Simulator X: A Scalable and Concurrent Architecture for Intelligent Realtime Interactive Systems

Marc Erich Latoschik*
Intelligent Graphics Group, University of Bayreuth, Germany

Henrik Tramberend†
Beuth University of Applied Sciences Berlin, Germany

Abstract

This article presents a platform for software technology research in the area of intelligent Realtime Interactive Systems. Simulator X is targeted at Virtual Reality, Augmented Reality, Mixed Reality, and computer games. It provides a foundation and testbed for a variety of different application models. The current research architecture is based on the actor model to support fine grained concurrency and parallelism. Its design follows the *minimize coupling and maximize cohesion* software engineering principle. A distributed world state and execution scheme is combined with an object-centered world view based on an entity model. Entities conceptually aggregate properties internally represented by state variables. An asynchronous event mechanism allows intra- and interprocess communication between the simulation actors. An extensible world interface uses an ontology-based semantic annotation layer to provide a coherent world view of the resulting distributed world state and execution scheme to application developers. The world interface greatly simplifies configurability and the semantic layer provides a solid foundation for the integration of different Artificial Intelligence components. The current architecture is implemented in Scala using the Java virtual machine. This choice additionally fosters low-level scalability, portability, and reusability.

Index Terms: 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 real-time interactive systems (RIS), i.e., for VR, AR, MR, and computer games. Current technological challenges are hard- and software diversification as well as concurrency, from small scale parallelization to networking. Simulator X is an evolutionary platform for software technology research in the area of intelligent RIS. The current architecture (see Figure 1) incorporates some well known abstractions and good practices as well as some interesting experimental features:

- Unified concurrency model based on actors [14].
- Shared entity model using a distributed state.
- Component architecture using the concurrency model.
- Functional/OOP approach based on Scala.
- Semantic binding for the integration of AI.
- Generic support for multimodal interfaces.

*marc.latoschik@uni-bayreuth.de
†tramberend@beuth-hochschule.de

This paper concentrates on the first four aspects. After this introduction, we will first discuss related work. The following sections match the Simulator X architecture bottom to top, hence we start with the description of the core system followed by a description of the world interface. Each section includes discussions and reflections of lessons learned which will be summarized in the concluding section alongside some remarks regarding future work.

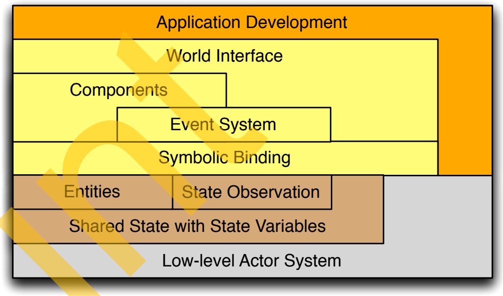

Figure 1: General architecture of the Simulator X platform. The core system is based on the actor model to realize a distributed state, state observation, and entity model. A semantic layer connects the core system with a world interface which provides a high-level event and component & configuration system as well as a general semantic access scheme.

2 Related Work

A RIS middleware provides a software frame for recurrently required simulation tasks ranging from tracking, input processing, collision detection, physics simulation, Artificial Intelligence (AI), graphics, audio, and haptics rendering, to networking etc. The utilized coupling method is critical with respect to reusability and scalability [10, 25, 18]: Dedicated libraries for certain simulation tasks often require proprietary (1) data representations and (2) execution schemes. For example, a close coupling to scene graph libraries like OpenGL Performer^{TM}, Open Inventor, Open Scene Graph, OpenSG [24], or X3D is a source of several drawbacks [6, 2].

Data flow networks are a typical and well established RIS execution scheme, see, e.g., [7, 28, 1, 11, 9, 16]. They are simple to understand but they are also restrictive in their expressiveness [9]. Hybrid execution schemes which additionally incorporate, e.g., procedural elements [9], asynchronous events [16], or messages may cause severe consistency problems [9]. Still, a component-based or even service-oriented approach [5, 21] based on an event system provides reasonable decoupling and usually scales well in terms of networking [8, 13].

