where only the core is required for deployment and all others are optional. The paper also presents an evaluation methodology to assess the validity of the resulting architecture, which was applied to the proposed core layer and involving individual sessions with 12 experts in developing VE systems.

1. Introduction

A Virtual Environment (VE), within the context of this paper, consists of an alternate reality existing in the digital real where people come together to play, work and socialize together. This means that an online game qualifies as a VE. The VE community has recognized that there is no ideal VE system that addresses entire problem domain with all the desired functionality. This leads to the current state of affairs where there are a significant number of different VE systems, each catering a particular set of functional requirements. When the set of user requirements divert from the targeted initial user functionality supported by the VE system, then either the initial functionality is scoped to the one supported by the chosen VE solution or significant work is carried out that potentially may lead to a new solution, as in the case of [16]. Many of the difficulties associated to VE system development are derived from the complexity of the problem domain associated to VE. The adoption of software engineering principles and methodologies has allowed for gradual improvement to the software quality of various VE systems in terms of stability and flexibility. However, the monolithic nature of VE system architecture remains in place, making it difficult to achieve code interoperability, which would allow a developer to combine elements from different systems. As a result, the choice of VE system still places constraints on the supported functionality. This is clearly evident in the clone phenomena of the game engines that support a particular genre, which normally results in a new game engine if new functionality is necessary.

This paper presents an approach to develop VE systems that are more resilient to the effects of software aging [19] and promote code interoperability between different solutions. The next section presents a short overview of the related work within the VE community, but also some lessons are drawn from the operating systems community. The proposed reference architecture based on layers is provided in section 3, which then leads to description of its foundational layer (section 4). The results of an evaluation methodology are discussed in section 5. In section 6, examples of other components from the other layers are briefly discussed. Finally some conclusions are drawn in section 7.

2. Related Work

The related work touches on some of the existing VE systems, but also draws some lessons from operating systems community.

2.1. Virtual Environment Systems

The Distributed Interactive VE (DIVE) [10] system scoped the problem domain to collaborative VEs with particular focus on close high fidelity social
communication within small groups of users. The DIVE system supports a partial replication policy of the data model across a peer-to-peer network architecture using multicast as the communication model to distribute changes to the database. Each host initiates their session by loading the entire world from various file locations across the network or through state transfer. Any local changes are communicated to other remote hosts by means of events and consequently, all updates from remote hosts are received as events via the network. An important design decision was to develop a system that supported rapid development of a VE by means of content development combined with scripting to support the dynamics of an alternate reality. Although DIVE has greatly benefited from software engineering practices and achieved high internal flexibility, the system remains monolithic from an external perspective without the possibility of extracting a sub-system to integrate into another VE system.

The VRJuggler platform [2] is a toolkit to build a VE system that is portable, flexible and configurable at runtime. The core of the toolkit is the vjKernel, which alleviates the developer of considering low-level details regarding the management of system resources such as devices and processes. The main aim of VRJuggler is to provide a Virtual Platform (VP) that makes devices, and their configuration, transparent to developers. This allows the developers to build a VE system disregarding the target configuration of the hardware and expect the applications to work as a result of the decoupling. The VRJuggler is not a turn-key application that supports users within a VE. A key limitation, resulting from a design decision, is the inability of supporting multi-users across a network. However, VRJuggler does support multiple users locally sharing the same hardware responsible for the rendering albeit each user having their own devices. Although the system achieves the target goals, VRJuggler enforces a non-flexible framework for rendering.

The VHD++ [20] main focus is to provide an open flexible system as the on within the community of VEs and the games industry. The VHD++ system takes a three layered approach, consisting of a system, simulation and application. However, all layers are mandatory. The semantics increase from the system towards the application, which implies an increase in productivity whilst reducing the flexibility. The VHD++ core includes some high semantics, such as the scheduler that fails to support real-time as intended, thus introducing code hematomas since the application is required to deal with timing issues. The richness of features supported by the kernel raises the potential for implementation dilemmas.

The main motivation of MAVERIK [11] is to provide a VE system that abstracts the rendering process from the spatial data structure and particular processing methods. As with other solutions, design choices were made that do not support code interoperability albeit the high flexibility to support multiple rendering techniques. The implementation is based on a single processing loop, which imposes performance constraints and although the pipeline is highly flexible techniques such as global illumination and transparency are not easily supported.

