7

VR Development with InTml

Pablo Figueroa
Universidad de los Andes
Colombia

1. Introduction

Controlling complexity is the essence of computer programming. Brian Kernighan
VR applications are very interesting pieces of technology, not only from the point of view of final users who are immersed in a compelling experience, but also to developers. VR applications are a real challenge in terms of development constraints: they gather information from users through several and possibly redundant input devices, they have to compute a simulation in the order of miliseconds, and they have to deliver output through several devices and modalities at interactive rates. In terms of APIs, VR applications are built on top of a wide variety of software technologies in order to accomplish their goals: from low level drivers that communicate with devices to specialized 3D render APIs, from sockets to real time geometrical algorithms, from XML readers to streaming technologies. Developers should also know about several fields related to computing such as networking, data mangling, simulation, computer graphics, haptics, and human factors. There are several toolkits, libraries, and frameworks that developers could use in this endeavor, so applications can benefit from previous solutions.
However, to this date, VR development is still a challenge. On top of the steepy learning curve of most VR development toolkits, final applications may be unstable, prone to errors, hard to customize to particular user needs and features, difficult to deploy, and technology dependent, among other concerns. Part of these issues are related to the inherent complexity of the technologies involved in development, where the lack of standards and the wide variety of providers make work harder. Part is also related to the inherent focus of a particular VR application, which usually concentrates resources in certain goals (i.e. a particular user study), while treating others as not as important (i.e. code reusability).
Common solutions in VR development are VR toolkits and APIs, which may offer standard solutions to certain problems. Although there are examples of mature tools in the field, some of these may be either too difficult, too limiting, or too low level for novices. Some researchers such as Trenholme & Smith (2008) have tried to use tools and techniques from the game industry, which by far exceeds the size and economic force of its VR ancestor. Most of the success of the game industry is due to the vast amount of resources dedicated to improve gamer's experience, but it is also important to notice the availability of powerful game engines which help developers to handle complexity. A game engine allows developers to create compelling results in a short time, by hiding complex parts of a solution under specialized APIs. However, some solutions, shortcuts, and workarounds in games are not adequate for VR, where simulation fidelity and device support are very important.

Our long term goal is to facilitate VR and Mixed Reality (MR) development, by dividing its complexity among several people with different roles. For this reason we created the Interaction Techniques Markup Language (InTml) and a set of tools around this concept. InTml allows us to divide concerns in two main categories, one directed to architectural design, and the other related to code; one high level abstraction, and the other low level implementation. InTml offers an abstraction for the description of VR applications, independent from a particular set of device drivers, VR toolkits, libraries, and programming languages; an abstraction powerful enough to describe a wide variety of applications in the VR domain. This could make VR applications more portable in the future, since their abstract description in InTml may be ported to several technologies. Finally, InTml makes easier to identify particular devices and interaction techniques in an application, so they can be replaced if it is important to port an application to a new hardware environment. We call this process *application retargeting*, and we hope that in the future it will be an important element of VR application maintenance and evolution.
This chapter is organized as follows: First, we present some introductory examples and several relevant aspects in InTml development. Next, our development process and variability factors are presented. Later we present more examples of use and relevant related work. Finally, we describe future work.

2. An introduction to InTml

An InTml application is basically a set of components connected between them. Such components are called filters, which may represent devices, content, or behavior in an application. There is a library of available filter classes, and it is possible to add new classes to the system. This system can be executed in several runtime implementations, based on generic programming languages. We show first an abstract example of an InTml application and later we describe the concept of a filter class. Then, we show how such applications are created and executed in our IDE. Finally, we present the InTml's abstract execution model and an example of an execution, which can be implemented in both a parallel or sequential fashion. In general, InTml hides from designers certain elements of complexity, which will be described in Section 3.1, so developers can use or improve technologies behind the scene.

2.1 An InTml application

Our first example is shown in Figure 1, an application that allows a user to move a virtual hand with a tracker and touch virtual objects. In this example, a device (*handTracker*) gives position and orientation information to an object (*handRepr*). The behavior filter *SelectByTouching* receives the actual *handRepr* and *scene* objects, and any changes in position or orientation from *handRepr*. Once a collision is detected the selected objects are passed to *Feedback*, which activates a white bounding box around such objects. At the end of each execution step, *console* will render all objects in the scene (both *handRepr* and objects inside *scene*).
An InTml application is composed of instances of filter classes, constants, and object holders. Constants can give initial values to selected input ports in the system, and object holders are used as an indirection mechanism. Filters can also be sent as events through the dataflow, which is shown as an output port with a special decoration (two examples are *handRepr* and *scene* in Figure 1). We also use a special decoration for an output device (i.e. *console*), in order to avoid line cluttering of connections from all objects to the output device.
As we have mentioned, object holders are an indirection mechanism inside InTml. They are placeholders that can hold any piece of content in an application, i.e. any filter that represents

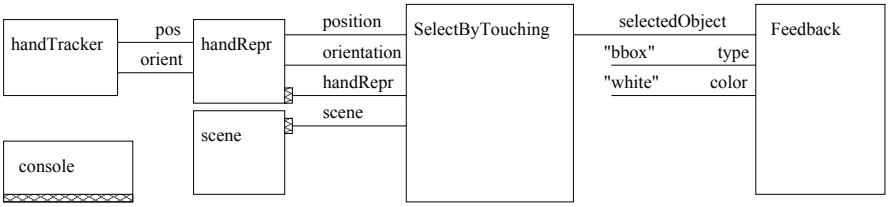

Fig. 1. Simple Application. Touching Objects with a Virtual Hand

content. They have an special input port that is used to change the contained object, and an output port that informs interested filters about changes in the contained object. Other filters can connect to and from an object holder, and those connections will be attached to the contained object during execution. Figure 2 shows examples of object holders inside a composite filter. *GoGoIT*, a composite filter that models the Go Go selection technique by Poupyrev et al. (1996), consists of three object holders (*cube*, *current*, *previous*) and three behaviors (*gogo*, *SelectByTouching*, *FeedbackOne*). *gogo* takes two configuration parameters (*K*,*D*) plus position and orientation of the user's head and hand in order to compute a virtual hand's position and orientation plus the visibility of a cube, that represents a user's real hand. *SelectByTouching* takes a computed virtual hand position, a virtual hand geometry, and a set of selectable objects in the scene in order to compute a selected object. Finally, *FeedbackOne* uses two object holders in order to modify color and bounding box of the current and previous selected objects. These two last object holders are a good example of the indirection mechanism: no matter which object is selected at any time, *FeedbackOne* can refer to it and send it events.