Execution scheme and data management are often strongly connected. Here, entity like models have proven to be a highly advantageous, reusable, and scalable representation [22, 16, 27]. Mapping of entities to objects in object-oriented programming (OOP) is straight forward [12] but often links data representation to a given execution scheme using the objects' method interfaces. Simulator X conceptually views entities only as sets of loosely associated properties, each having possibly independent execution schemes. The general idea is similar to the remote distributed object semantics found in Repo 3D [20]. However, the utilized actor model [14] of Simulator X strongly decouples data representation from the graphics representation and from given execution schemes. Simulator X actors are independent, asynchronous, and communicating execution threads processing on a conceptual unified entity model on top of a distributed state representation. They provide scalable concurrency on multiple levels of granularity. This is a different actor concept as in [3] where it is a collection of properties whose changes are distributed by a data flow system or in [12, 13], where it denotes special types of active objects.

Entity models in Intelligent Virtual Environments [4] (IVEs) require a semantic representation of scene content as the basis for several AI methods, see, e.g., [26, 23, 15, 19]. Typical examples include multimodal interactions as well as the simulation of human like agents. A seamless integration of semantics into a simulator's data representation still is an open topic of research. Simulator X provides a semantic binding for all entities and their properties using the concept of a world interface.

3 Actor Model and State

Following the actor model, a Simulator X program consists of a set of co-operating actors:

- Each actor represents a single thread of control. The level of parallelism depends on the actor implementation and hardware.
- Actors communicate exclusively via asynchronous message passing.
- Storage of program state is managed locally by each actor. There is no globally accessible shared state.
- Actors are light-weight, and new actors can be spawned at any time.

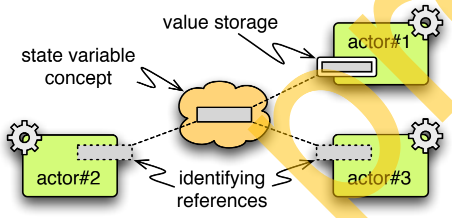

Figure 2: A *state variable* is referenced by three actors. Conceptually, all three actors see the state variable as a piece of globally accessible, shared world state. Technically, only actor#1, the *owner*, locally stores the value of the variable and manages access to it. The other actors carry opaque references to the variable and need to communicate with the owner to access its value.

Many algorithms that can conveniently be formulated using established shared state concurrency models need to extensively be reformulated for the actor model. Communication via shared state is an accepted programming model for concurrent VE applications. Many frameworks use it as a basis for higher-level programming abstractions like scene graphs or entities, and application developers have grown familiar to use it. Therefore, switching to a message based programming model may at first lead to a regression in productivity, or prevent adoption of the new model altogether.

To address the issues that arise from the inability of using shared state for communication, the Simulator X architecture provides *state variables* that create the illusion of consistently shared mutable state for the application programmer, but are implemented using message passing. Figure 2 illustrates a state variable being referenced by three actors.

State variables represent uniquely identifiable mutable global variables that store immutable values. They are created and owned by one actor, but are potentially visible and accessible to all actors in the system. The state variable's value is locally stored only at the owning actor who can access and modify it directly. The owner handles all access to an owned state variable from requesting other actors.

3.1 Actors and Entities

Simulator X defines an entity as a collection of properties that describe an application object. A property associates a state variable with a *symbol* that denotes it and that grounds it into an application specific semantic representation of that property. Figure 3 shows an entity with two named properties. The property values are stored in state variables that are managed by two different actors.

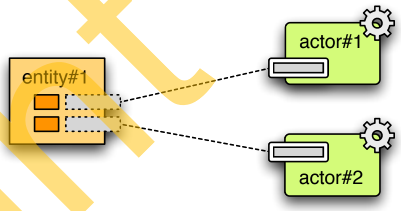

Figure 3: Entity#1 is a collection of two properties. Each property associates a state variable with a symbol that serves as an anchor for semantic information. The state variables are owned and controlled by two different actors.

This representation of entities has two interesting characteristics that differentiate our data model from the classic object-oriented data model:

- Storage control and access of property values is decoupled. Entities merely collect references to state variables and delegate value management to the actors that control the respective state variables.
- Entity model and definition of behavior and execution scheme is decoupled. Entities are just (references to) data. The behavior of an entity as a whole is determined by potentially multiple independent and concurrent actors observing its properties and/or performing value changes.