The Bamboo [21] VE system aims to provide a flexible open system to facilitate the development of VE systems. The foundational layer is the Netscape Portable Runtime (NSPR), which provides a platform abstraction over the different operating systems, thereby facilitating some code that is cross-platform. The underlying design principle of Bamboo is to decompose a system into well specified building blocks encompassed into Modules, which consists of a well-defined interface used by the Kernel for management of the system. During run-time the Bamboo Kernel uses configuration files to locate the Modules to be dynamically linked, resolving any dependencies. The Kernel uses Language Loaders, as plug-ins, as an abstraction from the particular implementation language used in the development of a Module. It is not the intent of Bamboo to be used as a VE system. Instead the high semantics is delegated to the development of new Modules or the usage of existing Modules.

The High Level Architecture (HLA) [12] is a de facto standard of a software architecture for building large scale VEs based on simulation components. Unlike the previous solutions, HLA has both reusability and interoperability as two of its system attributes to allow mix of simulation components from different sources. Although HLA has become a standard and has achieved some of its design goals, its usage remains within the domain of military simulation projects. The findings of [1] corroborate this tacit understanding, identifying the cost/benefit ratio as the main culprit. The adoption of HLA implies significant costs in development resources and reuse is impeded by a very steep learning curve coupled with an overly complex architecture with performance constraints and semantic interoperability problems.

2.2. Operating Systems

With operating systems, the initial monolithic architectures raised serious problems such as flexibility, extensibility, reliability and maintenance. In response to the drawbacks, new architectures emerged,
namely the concept of micro-kernel [7] that consisted on a minimal kernel where the services and policies are delegated to the user space of an operating system. The first generation of micro-kernels (ie: Choices [5]) produced disappointing results since their utility was compromised by the poor performance and failure to adhere to the core design principles of small, simple and flexible kernels. A careful analysis [14] demonstrates that the fallacy related to poor performance is due to poor selection of implementation strategies. This realisation led to a second generation [15] of micro-kernels (ie: Exokernel[9]) where major restructuring was done to increase the performance whilst addressing the shortcomings in the design. To address the concerns of flexibility, some solutions, such as Apertos [22], have adopted open implementation (OI) as a foundational design paradigm, but the associated performance penalty to the approach is a significant deterrent for wider adoption. The performance concerns led to the Exokernel design where abstractions were deemed to be the cause of the capitulation of the micro-kernel concept, thus all abstractions were eliminated [8]. The result was tight coupling of the operating system to a particular hardware configuration and consequently large applications burdened with repeated common functionality. Therefore, the challenge is to achieve the best equilibrium between flexibility and performance.

3. A System of Systems Approach

A key strategy towards handling complexity of a problem domain is system decomposition [18]. Although this has been used successfully to deal with internal modifiability of a system, code interoperability between different solutions remains an extreme difficult challenge that remains unsolved. This is largely due to the single system engineering approach, which leads to solutions with closed boundaries and consequently may be viewed as monolithic, forcing a person to either adopt the entire solution and be constrained by the associated sub-domain, or to develop a new solution tailored to the desired functionality that could not be addressed by the extensibility constraints of an existing solution, as reported in the case study of [16]. The proposed architecture, presented in UML2.0 diagram of Figure 1, takes decomposition further resorting to a system of systems engineering approach, where all the architecture is functionally coherent, but each subsystem is considered independently from one another and totally decoupled. The low level APIs aggregate the common libraries that are accessible to the remainder of the system, such as OpenGL and BSD Sockets.

