SOFTWARE ENGINEERING OF DISTRIBUTED SIMULATION ENVIRONMENTS

James Duff James Purtilo

Institute for Advanced Computer Studies

Computer Science Department

University of Maryland at College Park

Michael Capps David Stotts

University of North Carolina at Chapel Hill

Abstract

With the increasing popularity of simulation and virtual environment software, it is necessary to provide software engineering techniques to simulation program designers. In this paper we lay out the requirements that any such techniques will have to meet, then suggest a formalism and an interconnection tool that will allow the interconnection of re-usable simulator components to build distributed simulation software.

This research has been supported in part by the Office of Naval Research.

1 INTRODUCTION

In great part, software engineering is about predictability. Among the many ways open for us to build software, those based on principles embodied by software engineering methods are often preferred, in the hope that doing so will lead to a greater likelihood of the product having acceptable cost, timeliness and reliability. Embarking upon each software development effort as if an experiment – where we don't know where the road will take us – is no longer acceptable.

Today, software engineering often gives us this desired predictability. But not always. The successes are largely confined to situations where we can model requirements entirely within a single notational system, since it is when developers must incorporate diverse types of requirements that serious problems arise. For example, notations for expressing demands placed on a database do not serve developers attempting to employ formalisms in modeling user interfaces; neither do the analysis techniques for control systems well apply to the emerging area of control integration; and so on. We have tools to help us in each of many demanding areas, but by the same token we have little ability to integrate 'heterogeneous requirements.' Construction of applications in rich domains often regress to experiments – or gambles.

Our relative inability to simultaneously analyze these multiple classes of requirements adequately is a key barrier to realizing benefits of software engineering as broadly as we would like. The ability to integrate multiple formalisms – so developers working in rich and diverse application areas can be effective software engineers too – is also the object of study in our research at the University of Maryland.

The purpose of this paper is to examine one slice of the overall problem of heterogeneous software engineering, in order to illustrate some of the issues that arise, and to suggest a few techniques to assist software engineers in the area. We do so by focusing on controlled development of *distributed simulation programs*, a rapidly growing segment of the community. Distributed virtual environments, distributed simulation tools, networked gaming software, and even multi-user-dungeons are examples of this expanding market segment.

Simulation software is often very complex, combining hard problems from such diverse areas as real-time systems, databases, high performance I/O, visualization and modeling, graphics, parallel programming and computer architecture. For example, even a stand-alone Virtual Reality (VR) program must use complicated algorithms for collision detection, synchronization, timing, rendering, and the synthesis of audio and visual data, in order to provide realistic behavior at acceptable performance. Software engineers must not only be given support for designing in each of these areas, but must also be provided *interconnection notations* that allow the various parts to be 'glued' together.

Further complicating the problem for software engineers is the fact that simulation software is necessarily hardware-specific, and contains low-level code. The critical need for excellent simulator performance means that software engineers are unable to use any programming abstractions that hurt efficient operation.

This complexity means that realistic simulation environments can be extremely expensive to build. Furthermore, with so many difficult technical problems to overcome, software engineers have not put much effort into designing simulators as *components* that can be reused. Thus most large-scale simulation software development has produced "stand-alone" systems, so that every new project must start from scratch, reusing only those components that individual programmers are knowledgeable about.

We are studying how best to provide software engineering techniques to designers of distributed simulation environments, to enable them to reason about their systems, and build those systems quickly to be both correct and efficient. As a result of this research, we will provide an interconnection notation that allows software designers to glue together disparate control notations, make high-level design decisions, and build re-usable components. Currently there is not a clear understanding as to the requirements for the engineering of distributed simulators, not to mention how those requirements should be met.

In the next section, we present an example to both motivate the need for component-based simulation software engineering and help us discover the requirements of such a system. Then in Section 3, we present a description of our proposed partial solution, relate the proposed software architecture to the discovered requirements, and describe our work to date. In the final sections, we consider related work in the field, and discuss future work.