Entities can be seen as an aggregation of different *aspects* that describe an application object. An aspect here is a combination of a set of properties and the definition of a behavior that operates on the properties. Each aspect is provided and controlled by a specific actor.

Most interestingly, this model does not specify the cardinality of the resulting association between entity and actor. The aspects of one entity may simultaneously be controlled by several actors, while at the same time one actor may provide an aspect of any number or groups of entities.

4 World Interface

The world interface realizes an event system and component model based on a symbol registration and lookup service built on top of the distributed state, state observation, entity, and actor model. It hides technical details of the core system, e.g., internal data representations or execution schemes and interplay of processing components, from application programmers.

The world interface provides access to *relevant* aspects of the simulator's internal operation as well as to the distributed world state. Developers are free to define what they consider relevant. That is, in contrast to a fixed and given set of events and state variables as, e.g., found in many game engines, the world interface provides an intermediate layer. Custom execution schemes and logic components are developed on top of the core system to provide application developers with tailored application models. Four main operations are central for Simulator X's world interface:

Notification of events of relevant state changes.

Execution of actions changing relevant states.

Processing of state queries.

Configuration of relevant events and permitted actions.

On the lowest level, the world interface provides an interface to establish a symbolic binding for architecture elements (see Figure 1). It realizes a loosely coupled symbol-based registration and lookup service for state variables, entities, and actors. The symbolic binding eliminates the necessity to use direct references. This decoupling of concept from implementation is highly advantageous for symbolic AI methods and AI-oriented development styles where human readability is of central importance, e.g., for the development and maintenance of data- and knowledge bases.

4.1 Event System

Events communicate *meaningful* changes in and of the simulation. The world interface realizes an extensible event system. Developers are free to add or remove events as appropriate for a given application context or system configuration. The event system directly utilizes the message passing facilities of the core system. Two types of events are initially provided:

Generic events notify about any meaningful changes not necessarily reflected by the world state directly.

Value-change events specifically notify about value changes of a state variable.

In case of generic events, the world interface is responsible for initial handshaking between event sender and event receiver. Possible event senders register themselves at the world interface as providers of given events. Possible event receivers tell the world interface that they require the events. The world interface compares provided and required events and mutually informs both parties about each other in case of a match. After this initial handshaking, the world interface is removed as a middleman and event routing is peer-to-peer.

Value-change events complement the *change notification* access method of a state variable with an interface to register user defined callback handlers. They are of central importance for the current architecture.

The notification message sent by the core system consists of a reference to the state variable and of the new value itself. As described in section 3, state variables are always managed by the one specific actor who owns a given state variable while they may be referenced by an arbitrary number of different actors and entities (see Figure 2).

4.2 Components

Components are Simulator X's coarse grained application building blocks. Each component realizes a specific functionality and adds this *aspect* to user defined target entities. Simulator X currently includes components for graphics rendering, physical simulation, rule propagation and behavior (for Artificial Intelligence methods), as well as for 3D selection and interaction devices, and multimodal interaction (speech and gesture).

A component only requires a thin API layer on top of its internal implementation. The actor paradigm already conveys consolidation of related algorithms and data structures in independent functional units to foster high cohesion. The core system's state variable access schemes as well as the world interface's event system support decoupling between components to a large extend.

An architecture of distributed functionality given an object centered entity model raises questions about the initial setup of the combination of entities and components. A component will inevitably require access to properties relevant to the component's aspect, either to read out or to write new values. That is, components provide aspects for an arbitrary (and user-defined) subset of entities. In addition, each aspect requires access only to a relevant subset of an entity's properties.

Listing 1: *Definition of two entity aspects as realized by a graphics renderer component and a physics component.*

```
1  class GhostDesciption{
2    private val graphicsParam =
3      ShapeFromFile("models/agents/ghost.dae")
4    private val physicsParam =
5      Sphere(1f, ConstVec3f(0f,0f,0f))

7    val desc = new EntityDescription(
8      Aspect('renderer, graphicsParam,
9             Ontology.transform
10            requiredAs JVR.transform
11            using JVR.transform_converter),
12     Aspect('physics, physicsParam,
13            JBullet.transform
14            providedAs Ontology.transform
15            using JBullet.transform_converter)
16     )
17 }
```

Listing 1 illustrates the definition of relevant aspects for a given entity type, here for ghost-entities required in a virtual ghost hunting game.