![Figure 1 – Reference VE System Architecture](image)

The reference architecture identifies five main layers, each graphically represented by the UML icon of a package:

- The Virtual Environment Platform (VEP) is the core of the reference architecture and common to all VE systems. The design of the core should avoid any potential implementation dilemmas, thus being totally devoid of any semantic connotation of any particular subset of the problem domain of VE systems. As a result, the main functionality of the VEP is resource management based on an extensible kernel that is supported by a security model;

- The traditional core functionality of VE systems is to be captured and deployed as components that correspond to systems. These are aggregated together in the Core Virtual Environment Platform Components (CVEPC), of which Rendering subsystems (either one integrated solution, or a collection of different subsystems), the sensorial interface framework (both input and output devices) and the network subsystem are clear candidates. A distinguishing factor that qualifies a component as CVEPC is the fact that a VE system will be severely hampered in supporting the
functionality of VE system (ie: without a Network subsystem, it is not possible to support multiple users and no system can do without at least a visual rendering engine). Therefore these component subsystems should be carefully designed to support a wide range of applications since implementation dilemmas will reduce their potential of adoption and reuse. However, anyone of these components is optional and may be replaced by monolithic components from more semantically rich layers;

• The Virtual Environment Components (VEC) comprises all the components that are closer coupled to the particular application sub-domain being addressed by the system. This is the case of the Physics, Animation and Avatar to name but just a few;

• The Virtual Environment Application (VEA) layer contains all the remainder elements that are tightly coupled to the particular application domain of the VE system. So for example, the behaviors of geometric objects within the environment would be a likely candidate;

• The Sensorial Interface is a self-contained layer that is coupled to the underlying hardware, whilst providing a generic abstraction in terms of input/output for the other layers.

4. Virtual Environment Platform

The Virtual Environment Platform (VEP) aims to provide what is common to VE systems, thus much of the identified components should be present in different forms in each existing system. The design of the VEP requires the elimination of all abstractions with the slightest semantic connotation to a process or method not common to any virtual environment system. However, the platform must support dynamic extensibility beyond the base functionality of the system. An initial step to determine the scope and nature of the system requirements of the VEP is to derive some scenarios.

4.1. Scenarios

A survey [6] of software architecture methodologies demonstrates that a common element is the use of scenarios to aid the different stakeholders to agree and prioritise the required functionality of the system.

Table 1 – Final revised scenarios for VEP

<table>
<thead>
<tr>
<th>Nº</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A VE system can be built from a set of components provided by different sources.</td>
</tr>
</tbody>
</table>

There were multiple iteration cycles to validate, refine and add scenarios defining what would be the minimal required functionality to be supported by the VEP resulting in the scenarios of Table 1.

4.2. Reference Architecture

The VEP reference architecture (Figure 2) was implemented in the Java language as Java Adaptive Dynamic Environment (JADE), which some interfaces are described in [17] (the platform has been further refined but without any changes to architecture).

4.2.1. Security. The topic of security is an often neglected requirement within the VE community. This is exemplified by the Distributed Interactive
Simulation (DIS) where the target informs the simulation if it was hit, thus invulnerability is simply achieved by ignoring any HIT messages. With the possibility of run-time extensibility, security gains new significance as it is necessary to avoid rogue components to execute malicious or unauthorized behavior. The security subsystem needs to support authentication, access control, availability, integrity and auditing, but the framework should be based on a policy based mechanism that can be textually configured.

_In JADE, the choice was made to extend the security manager of the Java Virtual Machine (JVM) with its fine grain policies._

4.2.2. Resource Management. The resource management is without doubt the most complex part of the VEP with the following four subsystems:

- **Configuration.** With dynamic systems that are highly flexible it becomes necessary to provide an automated mechanism for configuration of the system and its resources. The interface reduces itself to setting up the parser, the syntax and interpreter to be used. This framework is common to most configuration implementation strategies and scripting engines, thus by establishing a common bootstrap procedure. In addition, a method should exist to permit polling of the state of the Configuration subsystem.
  - _In JADE, the Configuration subsystem supports a command line mechanism and a nested SAX parser used in the configuration of all components of the VE system, albeit each component having its own semantic interpreter._

- **Communication.** It is necessary for the various elements within a VE to communicate with each other. The use of direct interface leads to tight coupling, which compromises interoperability and reduces software aging resilience. An alternative approach is to adopt a communication mechanism where the elements within a system do not require prior knowledge of the target interfaces whilst supporting asynchronous and asynchronous communication. This can be achieved by means of events that provide information concerning the source and what type of event was triggered. The essence of any event model is the Publisher/Subscriber pattern and it should be independent if the system is executing on a single or multiple processes/machines.
  - _In JADE, two variations of an event model are available, centralized and delegation._

