Integrating Functional Reactive Programming in a High-Level VR Framework

Kristopher J. Blom Steffi Beckhaus

interactive media / virtual environments
University of Hamburg, Germany

Abstract: The recently developed Functional Reactive Programming paradigm provides a new potential method for the creation of dynamic, interactive Virtual Environments. Complex simulation environments are modeled as a combination of continuous time functions and discrete events. In this paper, we present a method to introduce the Functional Reactive Programming paradigm into higher level VR system, on hand the AVANGO VR system. The implementation highlights how the various mechanisms of the systems can be arranged to complement each other and work together. In order to allow the VE author to seamlessly incorporate the values into the AVANGO data-flow system, without low-level programming, a new reflective Field mechanism is developed to facilitate the use of the run-time specified external system.

Keywords: Virtual Reality, Functional Reactive Programming, Interactive, Dynamic Virtual Environments

1 Introduction

The creation of interactive, dynamic Virtual Environments remains an often elusive goal of Virtual Reality(VR). Even today, interaction in Virtual Environments (VE) often remains restricted to the users movement through the presented environment. This movement is also often the only dynamic component of the environment. Though there are no technical reasons that such environments cannot be created, as witnessed by modern computer games, there are many reasons for this in VR. The main reason is a lack of resources, specifically time and money. Contributing to this are the limited tools for the creation of such environments.

VR systems provide differing levels of help for implementing interactive, dynamic environments. The popular basic VR systems, such as VR Juggler, provide no help beyond hardware interfaces and access to a Scene Graph [BJ98]. Any dynamics or interaction in the environment have to be programmed, with help of the SG, by the application creator, typically in a low-level language. More advanced VR systems provide a variety of possibilities, most of which are designed similarly to either OpenInventor or VRML 2.0. These systems are based on the premise of incorporating a data-flow system orthogonally to the Scene Graph. The interaction and dynamics of a system are implemented within the data-flow graph. Typically, the new dynamics functionality is created by extending nodes with

the C++ language and then using the data-flow system for delivery of values to the proper places. An example of this type of system is the AVANGO VR system [Tra99].

Various attempts have been undertaken to further improve the VE authors' ability to program dynamic and interactive environments. In order to control the timing and flow of a dynamic environment, Cremer et al. proposed the usage of special Hierarchical Concurrent State Machines in [CKP95]. Similar systems can be found is the Interactive Storytelling area [BLM+04]. These system address the issue of high-level behavioral changes and interaction, but do not address how to actually program the dynamics. In order to support the creation of dynamics at a low level, the use of constraint networks has been proposed [Del00, TLG99]. The constraint networks approach, however, allows limited runtime versatility to the developed system, has difficulty handling interaction, and has problems scaling beyond small simulation environments.

The recently developed paradigm of Functional Reactive Programming (FRP) provides a new approach to incorporate both interaction and dynamics. The FRP paradigm defines the simulated environment in terms of continuous-time functions and discrete events. The dynamics of the system are naturally modeled by continuous time functions. Interaction occurs through manipulation of incoming values and through the Events. The event structure is primarily used to switch between running behaviors. This hybrid system model of simulation matches well to the structure of interactive, dynamic VEs.

In [BB07], it was shown how Functional Reactive Programming could be incorporated into a VR system; in that work a current FRP systems, implemented in the functional language Haskell, was embedded into VR Juggler. This was the first usage of the FRP system outside of the Haskell community and the first time it was coupled with an external system. The resulting mixed system structure was dubbed Functional Reactive Virtual Reality. A framework for FRP's usage was provided, including controlling the simulation loop, but the incorporation the values of the simulated FRP environment into the SG system was left to the users, as is typical in such basic VR system.