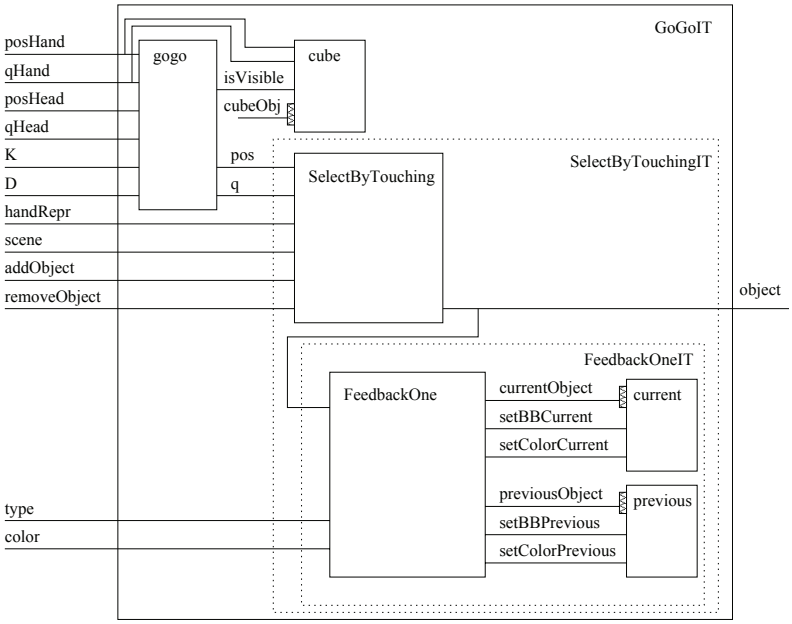

Fig. 2. GoGoIT, a Composite Filter

2.2 Filter classes

Each filter class defines a type of component that can be instantiated in an InTml application. A filter class' instance is a particular element in an application that receives the required information for its computation and produces certain information. Both required and produced information are modeled as a set of ports, which define the set of chunks of information the filter can receive or produce. Each port is defined in terms of a unique name and the type of information it may receive or produce. At any particular time, a filter can receive zero or more events in each one of its input ports, and produce zero or more events in its output ports.

Simple filter classes should encapsulate just one of the following elements: a piece of content, a behavior, or a device. A piece of content could be an interactive object in the 3D scene, a widget in the interface, a sound effect, or a haptic effect, with ports that allow developers to configure its initial state of modify such content during execution, i.e. activating an effect or changing an object's color. A behavior could be either the core algorithm for an interaction technique or an animation effect, with input ports for receiving the required information for its computation and output ports that carry the result of its computation. A device represents a physical device that users can see and manipulate, i.e. a joystick or a tracking system, with input ports for device configuration and output ports that capture and discriminate the information produced.

Instances of simple filters can be used to create applications or composite filters. A composite filter is a special type of filter class that can be used to hide complexity, and it can contain a set interconnected instances of simple or composite filters.

2.3 Abstract execution model

The abstract execution of an InTml application follows a pipeline model, with the following stages per execution frame:

- Data gathering. All data from input devices are gathered during a certain period of time. All events gathered during that period are considered simultaneous.
- Data propagation. Gathered information is propagated through the dataflow. All filters compute their output data from events in their input ports. Filters that represent content accumulate changes without affecting the object's state. This is to assure that any read operation over content will read a consistent state during an execution frame.
- Object holder's execution. If required, object holders change their contained objects first. After, they propagate received events to their contained filters.
- Changes in content. All content filters collect input events, compute their new state, and propagate changes through their output ports. Those changes will be received by interested filters in the coming frame of execution.

The computation of a filter, which occurs inside the data propagation stage or inside the changes in content stage, is divided in three main stages:

- Data gathering. All information generated in a certain time interval is collected. This stage is considered a preprocessing stage, in which filters select and manipulate the information they have received, in order to prepare for the next stage.
- Processing. In this stage a filter executes, given the collected input information and its internal state. Output information is generated, but not propagated
- Output propagation. Output information is propagated to all interested filters.

This model allows the parallel execution of filters, if the required computational resources are available, as we will show in Section 2.6.

2.4 Design and execution of InTml applications

By means of our IDE, an application is created by instantiating the appropriate filter classes. Filters may come from the predefined libraries of classes, organized by the three main categories: objects, devices, and behaviors. Developers can also add their own libraries of filter classes, if necessary, by creating the abstract description of each filter class (name, input and output ports). Figure 3 shows the application view the InTml IDE, that allows designers to load new libraries, create new filter instances, constants, filter holders[1], or links between filters. Figure 4 shows the library editor that allows the definition of new filter classes, by means of the specification of its input and output ports. This editor also allows code generation for each filter class in each one of the runtime environments[2].

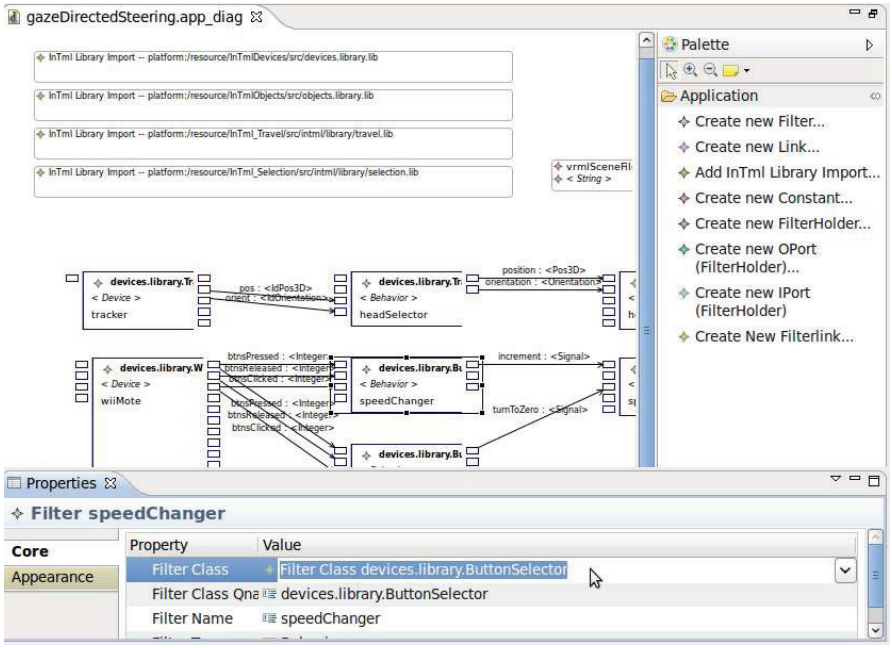

Fig. 3. Application View in the InTml IDE.

Applications can be run inside the IDE with the common method in eclipse, by the `Run As...` wizard.

The library of filter classes has been designed with reusability in mind, from a subset of interaction techniques presented in Bowman et al. (2004). However, this requires to rethink applications to an order that maximizes reuse. For example, Figure 5 shows a version of the application in Figure 1, with maximum reuse and extra functionality in mind. The *tracker*

[1]Object holders are called Filter holders in this interface, although they can only hold filters that represent objects. The creation of a filter holder involves the definition of its input and output ports. The IDE does not support yet composite filters.

[2]Currently, there are independent runtimes in C++, Java, and Actionscript.