To successfully match different data representations of properties in the various components, the semantic layer of Simulator X provides an ontology of possible property types and data representations while the components have to provide matching converter methods. The last parameter of the `Aspect` constructor (see lines 9–11 and 13–15 of Listing 1) defines a mapping between different data representations of entity properties as given by individual components. In detail, lines 13–15 define that the transformation value created by the physics component will be provided as a `transform` property and converted to the internal data representation using the `JBullet.transform_converter`. Similar, lines 9–11 define that the graphics component requires the `transform` property and that it will be converted using the `JVR.transform_converter`.

5 Conclusion

Simulator X's current architecture already provides a unified and scalable concurrent programming paradigm portable to all major desktop and many mobile computing platforms. The use of the Scala language incorporates modern functional language constructs while Scala's object oriented concepts have been highly beneficial during the developers' learning phase.

The message passing concurrency used by the actor model is augmented with an extensible entity model that creates the illusion of a globally shared world state, which has proven to be useful and easy to grasp even for unexperienced developers. Applications are built from loosely coupled dedicated components. Component integration only requires a thin layer which couples components to the world state. The system is highly configurable due to its reduced mutual dependency of components.

The world interface provides a consistent API layer with an extensible event system, component configuration, and entity composition. All relevant world interface elements have a symbolic binding that links to additional semantic information. The symbolic binding greatly simplifies world interface usage. In addition, it provides the necessary basis to incorporate AI methods as required for intelligent RIS applications.

Several demo applications were developed to demonstrate the usefulness and adequacy of the architecture and to provide some coarse high-level performance cues. The applications already combine physics simulation, high quality rendering, behavior simulation as well as multimodal (gesture and speech) interactions in game scenarios using immersive displays.

5.1 Future work

Future work will explore the usefulness of the current system's design in multiple RIS-related application areas. The goal is to collect more experiences for further technical refinements and alternatives. This endeavor will go hand-in-hand with the development of new components. A major task is the development of application models based on an integral AI-layer.

Probably one of the most interesting result of the Simulator X experiment will be the assessment wether the benefits that come with the all-out adoption of the actor model and the Scala language are worth the trouble to adopt to a completely novel development paradigm and language. So far, the results are promising but detailed evaluations have to follow.

Acknowledgements

Supported by the German Federal Ministry of Education and Research, program *Ingenieur*Nachwuchs, project SIRIS (#17N4409)

References

[1] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert. FlowVR: a Middleware for Large Scale Virtual Reality Applications. In *Proceedings of Euro-par 2004*, Pisa, Italia, August 2004.

[2] R. Arnaud and M. T. Jones. Innovative software architecture for real-time image generation. In *Proceedings of the I/ITSEC Conference*, 1999.

[3] I. Assenmacher and B. Raffin. Short paper: A modular framework for distributed vr interactionprocessing. In *Proceedings of the Joint Virtual Reality Conference of EGVE - ICAT - EuroVR*, 2009.

[4] R. Aylett and M. Luck. Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments. *Applied Artificial Intelligence*, 14(1):3–32, 2000.

[5] M. Bauer, B. Bruegge, G. Klinker, A. Macwilliams, T. Reicher, S. Riß, C. S, and M. Wagner. Design of a component-based augmented reality framework. pages 45–54, 2001.

[6] W. Bethel, C. Bass, S. R. Clay, B. Hook, M. T. Jones, H. Sowizral, and A. van Dam. Scene graph APIs: wired or tired? In *SIGGRAPH '99: ACM SIGGRAPH 99 Conference abstracts and applications*, pages 136–138, New York, NY, USA, 1999. ACM Press.

[7] R. Blach, J. Landauer, A. Rösch, and A. Simon. A Highly Flexible Virtual Reality System. In *Future Generation Computer Systems Special Issue on Virtual Environments*, volume 14, pages 167–178. Elsevier Amsterdam, 1998.

