Pablo Figueroa*
Universidad de los Andes
Colombia

Walter F. Bischof
Pierre Boulanger
H. James Hoover
Robyn Taylor
University of Alberta

InTml: A Dataflow Oriented Development System for Virtual Reality Applications

Abstract

This paper presents our research on the Interaction Techniques Markup Language (InTml). Our final goal in this work is to find ways to evolve and fit virtual reality (VR) applications over heterogeneous hardware platforms, a process we call *retargeting*. Toward this goal, we have developed a hardware-independent, component-based, formal model that describes the execution of VR applications; an XML language for describing complex and implementation-independent VR applications; a methodology for InTml-based development; a manual way to isolate and replace interaction techniques as a contribution to VR retargeting; and a set of tools for development support. This paper describes these topics and states future directions of our research.

1 Introduction

The field of Virtual Reality (VR) is now more than 30 years old, yet in several ways it is still in its infancy. After an initial overrated hype of spectacular applications envisioned by science fiction authors, and the ensuing frustration of unmet expectations, VR is finally becoming a real technology with clear solutions in specific industrial applications. Currently, there are working examples in industries such as car prototyping (DaimlerChrysler, 2006), oil exploration (SGI, 2006), military (MÄK, 2006), entertainment (Disney, 2006), phobia treatment (HITLab, 2006), and others. Many of these applications have demonstrated that VR technology has distinct advantages, but that more research is required to make it mainstream.

VR application development remains a daunting task, in part due to the lack of mature and widely accepted development tools and methodologies. A quality VR application requires the seamless integration of multiple, often novel, input and output devices. Single platform development is complicated by hardware calibration issues, software integration of specialized frameworks and libraries, and the inherently cross-cutting nature of VR technology that works against modularization. Adapting to multiple platforms is even more difficult.

Presence, *Vol. 17, No. 5, October 2008,* 492–511

*Correspondence to pfiguero@uniandes.edu.co.

This work addresses some of the issues related to the development of VR applications when a variety of hardware platforms is available, as is the case in current VR labs and development sites. Our main contributions are:

- We make the most important elements in the interface of a VR application visible at a high level of abstraction. Current representation methods of VR applications are either too formal or too close to a programming language to be understood by users, which precludes the analysis, evaluation, and improvement of interface issues.
- We define a clean separation between different software components in a VR application and their associated semantics. This separation allows us to reuse VR components, and to control and identify relationships and possible side-effects between components.
- We define a new way to transform an application from one hardware platform to another, a process we call *retargeting*. We propose a simple method for manual retargeting based on the high-level, component-based language called InTml.
- We separate two important roles in the development of VR applications, and we support such roles in a development process. One role is responsible for the overall architecture of the application. People in this role are concerned with interface issues, requirements coverage, and component reuse. The second role is in charge of fine-detail development of components and their tuning to a particular deployment environment. We consider this separation an important way to handle complexity in the development process by allowing collaboration developers to work in parallel on different issues.

We first describe previous work in the area. Later, we present the main concepts of InTml, its operational semantics, supporting tools, implementation details, and software engineering issues. Next, we describe the concept of VR application retargeting, as the process of accommodating VR applications to different VR setups while getting the most from each environment. Finally, we present examples, lessons learned, conclusions, and future work.

2 Background

There are many toolkits for VR development, with different scope and complexity. Some allow users to configure a wide spectrum of aspects, while others hide some decisions from developers 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. Reviewing the way programs are developed in several VR toolkits (Shaw, Liang, Green, & Sun, 1992; VRCO, 2003; SGI, 2003; Blach, Landauer, Rosh, & Simon, 1998; Sense8, 2000; CMU, 1999; Bierbaum et al., 2001; Web3D Consortium, 2003; Taylor et al., 2001; Sastry, Boyd, & Wilson, 2001; Allard et al., 2004), one sees 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 languages tend to limit support for devices and novel behavior, precluding the evolution of VR applications written in such languages. In contrast, we present a VR development environment and methodology that provides solutions to the following issues: application architecture, division of responsibilities during VR development, implementation and integration of novel techniques and behavior, and high level description to ease communication between developers and users.