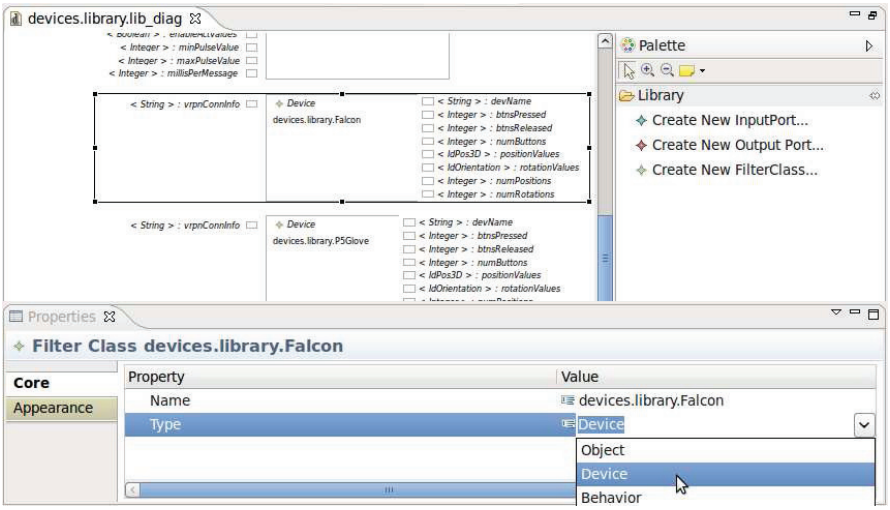

Fig. 4. Library View in the InTml IDE.

device receives a configuration string and outputs streams of positions and orientations, from all tracker elements it may have. The *handSelector* and *headSelector* filters separate from these streams the trackers with ids 0 and 1, and values from those devices can be transformed (i.e. moved, rotated, or scales) at *handOffset* and *headOffset*. The output of these two filters transform a *virtualHand* object and the system's *camera*. The *scene* filter loads and separates objects from an input file, and some of them are identified for selection at *objectsForSelection*. Finally, *collision* receives the *virtualHand* and the objects for selection and outputs objects that are collided by the virtual hand, which are visually enhanced by *feedback*. This diagram may be reduced to the one in Figure 1, by encapsulating *tracker*, *handSelector*, and *handOffset* into a composite filter, by eliminating the filters involved in camera movement, and by making explicit the *console* filter.

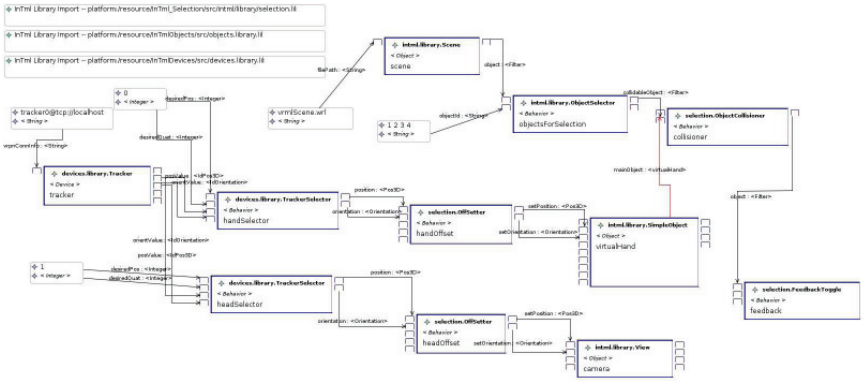

Fig. 5. Library Based Version of Touching Objects with a Virtual Hand.

2.5 IDE development

Our current IDE is based in the concepts of Model Driven Architecture by Stahl & Veolter (2006) and Software Product Lines by Clements & Northrop (2002); and it uses technologies such as The Eclipse Foundation (2007)'s IDE for the basic but extendable environment, EMF and GMF as frameworks inside eclipse for the visual programming environment and the openArchitectureWare.org (2008)'s oAW for code generation in Java, ActionScript, and C++.
This IDE has been developed as follows: first, an eCore[3] model of InTml is developed. Such model includes model constraints that help designers to identify errors during development. Then, graphic elements, graphic tools, and interface code are defined for the core model, as it is required by GMF. Based on this output, oAW's templates and constraints are defined in order to generate code for the targeted platforms. It is interesting to notice that both GMF and oAW provide mechanisms for constraint description, which provide a better interface and error feedback to designers. A side effect of this last development is a change in the final XML format for an InTml application: Initially we had defined our own format and DTD. With this final development, we have to use the XML format generated by GMF. This is a minor issue, since the visual programming environment provides a much better experience to designers than our previous XML editor.
During development we have performed two usability tests, the first one with VR developers and the second with non-programmers. In the first test we showed our IDE to 4 students with previous experience in VR development. Subjects received a short introduction to the IDE, see how a small example was developed, and were asked to answer some questions regarding the interface. Those comments were used in order to produce and improved version. In the second experiment 26 graduate students in an extension course of our Arts Department received training in InTml, and produced two designs in which they could optionally use our IDE. Finally, they were asked to fill a questionnaire about InTml's ease of learning, IDE's feedback, restrictions, consistency, and functionality. After 9 hours of exposure, they found InTml easy to learn, although some problems in understanding the execution model were detected. We believe this is due to the lack of experience they had with the actual application in execution, since they were required to design an application, not to execute it. In terms of the IDE they found issues with feedback, which are part of further changes to the environment.

2.6 An example of InTml's operational semantics

We have developed in the Z formal notation by Spivey (1992) a language and platform independent description of the InTml model. We describe in such a notation the concept of a filter, how filters can be composed of filters that hide complexity, how filters process information at any time step, how information gets propagated thoughout a dataflow of filters, and controlled ways to change the dataflow at runtime. The details of the formal description are mentioned in Figueroa et al. (2004). Although this description requires a good understanding of Z as a formal language and in consequence it may not be suited for general communication of the InTml capabilities, it is very precise and programming–language neutral. In particular, it has been used as blueprints for both C++ and Java implementations. Here we show with an example the main features that such a model gives to our VR applications. In Figueroa (2008) it is possible to find this description plus some motivations for this model.

[3] An eCore model is a UML class model with limited syntax.

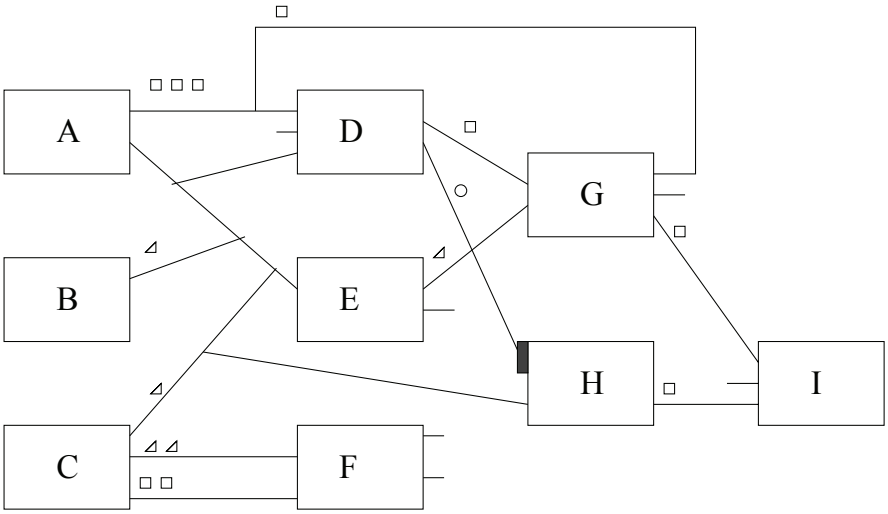

Fig. 6. A Time Step in an InTml Application. By convention, events are propagated from left to right.