In this paper, we present an implementation integrating FRP into a higher-level VR system, AVANGO. This implementation shows how FRP can be used to extend a higher-level system's functionality in a complimentary way. Incorporated into a higher-level system, FRP can be used to implement the dynamics and even interaction of an environment, without the need of programming derived C++ nodes. In this implementation, the standard AVANGO dataflow mechanisms are used to deliver the FRP produced values to the appropriate places, removing the need for the user to program in a low-level language. In order to achieve this seamless use of FRVR in AVANGO, a new addition to AVANGO, which enables AVANGO to handle a form of run-time established Fields, is also presented. Using this special connection mechanism, the author can establish Field Connections using named references, as though it was a normal Field.

In the following section, a brief explanation of Functional Reactive Virtual Reality is given, along with a high-level overview of the AVANGO system. In Section 3 the im-

plementation of FRVR in AVANGO is presented. Section 4 introduces the new reflective Field/FieldConnection mechanism. Section 5 concludes this paper and provides some future avenues for research in this area.

2 Background

The Functional Reactive Virtual Reality (FRVR) system introduced in [BB07] provides a first implementation to leverage the recently developed programming paradigm, Functional Reactive Programming. The FRP paradigm implements systems as a combination of continuous time functions and discrete events [CNP03]. Most of the FRP implementations are written in a lazy evaluation, pure functional language, Haskell.

In FRVR, the dynamics of a VE are implemented with FRP's special functions, *Signal Functions*. These Signal Functions (SF) are based upon an explicit, first rate notion of time. The basic functionalities provided by FRP build upon on SFs, such as integrate and derive, to build teh system's behaviors. Additionally, the FRP paradigm is based on a reactive nature. A special *Event* concept enables the system to be reactive and, thereby, interactive. Event occurrence cause the system to switch between different SFs. Events can come from either external sources along with any other input or be generated internally. Internal events include all of the timing events, such as triggering events at a specific time or after so long, and any function can trigger events conditions. An array of event producing SFs produce these higher-level time based reactive functionalities. The complete description of the FRP paradigm can be found in [Cou04, CNP03].

The FRVR system leverages FRP's natural handling of time to implement the dynamic portions of VEs. The FRP portion of the system is used to control the movement of dynamics. In contrast to using simple animations, the FRP system allows the dynamics to be made interactive. Interaction with an object can be achieved reactively through a simple change of behavior, triggered through an Event, for instance selection of a moving object.

In the previously presented implementation, FRVR was linked into a basic VR system, VR Juggler. This paper looks at the inclusion of FRP in a VR system that already includes a higher level support, AVANGO [Tra03, Tra99]. As mentioned in the introduction, AVANGO's design, and that of many higher level VR systems, follows that of OpenInventor [WG93]. AVANGO implements this, by building a data-flow layer on top of the OpenGL Performer Scene Graph. AVANGO provides two additional components beyond the data-flow system, a distribution system for networked virtual environments and a scripting layer.

AVANGO's distribution system is built directly into the Performer derived nodes. The scripting layer, built with the scheme programming language, provides two functionalities, callbacks and run-time interpreted scripting. Through the callback functionality, nodes can have internal processing performed in scheme. While this functionality is rarely used, run-time interpreted scheme is heavily used, both as a scripting level programming environment and in interactive development. Bindings allow the user to create objects and to establish the connections between objects within scheme. The connections between objects are part of the