- **Namespace.** In any VE system, there are numerous resources with a wide and diverse nature, ranging from a simple image as a texture to a complex mathematical model encoded in specific programming language. When considering code based resources, it is necessary to validate with the security manager if the resource can be integrated into the system and with what privileges. Therefore, a main building block of VEP is Namespace, which corresponds to an abstraction providing a context for resources. In addition to a clear interface to manage namespace, finding or retrieving resources is done based on the use of a Search Criteria. Since the result of a Search Criteria use is a Namespace, it is possible to chain searches with different relational operations, such as AND, OR, XOR.
  - _In JADE there is a default implementation for a local object registry, a SQL database based on object/relational mapping and a distributed namespace based on the Common Object Request Broker Architecture (CORBA; along with a library of search criteria._

4.2.3. Executive Kernel. The Executive Kernel is responsible for the runtime management of the various components during the lifetime of a VE system instantiation. The kernel is itself a Namespace that has a Communication model (event model), a Configuration framework, a Resource Locator and a Security Manager.

_In JADE, any of the subsystems are replaceable at compile-, link- or run-time._
5. Evaluation

Taking into consideration that VEP is not a monolithic system presented itself as a turn-key solution, it is necessary additional resource expenditure to develop the customized functionality of a particular VE system. The evaluation focuses on the quality attributes of the corresponding software architecture. Therefore, some of the traditional attributes measured in software architecture evaluations are not pertinent since they are implementation dependent.

Table 2 - Outline of evaluation methodology

<table>
<thead>
<tr>
<th>Nº</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This step presents overview of evaluation method and the documentation concerning the reference architecture and its core layer - VEP.</td>
</tr>
<tr>
<td>2</td>
<td>Questionnaire that helps identify the stakeholder profile.</td>
</tr>
<tr>
<td>3</td>
<td>This step presents the vision that drives the reference architecture and analysis of related work.</td>
</tr>
<tr>
<td>4</td>
<td>This step consists of the identification of the main scenarios to be supported by the software architecture.</td>
</tr>
<tr>
<td>5</td>
<td>This step presents the software architecture of the VEP, using JADE as the reference implementation.</td>
</tr>
<tr>
<td>6</td>
<td>Revision of the scenarios identified in step 4 and their reprioritisation, selecting the most important ones.</td>
</tr>
<tr>
<td>7</td>
<td>Discussion of the architecture to validate if the final set of scenarios of step 6 can be supported by JADE. The result may identify some architectural changes.</td>
</tr>
<tr>
<td>8</td>
<td>Overall evaluation by the stakeholder of the architecture in fulfilling the initial scenarios and having the flexibility to support the revised scenarios.</td>
</tr>
<tr>
<td>9</td>
<td>Questionnaire that quantitatively captures the evaluation.</td>
</tr>
<tr>
<td>10</td>
<td>This step wraps up the process by collating all the information generated and synthesizing the results.</td>
</tr>
</tbody>
</table>

The proposed evaluation methodology is based on an extension of the Software Architecture Analysis Method (SAAM) [13], consisting of 10 well-defined steps as outlined in Table 2, which are aggregated into four distinctive phases:

- **Briefing.** Presentation of all relevant documentation pertaining evaluation method and the VE architecture and various layers. Steps 1 and 2.
- **Analysis.** Discussion of scenarios driving the VE system requirements to be addressed by the architecture. Steps 3, 4 and 5.
- **Brainstorm.** Revision of scenarios and impact on architecture, leading to evaluation. Steps 6, 7 and 8.
- **Synthesis.** Final evaluation of the expert and cross-analysis of the evaluation results from all experts. Step 9 and 10.

The aim of the evaluation was to target experts within the field of VEs or games. Therefore, the recruitment of subjects was done based on scheduled working sessions with each expert.

5.1. Questionnaires

The questionnaire from step 2 consisted of a questionnaire with 24 questions aiming to determine the experience of the expert concerning VE system development. In addition, the questionnaire would capture their inclination towards software quality and their awareness of the problems that plague the traditional development process.

The questionnaire from step 9 consisted of 16 questions with the aim to collate qualitatively the assessment of experts concerning the VE reference architecture and VEP. In particular, four questions were targeted at qualifying their predisposition to adopting the JADE in initiating future projects.