Figure 6 shows an example of a the state of an InTml application in a particular time step, in which we consider **H** an object holder, **E** and **F** two content objects. This example shows the following features of an InTml application during execution:

- A filter can have several input and output ports, which may or may not be connected to other filters. In this way, filters can be reused in different scenarios without common restrictions imposed by standard function calls, which parameters are always mandatory.
- Different filter types such as devices, interactive content, and behavior are first class citizens of this description. Appart from the details already described in the execution of object holders and content objects, all filters seem equal from the design point of view.
- A time step defines a lapse of time in which all events from input devices are considered simultaneous. If **A**, **B**, and **C** are devices, all events they generate during such a small lapse of time will be processed together, no matter the particular generation rates from each device. In this example, **A** has generated three events in one of its output ports, **B** just one, and **C** outputs five events in total.
- Cycles are allowed in the description of an application, but they are broken for the execution of a time step. In the case of this example, the cycle **DGD** is broken and treated in a special way, i.e. delaying events from **G** to **D** to the next time step.
- Filters execute at most once in a time step, and the information they produce is considered simultaneous. A topological sort can be used in order to find a sequential execution order, i.e. **ADBCEGHFI** in the example. Such an order could be paralelized in the subsets **[**{**A,B,C**},{**D,E,F**},{**G,H**},{**I**}**]**, without any effect on the inputs and outputs of each filter. In this regard, InTml guarantees a consistent execution no matter the number of execution threads.
- An object holder has a special input port that allows to replace the contained object (i.e. the connection from **D** to **H** will provide objects to be contained in **H**). Events received in other

ports are propagated to the contained object (i.e. events from **C** to **H**), and events generated from the contained object are propagated to registered filters (i.e. events from **H** to **I**).

– Since content objects can be related in structures that are not evident from the InTml dataflow (i.e. in a scene graph), which may require rule checking and change validation, and since content objects can be queried by several filters in the dataflow, all changes in objects are queued until the end of a time step. For example, **E**, **F**, and the object inside **H** could be parts of the same animated avatar, which have to fulfill certain rules and restrictions in its movements. Again in our example, this means that although all filters will execute at most once in a time step, output events from **E** and **H** will be delayed one step[4]. and the entire dataflow will require two frames of execution in order to execute each filter at least once.

3. Development process

Our development process is depicted in Figure 7. We divide tasks between two roles: a designer and a developer. A designer is a novice or non programmer that is interested in developing novel applications based on a set of predefined filters. A developer is a programmer that knows how to create novel filters and novel applications, or it could also be a support asset for a designer that requires to implement novel filter classes. We show here how we used this process from a designer's point of view for a family of applications described at Figueroa et al. (2005), a matching test that shows three objects and three copies of such objects, to be executed in four VR hardware setups.

Identify application goals We identify the set of use cases that the application has to fulfill: In this example it could be to select an object, move an object to the position and orientation of its copy, remove matched objects, define an initial state for objects, and save chronology of interesting events.

Describe application requirements in InTml documents For this stage we define a dataflow that fulfills all goals. We have found that it is more readable for designers to make one dataflow per goal, with cross references between them. Each dataflow is a subset of the entire application, and it is called a Task View. We do not show here the task views for this application, but examples of Task Views can be seen in Section 4.1.

Are current libraries enough? Members of this family of applications were consecutively developed. The first application of this family was developed from scratch, so there was no library at that time and all filters were application dependant. From this version on, each new application adapted existing filters in order to make them more reusable, or created new ones when necessary. In this case, the entire set of tasks for developers were performed as part of this step.

Check correctness in InTml documents Basic checking of InTml documents can be automatically done by tools: types and names of ports in instanced filters, type correspondence in port connections, or validity of filter classes, among other things. We have developed some tools in order to identify initial problems.

Execute/Test InTml application Once filter classes are implemented by developers, designers can run their design and test its usability. In our case we tested our prototypes with users from our staff, in order to identify improvements in their user interfaces.

[4]This example is not interested in the output of F, which is also delayed

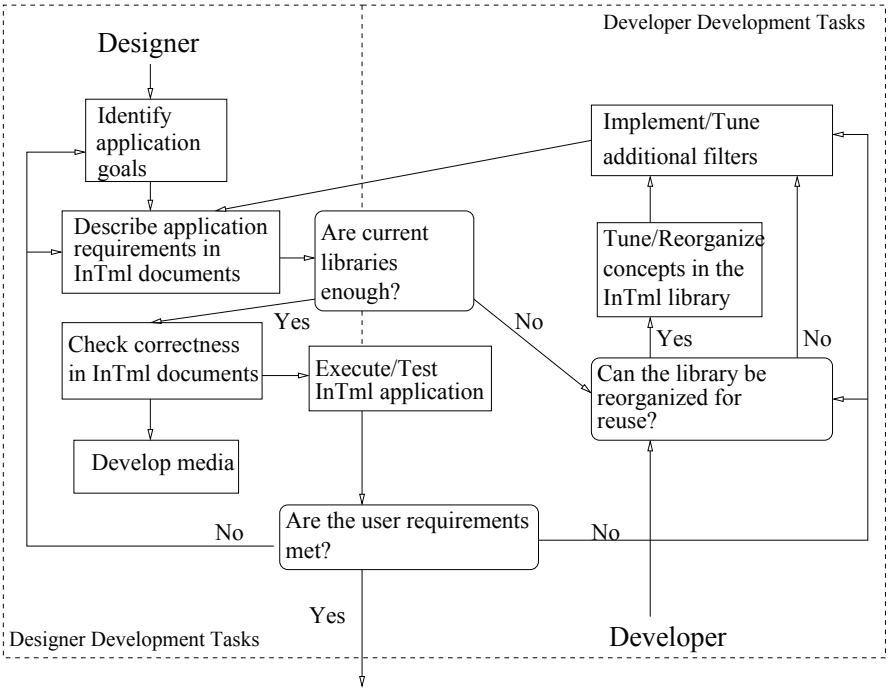

Fig. 7. InTml–Based Development Process

Develop media If required, specific application media should be developed in this step. Since it is possible to use basic models as surrogates in initial stages of development, it is possible to delay this task until the end, or even make this task in parallel. In the case of our example, 3D objects were obtained from public repositories.

Are the user requirements met? Once usability tests are performed, it is possible to identify improvements. Here such improvements are defined in terms of new goals, which are input for the new cycle in the development process.

3.1 Variability in application families

Variability is what makes different a set of applications with the same functionality but in different hardware setups. Variability's description in novel applications is very complex, due to the big variety of user types, devices, interaction techniques, visualization aids, frameworks, and libraries that may be used. We decided to ease development of non–programmers by dividing the variability spectrum in issues at the level of InTml language and issues at the level of the InTml implementation. An InTml family of applications consists of the following elements:

– A common set of basic types. Basic types in InTml are the equivalent of basic types in common programming languages, such as int or float. They have to be instantiated to available types in a particular InTml implementation.