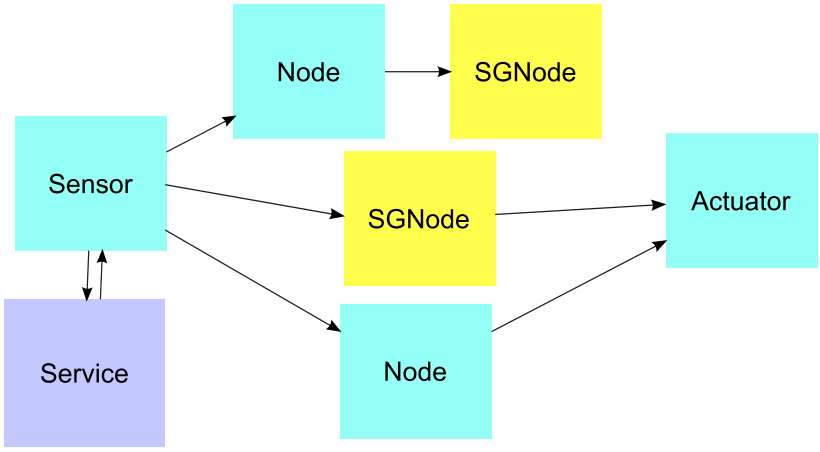

Figure 1: A demonstative data-flow in the AVANGO system. Data flows from the sensors, through various nodes to actuators and SG derived nodes.

data-flow graph, implemented by AVANGO's Fields and Field Connections. The Fields encapsulate the state of the node and present an interface to the values. The Field Connections maintain one-way updates between Fields, keeping the values up to date automatically.

The basic flow of data through the AVANGO data-flow graph follows that of OpenInventor and is shown in Figure 1. Special *Sensors* detect the outside world or other sources of new information and place them in Fields. The Sensors often get the new values from *Services*, which centralize widely used code. All nodes and objects in AVANGO make use of the Fields to get values and pass values to any underlying mechanism, e.g. in the case of SG nodes, they place the incoming values into the underlying SG component. Finally, *Actuators* send output to the user. The SG is an implicit Actuator and there is currently only one other Actuator.

3 FRVR-AVANGO Implementation

The FRVR system has been designed to be portable across VR software systems. Instead of coupling the system into a specific VR, as classically performed, the FRVR system is loosely coupled to the VR system. Partially due to deficiencies in the Haskell compiler and partially to achieve this loose coupling, the Haskell FRP simulation runs in a separate thread. Information is exchanged between the VR system and the Haskell FRP implementation via a shared memory BlackBoard [Cor91]. The BlackBoard (BB) implementation is simply a shared memory arena, where values are tagged with names.

The basic integration of FRVR in AVANGO adheres to AVANGO design principles, and is shown in Figure 2. The access to the Haskell FRP side is encapsulated into a special Service, *fpFRVRService*. This Service initializes and maintains the connection with the Haskell thread and handles movement of data into and out of the BB. A special class of Sensor, fpFRVRSensorBase, implements the basics of connecting to the fpFRVRService to

