Joint Virtual Reality Conference of EuroVR - EGVE - VEC (2010)
T. Kuhlen, S. Coquillart, and V. Interrante (Editors)

Short Paper: Engineering Realtime Interactive Systems: Coupling & Cohesion of Architecture Mechanisms

Marc Erich Latoschik[1] and Henrik Tramberend[2]

[1]Intelligent Graphics Group, Bayreuth University, Germany
[2]Beuth University of Applied Sciences, Berlin, Germany

Abstract

This paper reviews coupling and cohesion as software quality criteria for the development of Realtime Interactive Systems (RIS). The applicability of these criteria to evaluate RIS architecture mechanisms is examined while the utilization of existing software metrics is discussed. Three commonly found mechanisms, scene graphs, event systems and entity models, are evaluated with respect to a minimization of coupling and a maximization of cohesion. The paper motivates an analytical approach to the evaluation of software techniques as well as a strengthening of software technology aspects in the field of interactive simulations in general given current challenges of diversification, parallelization, and interconnection.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality D.2.11 [Software Engineering]: Software Architectures—Domain-specific architectures, Patterns

1. Introduction

Software technology is of central importance for application development in the areas of Virtual, Augmented, and Mixed Reality (VR/AR/MR). Hardware capabilities and software requirements keep changing while expectations for applications rise continuously. Current technological challenges are hard- and software diversification, parallelization on different levels, and interconnection and networking. At the same the, virtual worlds are expected to be rich and lively, intelligent, and highly interactive.

To make matters worse, research institutions are often characterized by a high degree of staff fluctuation. A new generation of developers will need a long time to arrive at the state-of-the-art and to be able to advance the field. Changes in the research staff often lead to abandoned software systems that are replaced too often by ad-hoc and inferior re-implementations. Reproduction of scientific results becomes extremely difficult–if not impossible. Scientific progress will be increasingly restrained without a solid and lasting technological foundation.

Architecture mechanisms and software techniques currently being used often appear to be outdated or poorly adapted and implemented. There is a lack of a methodological and comprehensive analysis and comparison of common mechanisms and techniques as well as a lack of common terminology customized to this specific field. As a result, engineering knowledge essential for future research work can not be communicated to the next generations and idiosyncratic solutions are created.

Idiosyncrasy has a serious negative impact on the comprehensibility of software systems. Re-implementations are favored where maintenance and evolution of existing solutions would be the better option. Yet, these re-implementations often suffer from the very same technical deficiencies and, at the same time, are a significant drain on resources that could better be used for the primary research objectives.

Remarkably little attention has been paid to RIS software technology for years. There has been a conspicuous lack of relevant expert articles, so-called "system papers", and the quality of the articles that are published varies. This critical development has been noticed and several initiatives have been started as a consequence [BCW08, LRB*10, SSLR10].

2. Coupling and cohesion

Abstraction and modularization play an important role with respect to software quality criteria like functionality, usability, reliability, performance/portability, and supportability (FURPS+, [Gra92]). A modular architecture design should **minimize coupling** and **maximize cohesion** defining

coupling as the measure of the independence of relations between functional units and

cohesion as the measure of the semantic nature of relations between components of a functional unit.

The application of these quality measures requires some scale, ideally some quantitative metrics for objective evaluations. Multiple metrics do exist [Gra92, LM06, Jon08] which are based on the (automatic) analysis of source code, hence on the analysis of static compile-time interdependences as exposed by syntactic elements (function calls, parameters, inheritance,...).

The application of such metrics during RIS development is inherently difficult. Strong data and process dependencies do exist, from 6DOF (degree of freedom) data or object shape approximations in different modules, to complex interdependent execution schemes and process flows in order to simulate consistent worlds. Ideally, simulators would be build around simulation properties which in turn would be implemented by dedicated exchangeable and extensible software components. Just how distant RIS technology is from this goal is made clear by the mind experiment illustrated in Figure 1.