– A library of filter types. Filter classes are reuse units. Such classes can use qualifiers in order to group them, in a similar way as packages can group Java classes.

– Applications. As we have seen, an InTml application is a set of interconnected instances of filter classes, constants and object holders. An application is divided in tasks, a subset of the application's dataflow. Each application can copy and redefine tasks from other applications in the family.

These elements allow us to address the following types of variability among applications in the same family:

– Devices. Each input and output device is represented by a filter in InTml, which may be instantiated or replaced in an application as desired.
– User types. Support for several user types is represented as different applications in the family, which may share common tasks.
– Interaction techniques. Developers can (and should) change the interaction techniques of a particular application depending on the type of user and devices in use. Such a change consists in the replacement of devices, behavior, or content related to a particular technique from one application to another. This replacement is feasible because filters clearly separate interaction techniques from the rest of the application.

Although also important, the following variation points in a MR application family are hidden from the InTml designer, and should be implemented one level below by an experienced programmer:

– Levels of detail and performance. A particular content could be shown at different levels of detail, depending on the capabilities of the available hardware. In the same way, a particular behavior could be adapted to the particular computational power of the underlying hardware.
– Context awareness. Devices could adapt their behavior to environmental factors, such as light conditions.
– Runtime adaptation. Several InTml applications can be combined in just one executable, which may switch between implementations depending on external factors such as user types.
– Particular APIs and frameworks. InTml can be implemented on top of a wide variety of APIs and frameworks, depending on the desired functionality at the high level and how convenient is to reuse a particular piece of software. A programmer should take into consideration reuse tradeoffs and integrate new elements when feasible.

This separation of variation points allows non–programmers to define by themselves their own prototypes, without special considerations about the InTml implementation. It is up to programmers at the InTml's implementation level to exercise low level variations.

4. Examples

We show now examples of how application families can be designed by highlighting three main concepts: The design of an application in terms of Task Views, the variability of a task among several applications in a family, and the basic software support for a prototype of a MR platform. The examples below show only the most important elements, which should be complemented as shown in Section 2.4[5].

[5]The InTml IDE does not support yet Task Views, so examples are shown in the abstract style.

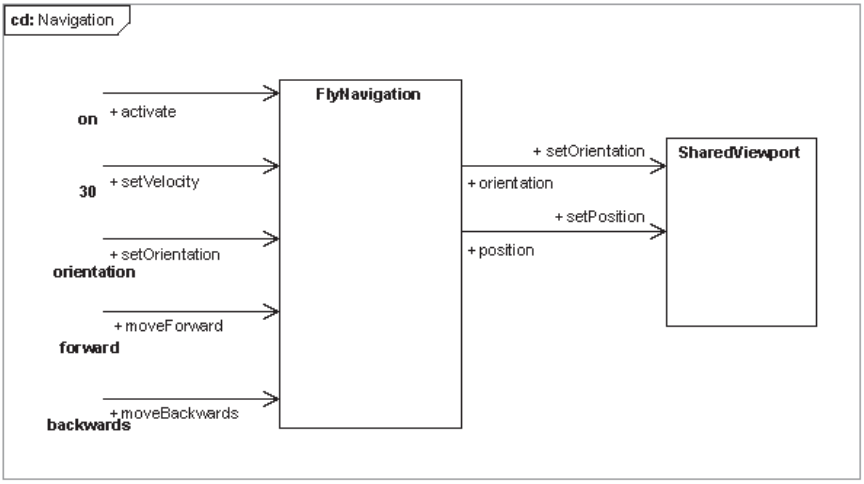

Fig. 8. CAVE Navigation in VWT

4.1 A client for a virtual steering application

We are collaborating in the development of heterogeneous and distributed clients for a virtual wind tunnel (VWT), that uses massive paralelism and fast algorithms for computational fluid dynamics by Boulanger et al. (2006). Figures 8, 9, and 10 show InTml diagrams for the following tasks: navigation in a CAVE environment, sharing viewpoints between clients, and control from a bluetooth–enable device. This application is going to be implemented in the following environments: a personal environment with a HMD and a Phantom, a Geowall–like environment, and a CAVE–like environment. Figures 9, and 10 apply to all implementations, while Figure 8 defines the navigation technique in a CAVE.

Navigation in a CAVE environment (Figure 8) uses a simple flying metaphor, in which a head tracker defines move direction, and wand buttons move backwards and forwards.

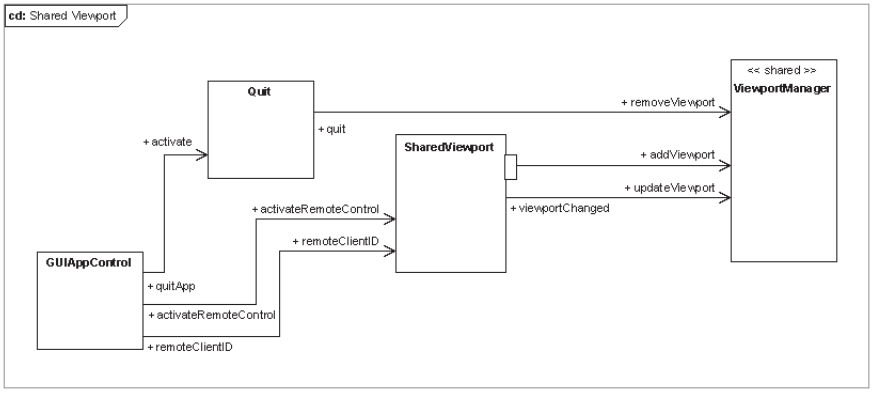

Fig. 9. Sharing a Viewport in VWT

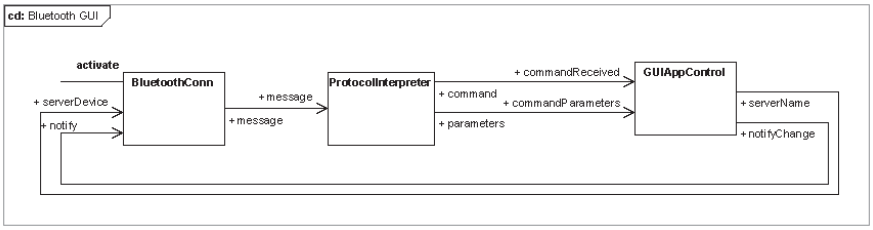

Fig. 10. VWT Control from a Bluetooth device

Shared Viewpoint in Figure 9 describes the task of sharing a representation of a user's viewpoint to all clients in a simulation. It shows the local viewpoint (*SharedViewport*) and how it is added to a pull of viewports, managed by *ViewportManager*. The implementation of this last filter handles the required networking, and the avatars' representation.
Finally, we have a PDA with a bluetooth connection, that allows us to send commands to the control of the application, represented by *GUIAppControl* in Figure 10.