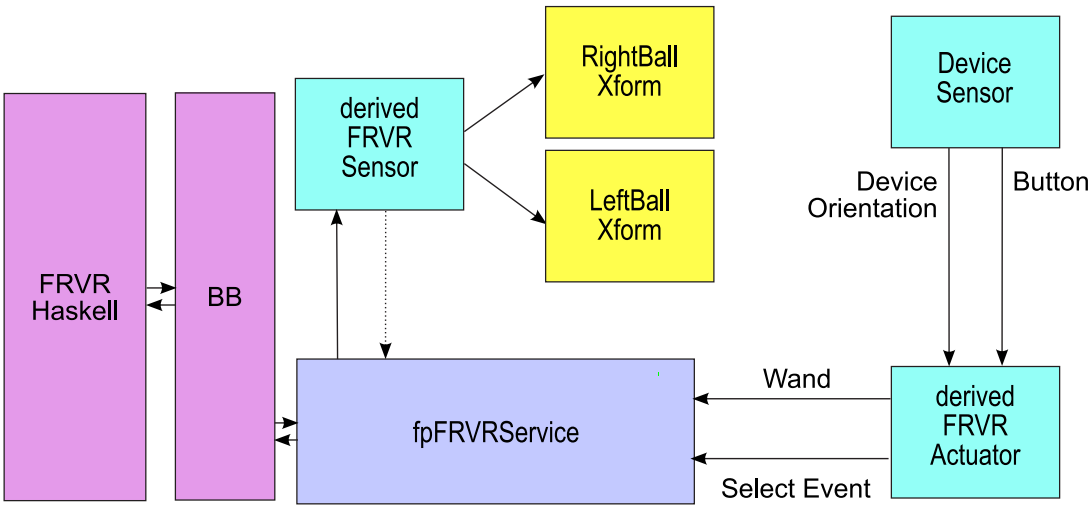

Figure 2: This figure shows the flow of information in the FRVR AVANGO coupling, on hand an implementation of a Newton's Cradle. The derived Sensor and Actuator classes are extended with Fields to deliver/receive the values from/to the FRVR system.

get values. The user can derive from this Sensor to define what values will be retrieved and define the output Fields of the Sensor. An alternative Sensor implementation is presented in Section 4. The Sensors inform the fpFRVRService of values of interest, using their names and initializing the values. Thereafter, the Sensors retrieve values every frame. The Sensor's implementation packs the values into Fields, thereby inserting them into the dataflow graph.

The flow of information inside the FRVR AVANGO implementation follows as illustrated in Figure 2. The Haskell FRP system is executed by the Service every frame, before the Sensors and other nodes updates occur. The Haskell FRP simulation side updates all values, including any asserted Events, as required. The Sensors, which access the fpFRVRService, then retrieve the new values and set them into the data flow. When the FRVR system is used to control dynamics in the environment, the setup is complete. The values can be routed via the Field mechanisms to the proper place, using scheme to perform the setup.

Incorporating interaction into the system is possible using similar techniques. Interaction, however, is a much more complex design question than that of simply driving dynamics with FRVR. Many different manners of integrating interaction exist, even within AVANGO itself. Here, we present basic support for interaction using FRVR. Interaction controlled in FRVR relies mostly on the correct values being placed into the BlackBoard. From there, the Haskell FRP code retrieves the values and events from the VR system, processing them as appropriate. The *fpFRVRActuator* class mirrors the fpFRVRSensor for inserting data in the BlackBoard.

The connectivity for a dynamic, interactive object can be shown with an implementation of a Newton's Cradle, for which the FRP side is described in [BB07]. The two outermost balls are independently modeled and hung under transformation nodes. A Sensor retrieves the orientation of the RightBall and LeftBall. Fields of those names are created in the

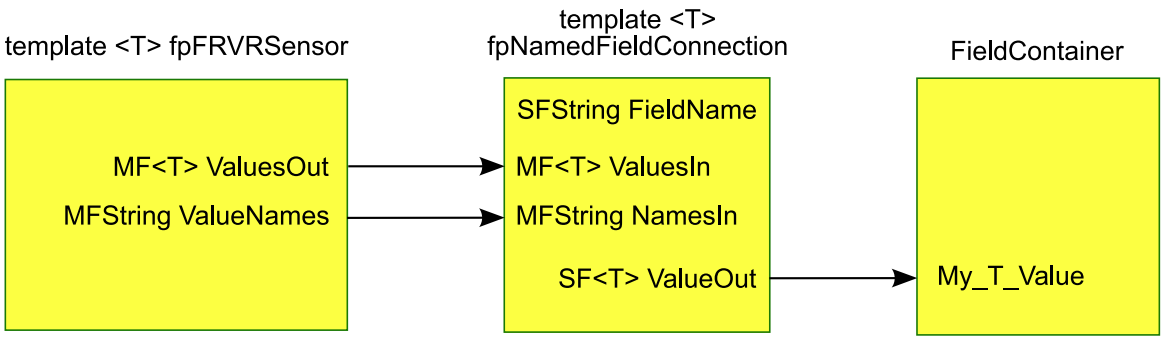

Figure 3: The class descriptions of MultiField reflection system of FRVR-AVANGO. The fpFRVRSensor and fpNamedFieldConnection template classes allow access to the run-time defined "Fields" as if they were normal Fields through their tags.