Figure 1: *A mind experiment: Imagine a virtual agent in a virtual world. A user is coupled to this virtual world through different input and output channels. Agent and user can perceive and act and freely interact with each other and the environment. The challenge is the following: to transfer the agent to an alternative virtual environment while maintaining all his properties.*

Even if such an abstract description layer for a simulation would exist, the interdependences in such systems are largely of a semantic nature and hence in general inaccessible to syntax based analysis tools. They may be exposed in the syntactic structure, but most compile time interdependences can be hidden by replacing them with runtime checks, e.g., using runtime-type information. This problem is noticeable with respect to coupling, it is obvious with respect to cohesion since its definition already relates this property to the semantics of relations. In addition, an evaluation based on a specific code base will evaluate exactly that systems while we are interested in the evaluation of specific architecture mechanisms in general.

As a result, a qualitative evaluation will be necessary to evaluate given RIS architecture mechanisms. We will now analyze three common mechanisms with respect to the coupling and cohesion criteria. Exemplary discussions will point out the pros and cons of each mechanism to hint in a direction of a potential discussion methodology and terminology.

3. Scene graphs

Scene graphs are an architectural mechanism present in many RIS platforms. Potential uses and applications, however, have been subject to long, intense and divergent discussions [BBC*99]. The problem is that in terms of an adequate representation of an architectural mechanism intended to respond to a specific problem, many scene graph systems are deficient. They add additional functionality and data to the scene graph (in some cases in extensive numbers), which compromises cohesion and contributes to the coupling effect.

Figure 2 shows a typical example of such an augmented scene graph. As a simple data structure, its main objective is to describe spatial relationships between geometries in the scene. The evaluation of a scene graph, i.e. rendering, is performed by recusive traversal functions. Additional information, like interaction or animation nodes, is not only ignored by these traversals, but establishes irrelevant couplings to other subsystems.

And yet, complete applications are built around extended scene graph mechanisms. These applications add additional node types for application-specific tasks, in accordance with the object-oriented paradigm. In general, inheritance principles are utilized in the implementations. Such a design is characterized by strong coupling and weak cohesion.

In terms of the need for greater diversification, parallelization and networking, this design has many disadvantages. For example, the inheritance strategy complicates replacement of the underlying graphics renderer. Sometimes object oriented software patterns, like the visitor pattern, are used to decouple the traversal functions from the data representation. However, the visitor pattern is often regarded as a crutch for a deficient programming language and, in this case, solves only half of the coupling problem.

Parallel evaluation can only be realized here with difficulty. A general temporal dependency exists for scene graph traversals. Camera, light, and render traversals must be ap-

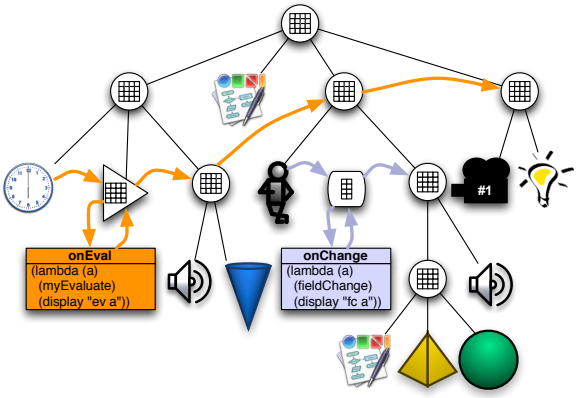

Figure 2: *Typical example of an augmented scene and application graph with inner nodes for transformation and grouping and leaf nodes for object geometries (shapes), material properties, light sources and cameras, but also for sound, animation and interaction nodes. The event system constitutes an additional value propagation layer independent from the transformation hierarchy.*

plied sequential or will introduce latencies in a pipelined architecture using state copies. Dividing the graph into partial graphs to be concurrently processed is obstructed by the horizontal dependency potentially caused by functional semantics of special node types, e.g., material nodes, in the traversal strategy. This is a consequence of the *statefulness*. Networking is only possible using shared state.