4.2 Navigation tasks in a family

Let's assume we are interested in navigating and showing information about objects in a small but complex VR office, in three hardware platforms: a CAVE, a PC with a joystick, and a cell phone with graphics acceleration. If we concentrate on the navigation task, it is possible to think on interesting and different implementations for such a task in each platform, as follows:

- In a CAVE, a user can navigate to an interesting object by pointing to such an object and selecting it. The system should compute a path from the current viewpoint position to a position in front of such an object. This technique is similar to Fixed–Object Manipulation in (Bowman et al., 2004, p.215), with extra behavior for path planning.
- In a PC with a joystick, a navigation technique that resembles the WALK mode in VRML could be used (*P2D2NavInPlane*). This mode features collision detection between the avatar and objects in the environment.
- In a cell phone, due to computation restrictions and limitations on the input device, it is more convenient to select pre–recorded viewpoints and paths than using the previous navigation techniques. It could be also important to reduce the complexity of the scene as possible.

Figures 11 and 12 show the InTml diagrams of such navigation techniques.

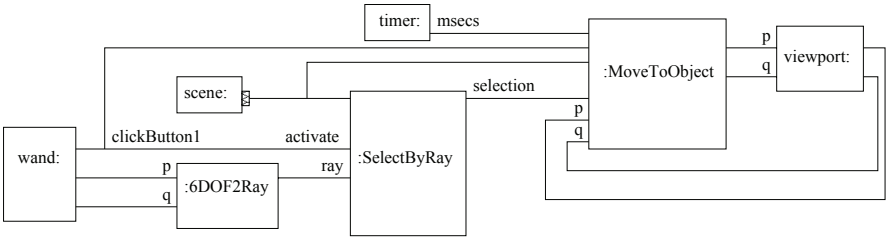

Fig. 11. Navigation in a Cave

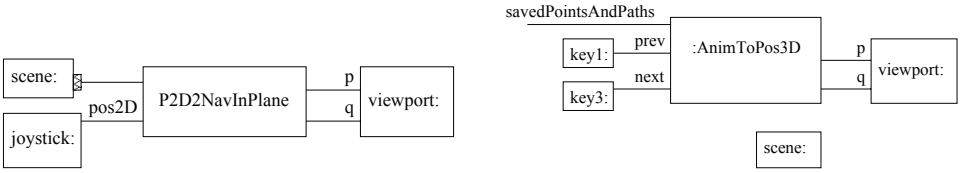

Fig. 12. Navigation in a PC and a Cell Phone

4.3 Software support for a new MR platform

We are developing an integrated MR platform based on the ARToolkit and VRPN, and we are designing a set of reusable filter classes as an API for developers. Such an API will facilitate development to a family of applications in that particular domain, and corresponds to the concept of core assets in the Software Product Line literature such as Clements & Northrop (2002). Figures 13 and 14 show some of the current set of platform-independent filter types, which correspond the following functionality:

- *MappedVRPNTracker*, which gives 6DOF data from an identified pattern and a definition of its local coordinate system.
- *Switch*, which sends as output one of the predefined inputs once a signal is received.
- *Scene*, which allows to select and copy an object by giving an id.
- *Map2DtoTerrain*, which outputs a 3D position over a terrain from a 2D one.
- *Collision*, which identify a collision between two objects.

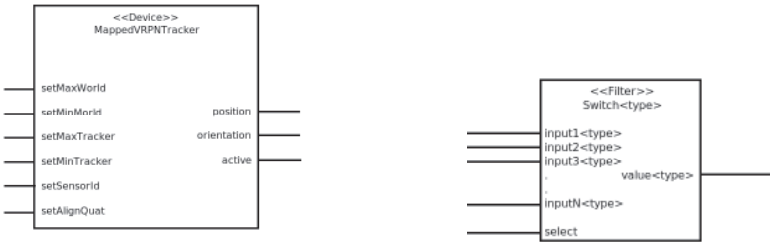

Fig. 13. A Tracker for ARToolkit Patterns and a Switch Filter

These filters were identified in a study of three concepts in interior design, and we plan to validate this API with other applications.

5. Related Work

There are many toolkits for VR development with different scope and complexity, even game engines that can be used for this purpose. Some allow developers to configure a wide

Fig. 14. A Scene and Terrain objects, plus A Colision Detection Filters

spectrum of aspects, while others hide some decisions in order to reduce complexity. Some environments are tailored to a particular hardware platform, and others allow developers to use a wide range of input and output devices. By reviewing the way VR programs are developed in several toolkits such as Shaw et al. (1992); VRCO (2003); SGI (2003); Blach et al. (1998); Sense8 (2000); CMU (1999); Bierbaum et al. (2001); Web3D Consortium (2003); Taylor et al. (2001); Sastry et al. (2001); Allard et al. (2004), it can be seen that most environments with wide coverage of hardware platforms require developers to take decisions on many detailed aspects at the same time, and to learn rather complex APIs in a general purpose programming language. Environments with easier to learn environments tend to limit support for devices and novel behavior, precluding the evolution of VR applications. However, there are some environments that offer a high level programming paradigm with a library of high level constructs and at the same time allow developers to create new high level constructs by writing code in a generic programming language. Such environments are at the same time easy to learn for novices and powerful enough for expert developers. Our solution follows this approach, and incorporates novel solutions related to application code structure, device management, and a scalable execution model.

One of the main problems of current environments, APIs and toolkits for VR development is the proposed structure for application-specific code. Developers should be able to easily incorporate novel devices, interaction techniques, or content to VR applications. However, this is not the case. Some environments such as the ones in Shaw et al. (1992); CMU (1999); Taylor et al. (2001) organize application-specific code around isolated callbacks, which process one event at a time. Each callback should include code related to parts of interaction techniques, event correlation, and modifications to output data structures. This scheme is difficult to scale to complex applications, since isolated callbacks are not enough as an organization scheme for an entire application and developers have to struggle in order to incorporate more advanced architectural styles. Other environments such as Bierbaum et al. (2001); SGI (2003) add new behavior around the main rendering loop. In this case, code with the new functionality can be written in specific callbacks, which are called at specific stages, usually before or after rendering. Again, this structure intertwines code related to interaction techniques, application behavior, or gathering input devices data. Finally, other environments allow developers to read as many devices as they want in a particular point of code, which is very convenient for event correlation, but can lead to coincidental coupling between devices.

There are also limitations related to the core APIs in use and the way they handle novel input devices. Current environments usually define a fixed set of input types, for example, keyboard events and mouse events, with extra information from special keys on the keyboard (i.e. shift, alt, and ctrl). Events from other devices are usually translated to available ones. For example, joystick events can be translated to mouse events. This limits the number of devices that can be simultaneously used and the type of input events that an application can receive. Some toolkits provide extension mechanisms for new devices or new events, but these capabilities target senior developers, and are rarely used.

Despite their success with standard interfaces, traditional architectures have the following limitations for VR applications:

– There are no provisions for more complex structures between callbacks, and their interactions are difficult to model. Generally, all callbacks are just at one level from the dispatcher, without relations between them. Java3D by Sun Microsystems (1997) allows passing control from one callback to another, but the scheme is limited to relationships

between two callbacks, and the code inside each callback has the same reusability problems mentioned here.