The responses were based on a Likert scale, between 1 and 7. The lower-end of the scale normally corresponds to when the respondent has a negative response to the question, whilst the upper-end of the scale indicates that the respondent has a positive response to the question. The questions were carefully structured on a qualitative approach since it would be difficult to homogenise a quantitative assessment across all stakeholders.

5.2. Experts

The evaluation method was carried out with the twelve individuals from different organizations and with different roles, albeit sharing a deep expertise in developing VE systems.

Each assessment consisted of an individual working session with an expert. The shortest session lasted 105 minutes, whilst the longest went beyond 310 minutes, but the median was 145 minutes.

5.3. Analysis

All the experts approached the problem of evaluating the reference architecture and then stressing JADE by considering how their current and future work could adopt the architectures. The maturity of JADE was demonstrated by the fact that no architectural changes were identified and any new
scenario was either foreseen or easily accommodated without any modifications.

Considering the varied background of the projects, from crowd simulation, online games, avatars, collaborative VEs, research in presence and co-presence, etc, JADE demonstrated to fulfill all the necessary requirements of different sub-domains of the problem domain associated to VE.

The evaluation also identified a keen interest to have further information and more details concerning the taxonomy and architectural guidelines for potential use in future projects involving the development of a VE system.

The final assessment indicates that the respondents agreed on the utility of a VEP and its potential for promoting reusability and interoperability in the development of VE systems. There was also agreement that JADE was deemed an appropriate reference implementation for the VEP. In addition to the informal evaluation methodology, the reference architecture and JADE reference implementation has been adopted successfully for different student projects, which has resulted in a rich library of components.

6. Components

Although VEP is the foundational layer that needs to be present, all the remainder layers are optional. This section will describe some examples from the other layers.

6.1. Core Virtual Environment Components

TreacleWell [23] is a component framework that supports networking functionality. It has four distinctive elements:

- **Connectors.** These elements provide a clear abstraction to the network for data communication, mapping the internal representation to the network device;
- **Messages.** This is an abstraction on the data to be handled by the various elements in TreacleWell, but the developer may choose on a particular implementation strategy (default is array of bytes);
- **Buffers.** These elements are repositories for messages, with multiple implementation strategies: sets, heaps, FIFO queues, tuple space, etc;
- **Flows.** These are containers of smaller FlowElements that the developer may connect together in an directed graph, supported by a centralized event model to ensure total decoupling.

A library of the four identified elements is available for a developer to build together a Well for data communication using an XML configuration file(s).

6.2. Virtual Environment Components

Some of the Virtual Environment Components developed to ease the development of VE systems are:

- **Meta Interest Management (MIM).** This component provides a framework for management of receiver interest independent of the particular interest policy used. MIM provides policies for static spatial interest and for aura based interest. MIM can be integrated with TreacleWell for the different communication architectures associated to the various interest models;
- **Meta Unified Datamodel (MUD).** The MUD framework is built upon the Model-View-Controller pattern, resulting in Node-Visual-Behavior. The resulting Module embodies the datamodel of a VE application and manages the data, which departs from the traditional approach of a single scenegraph;
- **Perceptual Network Metaphors (PNM)** [24]. This component provides a framework to bridge the problems of network communication and its impact on the immersiveness of the user. The approach taken is to integrate seamlessly into the perceptual feedback cycle of the user the additional information concerning the state of the network via a metaphor, such as the weather (congested network represented by rain in the VE).

7. Conclusions and Future Work

The adoption of software engineering principles has significantly improved the flexibility of VE systems, but interoperability remains a difficult challenge to solve, with solutions continuing to have a monolithic architecture that is coupled to a particular scope of the VE problem domain.

This paper presented a reference architecture based on a system of systems development approach and provided an overview of its foundational layer. Although the individual architectural elements are common to most systems, the adopted approach presented in the paper aims to provide a solution that addresses the issues of interoperability and software aging in addition to the classic software attributes of existing systems.

The overall result of the evaluation methodology recognizes the potential of the proposed VE reference architecture and the associated VEP to achieve the goal
of system interoperability whilst mitigating the effects of software aging.

8. Acknowledgements

This work was sponsored by the Portuguese Foundation for Science and Technology. The work has benefited from the useful and insightful comments from Anthony Steed, Jon Crowcroft and Mel Slater.

9. References