Hence, scene graph centered designs are suboptimal and fall especially short in relation to decoupling, and a maximized cohesion.

4. Event system and routing

Event systems are a popular RIS architecture mechanism providing inter- and intra-plattform communication as well as an execution scheme. Event systems have different levels of flexibility. On one end of the spectrum are systems that are hard-wired by the platform's architecture providing a well-defined but inflexible execution scheme. On the other end are extensible systems in which different schemes can be implemented and used. Figure 2 shows the inflexibly wired event logic in a field-routing system. Here, state modifications are monitored by guarding value changes and passing them to potential receivers. Application behavior is implemented by registering user-defined routines as call-back functions. The dispatch order of these functions is hard-wired.

One critical aspect of event-based programming is its tendency to reverse the control flow [HO06]. The classic procedure is the following: instead of, for example, reading user inputs via the selection of blocking operations, the event handlers are distributed over the software and installed in places determined by the architecture. Their implementation is delegated to other locations in the source code. Two problems result: 1) fragmentation of the interactive logic of the program and 2) a coupling of the execution model to the shared state access.

The fragmentation of the interactive logic generally weakens cohesion. This fragmentation is typically accompanied, however, by object-oriented modeling. Here the event receivers are the objects of the simulation (e.g., as entities, see below). Since these can be regarded as semantic units, it is imperative that object functionality within the architectural design be anchored to them, thus strengthening cohesion. Technically event systems lend themselves very well to decoupling. Conceptually this is not so obvious.

An event system with a fixed execution scheme tends to increase coupling, since it defines process dependencies. In contrast, a loose event model with few dependencies decreases coupling, while an anchoring of the functionality to the objects of the simulation tends to increase coupling. This too can be counteracted with a skillful selection of software design patterns.

An event system can adequately satisfy the need for diversification. This ability depends on the selection and availability of the programming language as well as suitable utility libraries.

Event systems can easily be networked by relegating the transport layer of the events to an inter-platform mechanism. If one reduces the inflexible wiring and data dependencies, the evaluation of event systems can easily be parallelized. And yet, inflexibly wired systems with a pre-determined execution scheme and inherent data dependencies lead to parallelization primarily on the level of only the event-handlers. This requires a fine grained parallelization method. System threads are too corse and require frequent context switches that are expensive, while the concurrent tasks are often small in terms of computational complexity. For this reason, so-called micro threads have been discussed and utilized for many years in computer game development. But their utilization requires separate custom-built scheduling systems.

5. Entity model

Steed summarizes many years of experiences made during the development of VR/AR applications. For the plethora of projects carried out at his institution, a considerable number of RIS tools and platforms have been tried out. He identifies one single architectural mechanism that has been reused again and again: an entity model as an abstract data model [Ste08].

An entity represents the smallest semantic unit of a RIS application. Entities link together different functional properties which are maintained by various modules. This approach can be seamlessly integrated into aspect-oriented design techniques. An entity model offers a level on which the

world can be described. Figure 3 shows the use of such a central entity model in a typical RIS design.

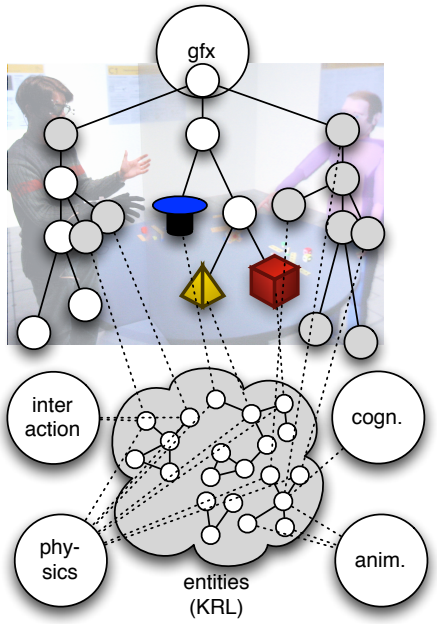

Figure 3: *Abstract entity model on a RIS platform. A central entity model based on a knowledge representation layer combines different elements, such as graphic representation, interaction, animation, cognition or physical behaviors. These elements are carried out by different components depending on the modular design. The detailed illustration shows the connection between a scene graph component with a representation of the user, the virtual agent and various scene objects [LBW05].*