– Since all events are queued and serialized, there is no provision for treatment of simultaneous events from different devices with different generation rates.
– Addition of new events from novel input devices is a difficult task, so it is usually avoided by reusing events from standard devices that are not presently in use. This creates problems due to usability differences between devices, and conflicts if new and old devices are used at the same time.
– There are limited possibilities for composition and reuse of third-party components due to the lack of an interface standard, and a notion of composition. It is difficult to compose callbacks that were previously developed for other purposes.

Our proposal uses data flow as the high level model for passing control and data between components, similar to the one in Allard et al. (2004). With such a structure it is possible to model complex dependencies between tasks and interaction techniques. In contrast, the callback model does not scale well to more complex structures, where dependencies among callbacks are required. A model based on dataflow can better define relationships between different behavior components in the system, and it clearly exposes component dependencies. Some systems such as Carey & Bell (1997); Web3D Consortium (2005); Ava (2000); Blach et al. (1998); Virtools (2007) have used a similar structure, but they usually take the very simple execution model of propagating one event at a time. Our approach differs from the one in Allard et al. (2004) in the way we have adapted the traditional execution models from pipeline processing, such as Synchronous Data Flow architectures presented by Battacharyya et al. (1996), to the following characteristics of VR applications:

– Not all information from input devices needs to be processed in any given period of time. Depending on the computation speed and the refresh rate of output devices, some information from input devices could be irrelevant or outdated. We allow components to define an interval of time where all received information is considered simultaneous, so redundant information inside the interval can be eliminated. Such information does not affect successive intervals, so discarded information do not affect future executions. The model in Allard et al. (2004) uses extra control connections in order to handle computation distribution, and only allows one output per interval of time in each module.
– New input and output devices are common in new applications. It should be simple to add new devices to an application. Moreover, simultaneous events from different devices should be easy to detect. Filters with several input and output ports are our solution to this problem. They can model any type of device in an uniform way, and it is easy to create new types of filters for new types of devices. On the other hand, a filter interested in simultaneous events from different devices just needs to include them as input and read all events received in a time interval from all its inputs.

Some intrinsic characteristics of VR applications are still not directly addressed by the present proposal, such as the desirable fixed refresh rate for output devices. However, it is possible to integrate the work by Shaw et al. (1992) that decouple device reading from simulation execution and even distributed solutions such as Allard et al. (2004), with some limitations.
A dataflow architecture also allows us to consider dynamic and static scheduling algorithms for machines with several CPUs. This approach cannot be implemented in current dataflow-based solutions such as VRML and X3D, due to intrinsic limitations on the order

of execution of their components in a program. Kwok & Ahmad (1999) discuss several algorithms for static scheduling, and solutions for arbitrary graph structures with arbitrary computational costs per node, such as CP/MISF and DF/IHS, are promising for high performance solutions in VR.
Our work in InTml differs from previous approaches in several ways:

- InTml provides a way to both hide implementation details and allow changes in any behavior that the application may provide. There are some development environments with high level, user-friendly languages (e.g. Web3D Consortium (2003); CMU (1999)), but they assume some interaction techniques that are either impossible or very difficult to override.
- InTml provides a formally described language and a component-based development environment suitable for reuse on different hardware platforms. Some component-based solutions are available in Blach et al. (1998); Web3D Consortium (2003); Dachselt et al. (2002), but without a formal description of their semantics.
- InTml can be implemented on top of a wide variety of existing libraries and toolkits, so it can provide a unified and executable description for VR applications.
- InTml takes a novel approach to the treatment of simultaneous, multimodal events from several devices. We define a dataflow model with a periodic execution that handles several events as simultaneous. Such a model is an evolution of the traditional single-threaded, one event at a time model, inherited from traditional WIMP interfaces.
- InTml is a domain specific language for defining the architecture of VR applications. Some languages in the field such as Web3D Consortium (2003); Autodesk (2006) concentrate mostly on geometry and on the PC-based interaction environment. Others, such as Wingrave & Bowman (2005) use state machines as a design abstraction, which we believe is very powerful although more complex for non-programmers. The same is true of hybrid languages such as the one in Smith & Duke (1999), which proposes a way to combine notations for discrete and continuous signals, using extensions to Petri Nets and state machines. InTml allows unsophisticated developers to model devices, behavior and content, all of them as first-class concepts that are easy to understand and present in any VR hardware platform.
- Some authors such as Massó et al. (2005); Dachselt et al. (2002) have proposed portable ways for describing VR applications, but they have been used on a subset of VR applications, usually Desktop VR.

From the point of view of VR development methodologies, there are some options such as the one by Tanriverdi & Jacob (2001), the user-centered approach in Neale et al. (2002), a UML-based approach in Kim (2005), and a methodology in Sastry et al. (2001) based on a hybrid language. While such alternatives have similarities with, and are extensions to the one presented here, our approach introduces and depends on the key concepts of retargeting and separation of roles.
There have been some attempts to define a concept similar to VR retargeting but restricted to computer graphics. Scalable graphics is a field that studies methods for parallel rendering of scenes. Several authors such as Humphreys et al. (2001); Eldridge et al. (2000); Nishimura & Kunii (1996); Molnar et al. (1992) have proposed algorithms for load balancing of the rendering task over several computers. Application retargeting in VR requires this type of rendering solution, in order to use the capabilities of clusters and parallel machines.

However, retargeting also involves changes in other important elements of a VR application, as we have shown. IBM presented similar ideas in its interpretation of Scalable Graphics by Boier-Martin (2003), but too few details are presented. In summary, our proposal describes how to retarget devices and interaction techniques in VR applications, as opposed to changes in graphic content only.
More details about InTml and its implementation can be found at Figueroa et al. (2008) and Figueroa (2010).

6. Future work

Supporting VR and MR development is a complex task, that requires an important effort in several directions. We believe InTml might be an interesting solution, that allows both developers and designers to construct new applications that can survive despite changes in particular technologies. We need to offer a more friendly environment for both developers and designers, and more functionality in order to make their work easier. In particular, we are improving the support for rapid prototyping from the point of view of a designer, by means of more functionality at the IDE and a more complete library of filter classes. The work of a developer is by no means easier, so we need to find out ways to facilitate the creation of code attached to filter classes, and ways to make the overall architecture clearer for debugging and understanding purposes.
InTml code is published under several open source licenses at Figueroa (2007)

7. References

Allard, J., Gouranton, V., Lecointre, L., Limet, S., Melin, E., Raffin, B. & Robert, S. (2004). Flowvr: A middleware for large scale virtual reality applications, *Euro–Par 2004 Parallel Processing*, Vol. 3149/2004, Springer Berlin / Heidelberg, pp. 497–505.

Autodesk (2006). Autodesk FBX, http://usa.autodesk.com/adsk/servlet/index?id=6837478&siteID=123112.

Ava (2000). Avango: A distributed virtual reality framework, http://imk.gmd.de/docs/ww/ve/projects/proj1_2.mhtml.