[8] D. Brown, S. Julier, Y. Baillot, and M. A. Livingston. An event-based data distribution mechanism for collaborative mobile augmented reality and virtual environments. In *Proceedings of the IEEE Virtual Reality 2003*, VR '03, pages 23–, Washington, DC, USA, 2003. IEEE Computer Society.

[9] M. Bues, T. Gleue, and R. Blach. Lightning: Dataflow in motion. In Latoschik et al. [17], pages 7–11.

[10] G. de Haan, M. Koutek, and F. H. Post. Flexible abstraction layers for VR application development. *Virtual Reality Conference, IEEE*, 0:239–242, 2007.

[11] P. Figureoa. InTml: Main Concepts, Examples. and Initial Lessons. In Latoschik et al. [17], pages 3–6.

[12] E. Frécon and M. Stenius. DIVE: a scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91–100, 1998.

[13] P. Fuhrer and J. Pasquier-Rocha. Madviworld: A software framework for applying a collaborative virtual world paradigm to the internet. *Journal of Systemics, Cybernetics and Informatics*, 4(2):34–45, 2006.

[14] C. Hewitt, P. Bishop, and R. Steiger. A universal modular ACTOR formalism for artificial intelligence. In *IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[15] E. Kalogerakis, S. Christodoulakis, and N. Moumoutzis. Coupling ontologies with graphics content for knowledge driven visualization. In *Proceedings of the IEEE VR2006*, pages 43–50, 2006.

[16] X. Larrodé, B. Chanclou, L. Aguerreche, and B. Arnaldi. Open MASK: an Open-Source Plaform for Virtual Reality. In Latoschik et al. [17].

[17] M. E. Latoschik, D. Reiners, R. Blach, P. Figueroa, and R. Dachselt, editors. *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), proceedings of the IEEE Virtual Reality 2008 workshop*. Shaker Verlag, 2008.

[18] M. E. Latoschik and H. Tramberend. Engineering Realtime Interactive Systems: Coupling & Cohesion of Architecture Mechanisms. In T. Kuhlen, S. Coquillart, and V. Interrante, editors, *Proceedings of the Joint Virtual Reality Conference of Euro VR – EGVE – VEC*, EG Symposium Proceedings, pages 25–28, 2010.

[19] J.-L. Lugrin and M. Cavazza. Making Sense of Virtual Environments: Action Representation, Grounding and Common Sense. In *Proceedings of the Intelligent User Interfaces IUI'07*, 2007.

[20] B. Macintyre and S. Feiner. A distributed 3d graphics library. In *In Computer Graphics (Proc. ACM SIGGRAPH '98), Annual Conference Series*, pages 361–370, 1998.

[21] A. MacWilliams, C. Sandor, M. Wagner, M. Bauer, G. Klinker, and B. Bruegge. Herding sheep: Live system development for distributed augmented reality. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, pages 123–, Washington, DC, USA, 2003. IEEE Computer Society.

[22] F. Mannuß, A. Hinkenjann, and J. Maiero. From scene graph centered to entity centered virtual environments. In Latoschik et al. [17], pages 37–40.

[23] S. Peters and H. Shrobe. Using semantic networks for knowledge representation in an intelligent environment. In *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*, Ft. Worth, TX, USA, March 2003. IEEE.

[24] D. Reiners, G. Voß, and J. Behr. OpenSG: Basic Concepts. www.opensg.org/OpenSGPLUS/symposium/-Papers2002/Reiners_Basics.pdf, february 2002.

[25] F. Rodrigues, R. Ferraz, M. Cabral, F. Teubl, O. Belloc, M. Kondo, M. Zuffo, and R. Lopes. Coupling virtual reality open source software using message oriented middleware. pages 17–24. Shaker Verlag, 2009.

[26] M. Soto and S. Allongue. Modeling methods for reusable and interoperable virtual entities in multimedia virtual worlds. *Multimedia Tools Appl.*, 16(1-2):161–177, 2002.

[27] A. Steed. Some useful abstractions for re-usable virtual environment platforms. In Latoschik et al. [17], pages 33–36.

[28] H. Tramberend. Avocado: A distributed virtual reality framework. In *Proceedings of the 1999 IEEE Conference on Virtual Reality (VR-99)*, Los Alamito, CA, 1999.