Sensor and FieldConnections to the transformation nodes automatically deliver values every frame. To include interaction, an Actuator with a Wand matrix field and an Event field for selection, inserts information into the FRP system. The Wand and Selection fields are then connected, via scheme scripts, to input devices, such as the stylus and stylus button. This setup is shown in Figure 2.

4 Run-Time Reflective Fields and Field Connections

In the previous section, an implementation integrating the FRVR system in AVANGO was described. In that design, the user is required to either implement a derived Sensor or embed the connections to the fpFRVRService into their own class. One of FRVR's design goals is to simplify the programming of interactive dynamics, making the requirement of using AVANGO's somewhat complex C++ class inheritance less than soptimal. In this section, we describe an extension to AVANGO that enables a style of reflection on run-time defined Fields.

This extension leverages the fact that the shared memory of FRVR uses a named memory approach typical of BlackBoard architectures for the exchange of information between the VR system and Haskell side. By maintaining two lists, one containing the names of values to be retrieved and the other holding the actual values, one can dynamically insert the FRVR generated information into the AVANGO dataflow system. A Sensor receiving values all of a single type can use the Multi-Field version of AVANGO's Fields to perform this. Unfortunately, this still requires programming by the user, as no classes are currently prepared to take Multi-Field values and extract a single value out of the Multi-Field.

In order to make this useful, AVANGO's Field Connection principle is extended by placing a connector in between. The developed connector, *fpNamedFieldConnection*, is attached to a special Sensor, implementing the above Multi-Field. The connector is set to sort out a specifically named value from the list. The output of the fpNamedFieldConnection is simply a single value Field of the same type. Figure 3 shows the design of the classes

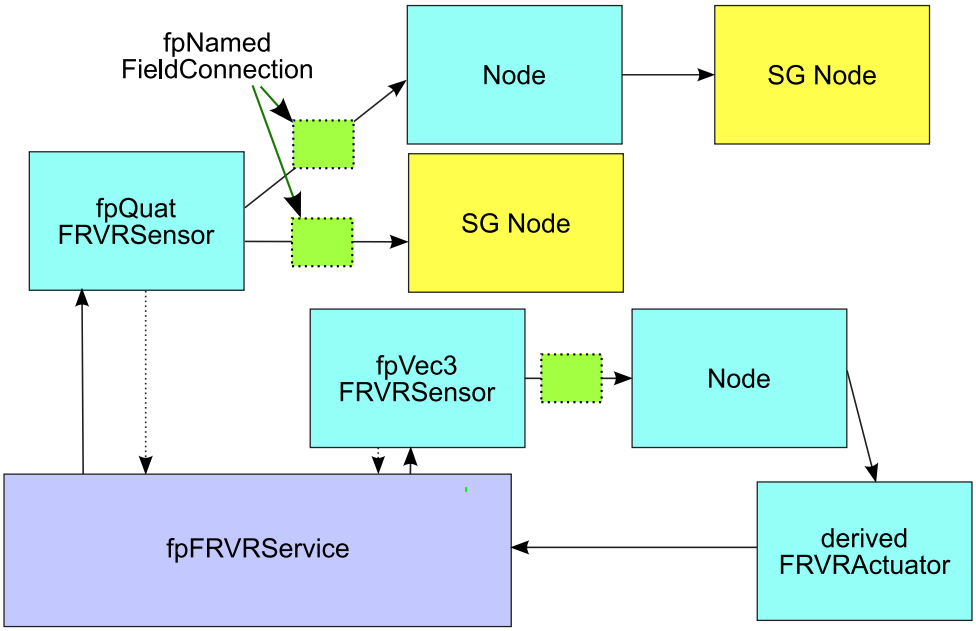

Figure 4: The flow of data through the extended FRVR-AVANGO system. Incorporating the fpNamedFieldConnection objects, removes the need for extending C++ classes to access FRVR values.

that implement this. In order to avoid a possible frame delay, caused by the Field update mechanism, the fpNamedFieldConnection processes the input on any change of the input Multi-Field. The processing of values from FRVR, through the extended fpFRVRSensor and using fpNamedFieldConnection, can be seen in Figure 4. In order to make this a more agile design pattern, both fpFRVRSensor and fpNamedFieldConnection are implemented as C++ templates, parameterized across the Field/data type.