Battacharyya, S. S., Murthy, P. K. & Lee, E. A. (1996). *Software Synthesis from Dataflow Graphs*, Kluwer Academic Publishers.

Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A. & Cruz-Neira, C. (2001). VR Juggler: A Virtual Platform for Virtual Reality Application Development, *Proceedings of IEEE Virtual Reality*, IEEE, pp. 89–96.

Blach, R., Landauer, J., Rosh, A. & Simon, A. (1998). A flexible prototyping tool for 3d real-time user interaction, *User-Interaction, Proc. of Virtual Environments*, Springer Wien, pp. 54–1–54–10.

Boier-Martin, I. M. (2003). Adaptive graphics, *Computer Graphics and Applications* 23(1): 6–10.

Boulanger, P., Garcia, M. J., Badke, C. & Ryan, J. (2006). An advanced collaborative infrastructure for the real-time computational steering of large CFD simulations, *European Conference on Computational Fluid Dynamics (ECCOMAS CFD 2006)*, TU Delft. http://www.eccomascfd2006.nl/.

Bowman, D., Kruijff, E., Joseph J. LaViola, J. & Poupyrev, I. (2004). *3D User Interfaces: Theory and Practice*, Addison Wesley.

Carey, R. & Bell, G. (1997). *The Annotated VRML 97 Reference*, chapter 2.10.

Clements, P. & Northrop, L. (2002). *Software Product Lines*, Addison Wesley.

CMU (1999). Alice: Easy interactive 3D graphics, http://www.alice.org. Carnegie Mellon University.

Dachselt, R., Hinz, M. & Meiner, K. (2002). Contigra: an XML–based architecture for component-oriented 3d applications, *Proceeding of the seventh international conference on 3D Web technology*, ACM, ACM Press, pp. 155–163.

Eldridge, M., Igehy, H. & Hanrahan, P. (2000). Pomegranate: a fully scalable graphics architecture, *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 443–454.

Figueroa, P. (2007). Intml development tools, http://sourceforge.net/projects/intml.

Figueroa, P. (2008). Intml: Main concepts, examples, and initial lessons, *Proceedings of the IEEE Virtual Reality 2008 Workshop on Software Engineering and Architecture for Realtime Interactive Systems (SEARIS)*, Shaker-Verlag, pp. 3–6.

Figueroa, P. (2010). Insights on the design of intml, *Presence: Teleoperators and Virtual Environments* 19(2): 118–130.
URL:*http://www.mitpressjournals.org/doi/abs/10.1162/pres.19.2.118*

Figueroa, P., Bischof, W. F., Boulanger, P. & Hoover, H. J. (2005). Efficient comparison of platform alternatives in interactive virtual reality applications, *International Journal of Human-Computer Studies* 62(1): 73–103.

Figueroa, P., Bischof, W. F., Boulanger, P., Hoover, H. J. & Taylor, R. (2008). Intml: A dataflow oriented development system for virtual reality applications, *Presence: Teleoper. Virtual Environ.* 17(5): 492–511.

Figueroa, P., Hoover, J. & Boulanger, P. (2004). Intml concepts, *Technical report*, University of Alberta. Computing Science Department.

Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M. & Hanrahan, P. (2001). Wiregl: a scalable graphics system for clusters, *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 129–140.

Kim, G. J. (2005). *Designing Virtual Reality Systems. The Structured Approach*, Springer.

Kwok, Y.-K. & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Computing Surveys (CSUR)* 31(4): 406–471.

Massó, J. P. M., Vanderdonckt, J., Simarro, F. M. & López, P. G. (2005). Towards virtualization of user interfaces based on usixml, *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, ACM Press, New York, NY, USA, pp. 169–178.

Molnar, S., Eyles, J. & Poulton, J. (1992). Pixelflow: high-speed rendering using image composition, *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 231–240.

Neale, H., Cobb, S. & Wilson, J. (2002). A front ended approach to the user-centred design of ves, *Proceedings of IEEE Virtual Reality*, IEEE, pp. 191–198.

Nishimura, S. & Kunii, T. L. (1996). Vc-1: a scalable graphics computer with virtual local frame buffers, *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 365–372.

openArchitectureWare.org (2008). openarchitectureware, http://www.openarchitectureware.org/.

Poupyrev, I., Billinghurst, M., Weghorst, S. & Ichikawa, T. (1996). The go-go interaction technique: non-linear mapping for direct manipulation in vr, *Proceedings of the 9th annual ACM symposium on User interface software and technology*, ACM, ACM Press, pp. 79–80.

Sastry, L., Boyd, D. & Wilson, M. (2001). Design review and visualization steering using

the inquisitive interaction toolkit, *IPT/EGVE 2001: Joint 5th Immersive Projection Technology Workshop / 7th Eurographics Workshop on Virtual Environments*.

Sense8 (2000). Virtual reality development tools. The sense8 product line, http://www.sense8.com/products/index.html.

SGI (2003). Iris performer home page, http://www.sgi.com/software/performer.

Shaw, C., Liang, J., Green, M. & Sun, Y. (1992). The decoupled simulation model for virtual reality systems, *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, pp. 321–328.

Smith, S. & Duke, D. (1999). The hybrid world of virtual environments, *Eurographics Proceedings*, Vol. 18, Blackwell Publishers, pp. 298–307.

Spivey, M. (1992). *The Z Notation: A Reference Manual*, 2nd edition edn, Prentice-Hall.

Stahl, T. & Veolter, M. (2006). *Model-Driven Software Development : Technology, Engineering, Management*, Wiley.

Sun Microsystems (1997). Java 3D Home Page, http://java.sun.com/products/java-media/3D/index.html.

Tanriverdi, V. & Jacob, R. J. (2001). VRID: A design model and methodology for developing virtual reality interfaces, *in* ACM (ed.), *Proceedings of the ACM Symposium of Virtual Reality Software and Technology*, ACM, ACM Press, pp. 175–182.

Taylor, R. M., Hudson, T. C., Seeger, A., Weber, H., Juliano, J. & Helser, A. T. (2001). VRPN: A device-independent, network-transparent VR peripheral system, *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, pp. 55–61.

The Eclipse Foundation (2007). Eclipse, http://www.eclipse.org/.

Trenholme, D. & Smith, S. P. (2008). Computer game engines for developing first-person virtual environments, *Virtual Reality* 12(3): 181–187.

Virtools (2007). Virtools, http://www.virtools.com/index.asp.

VRCO (2003). Cavelib library, http://www.vrco.com/products/cavelib/ cavelib.html.

Web3D Consortium (2003). Extensible 3D (X3D^{TM}) Graphics. Home Page, http://www.web3d.org/x3d.html.

Web3D Consortium (2005). ISO/IEC FDIS 19777-1:2005. extensible 3D (X3D) language bindings part 1: ECMAScript, http://www.web3d.org/x3d/specifications/ISO-IEC-19777-1-X3DLanguageBindings-ECMAScript/.

Wingrave, C. A. & Bowman, D. A. (2005). Chasm: Bringing description and implementation of 3d interfaces, *Proceedings of the IEEE Workshop on New Directions in 3D User Interfaces*, Shaker Verlag, pp. 85–88.