An analysis of cohesion and coupling shows strong analogies to the statements made about the event system. This is due to the fundamental similarities between the object-oriented design and entity model. At the same time an entity model raises the discussion to an abstract level of aspects. This abstraction is precisely the decoupling of functionality from the object model and the transfer of this functionality to the domain of specialized function units. As such it leads to both greater decoupling and stronger cohesion.

If the right description language is used, an entity model is quite tolerant to diversification. The question as to which language and architectural mechanism are most suitable for the entity layer leads us into the field of model driven software design (MDD). This area is of importance for current research, and the usefulness of a wide range of architectural mechanisms must be tested. Some work now favors a knowledge representation layer with an ontological link between the entities and their aspects by means of a semantic representation (see Figure 3).

6. Conclusion

There is a need for an analytical evaluation of RIS related software technology required for VR/AR and MR. Future research work calls for a solid technological foundation that cannot just consist of a amalgamation of idiosyncratic software solutions. To set out into new territory, existing mechanisms and techniques have to be questioned and re-evaluated. This endeavor should be flanked by a common methodology and terminology. A RIS software technology body of knowledge should identify possible solutions for re-curring problems and identify their applicability, strengths, and weaknesses. This would be the basis for innovative approaches which answer current and future challenges.

Acknowledgements: Supported by the BMBF Germany, program *Ingenieur*Nachwuchs, project SIRIS (#17N4409).

References

[BBC*99] BETHEL W., BASS C., CLAY S. R., HOOK B., JONES M. T., SOWIZRAL H., VAN DAM A.: Scene graph apis: wired or tired? In *SIGGRAPH '99: ACM SIGGRAPH 99 Conference abstracts and applications* (New York, NY, USA, 1999), ACM Press, pp. 136–138. 2

[BCW08] BRUNNETT G., COQUILLART S., WELCH G.: 08231 abstracts collection – virtual realities. In *Virtual Realities* (Dagstuhl, Germany, 2008), Brunnett G., Coquillart S., Welch G., (Eds.), no. 08231 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. 1

[Gra92] GRADY R.: *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, 1992. 2

[HO06] HALLER P., ODERSKY M.: Event-based programming without inversion of control. In *Proc. Joint Modular Languages Conference* (2006), Springer LNCS, pp. 4–22. 3

[Jon08] JONES C.: *Applied Software Measurement: Global Analysis of Productivity and Quality*, 3. ed. McGraw-Hill Osborne Media, 2008. 2

[LBW05] LATOSCHIK M. E., BIERMANN P., WACHSMUTH I.: Knowledge in the loop: Semantics representation for multimodal simulative environments. In *Proceedings of the 5th International Symposium on Smart Graphics 2005* (Frauenwoerth Cloister, near Munich, Germany, 2005), pp. 25–39. 4

[LM06] LANZA M., MARINESCU R.: *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006. 2

[LRB*10] LATOSCHIK M. E., REINERS D., BLACH R., FIGUEROA P., DACHSELT R. (Eds.):. *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), Proceedings of the annual IEEE Virtual Reality 2008–2010 workshops* (2008–2010), Shaker Verlag. 1

[SSLR10] SLATER M., STEED A., LATOSCHIK M. E., REINERS D. (Eds.): *Reflections on the Design and Implementation of Virtual Environment Systems*, vol. 19 of *PRESENCE journal special issue*. MIT Press, 2010. 1

[Ste08] STEED A.: Some useful abstractions for re-usable virtual environment platforms. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), proceedings of the IEEE Virtual Reality 2008 workshop* (2008), Latoschik M. E., Reiners D., Blach R., Figueroa P., Dachselt R., (Eds.), Shaker Verlag, pp. 33–36. 3