5 Conclusion and Future Work

This paper has shown how the recently developed paradigm, Functional Reactive Programming, can be effectively integrated into AVANGO, a complex higher level VR system. FRP is coupled loosely into AVANGO, requiring no changes to the underlying system. The implementation uses AVANGO's data-flow system for the distribution of data. A developed extension to AVANGO further simplify the users work, by providing a form of run-time reflective Field Connection. In combination, FRP in the high-level AVANGO VR system, allows the user to create dynamic, interactive VEs without the need to program neither dynamics in low-level languages nor connecting stubs in the low-level languages.

There are several intriguing areas to pursue with regards to FRVR and AVANGO. The implementation shown here works well and is a desirable way to integrate the systems. However, there is an additional way of integrating the Haskell FRP environment. Having a FRP callback per node, like AVANGO's scheme callbacks, would alleviate some of the issues of having to route values to the FRP system and from it. This would potentially increase

the expressive power of such a combination even further. In total, this first demonstration of the FRP system's use in a complex external system provides a new level of support for creating dynamic, interactive VEs.

References

[BB07] Kristopher J. Blom and Steffi Beckhaus. Functional Reactive Virtual Reality. In *the Short Paper Proceedings of the IPT- EuroGraphics workshop on Virual Environments (IPT-EGVE '07)*. EuroGraphics, June 2007.

[BJ98] Allen Bierbaum and Christopher Just. Software Tools for Virtual Reality Application Development. In *SIGGRAPH Course Notes*. ACM SIGgraph, 98.

[BLM+04] S. Beckhaus, A. Lechner, S. Mostafawy, G. Trogemann, and R. Wages. alVRed - methods and tools for storytelling in virtual environments. In *Internationale Statustagung "Virtuelle und Erweiterte Realität"*, Leipzig, Germany, 2004.

[CKP95] James Cremer, Joseph Kearney, and Yiannis Papelis. HCSM: a framework for behavior and scenario control in virtual environments. *ACM Transactions on Modelling and Computer Simulation*, 5(3):242–267, 1995.

[CNP03] Antony Courtney, Henrik Nilsson, and John Peterson. The Yampa Arcade. In *ACM SIGPLAN Haskell Workshop*, pages 7–18. ACM SIGPLAN, 2003.

[Cor91] Daniel D. Corkill. Blackboard Systems. *Journal of AI Expert*, 6(9):40–47, 9 1991.

[Cou04] Antony Courtney. *Modeling User Interfaces in a Functional Language*. PhD thesis, Yale University, May 2004.

[Del00] Leonidas Deligiannidis. *DLoVe: A specification paradigm for designing distributed VR applications for single or multiple users*. PhD thesis, Tufts University, 2000.

[TLG99] Russell Turner, Song Li, and Enrico Gobbetti. Metis - an object-oriented toolkit for constructing virtual reality applications. *Computer Graphics Forum*, 18(2):121–130, 1999.

[Tra99] Henrik Tramberend. Avocado: A distributed virtual reality framework. In *Proceedings of IEEE Virtual Reality*, pages 14–21. IEEE Society Press, 1999.

[Tra03] Henrik Tramberend. *Avocado: A Distributed Virtual Environment Framework*. PhD thesis, Universitat Bielefeld, 2003.

[WG93] Josie Wernecke and Open Inventor Architecture Group. *The Inventor Mentor: Programming Object-oriented 3D graphics with Open Inventor, release 2*. Addison-Wesley Publishing Company, 2 edition, 1993.