One of the main problems of current environments, APIs, and toolkits for VR development is the proposed structure for application-specific code. Some environments such as the ones in Shaw et al. (1992), CMU (1999), and 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 interaction techniques, event correlation, and modifications to output data structures. Functional decomposition can be used in order to improve readability, but the reusability of such schemes has not been proven, and in general it is not achieved, given the lack of support from development environments. Other environments such as Bierbaum et al. (2001) and SGI (2003) add new behavior at the beginning and at the

end of the main 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, events from pressed keys and a composite type for mouse button 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 (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 main 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 data flow can better define relationships between different behavior components in the system, and it clearly exposes component dependencies. Some systems (Carey & Bell, n.d.; Web3D Consortium, 2005; Avango, 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 (Battacharyya, Murthy, & Lee, 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 does 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 a 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 previous solutions that decouple device reading from simulation execution (Shaw et al., 1992) and even distributed solutions such as Allard et al. (2004), with some limitations.

A data flow architecture also allows us to consider dynamic and static scheduling algorithms for machines with several CPUs. This approach cannot be implemented in current data-flow-based solutions such as VRML and X3D, due to intrinsic limitations on the order of execution of their components in a program. Kwok and 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 (Blach et al., 1998; Web3D Consortium, 2003; Dachselt, Hinz, & Meiner, 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 data flow 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 (Web3D Consortium, 2003; Autodesk, 2006) concentrate mostly on geometry and on the PC-based interaction environment. Others, such as described in Wingrave and Bowman (2005) use state machines as a design abstraction, which we believe is very powerful although more complex for nonprogrammers. The same is true of hybrid languages such as the one in Smith and 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 have proposed portable ways for describing VR applications (Massó, Vanderdonckt, Simarro, & López, 2005; Dachselt et al., 2002), 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 and Jacob (2001), the user-centered approach in Neale, Cobb, and Wilson (2002), a UML-based approach (Kim, 2005), and a methodology based on a hybrid language (Sastry et al., 2001). 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 (Humphreys et al., 2001; Eldridge, Igehy, & Hanrahan, 2000; Nishimura & Kunii, 1996; Molnar, Eyles, & Poulton, 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 (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.

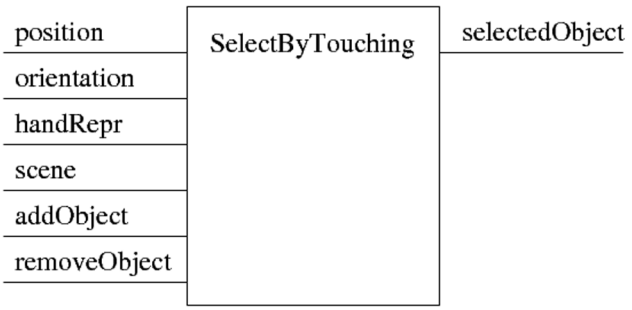

Figure 1. *An example of a filter: select by touching.*

3 InTml

3.1 Concepts

Here we update our earlier description of InTml (Figueroa, Green, & Hoover, 2002). We define VR applications as data flows of interconnected filters, described by the Interaction Techniques Markup Language (InTml). Filters are the building blocks that describe the standard connections for any of the following entities: input or output devices, interaction techniques, object behavior, animations, geometric objects, and other media objects. Details about gathering information from devices or about object behavior code are described at a lower level of abstraction through the use of general-purpose programming languages. Further, geometry or other media types related to VR objects are produced with any of the available tools for that purpose, such as Maya (Alias Wavefront, 2003), 3D Max (Discreet, 2003), or Blender (Blender.org, 2003). InTml is an integration language for all of the elements involved in VR applications. It enables the designer to concentrate on the architecture of the application, without dealing with too many details. For example, while data-flow-based languages such as VRML focus on description of geometry and animation, InTml focuses on the explicit identification, separation, and integration of application-specific behavior, object behavior, and events from input devices. Geometry is something that is described at a lower level in a loadable format, and InTml refers to it as a reference to an object. The same can be applied to sound or haptic content.

A filter represents a device, interaction technique, behavior, or content in a VR application. Its interface is defined in terms of input and output ports, which are the type of events it can receive or produce, respectively. Some input ports can be considered parameters, or ports that receive information only once at application startup. A filter can have an internal state, which is important in order to model complex filters, but this low-level concern is abstracted away from architecture level. Figure 1 shows a way to represent a filter, *SelectByTouching,* with input ports on the left of a box and output ports on the right. In this example, the output port is a selected object from the scene, and the input ports are the geometry used as hand representation, the current position and orientation of such an object, the scene of objects to pick from, and the events that inform about objects added or deleted from the scene. Note that the input ports for the hand representation and the scene can be considered parameters because they do not change once they are assigned.

The computation of a filter is divided into three main stages:

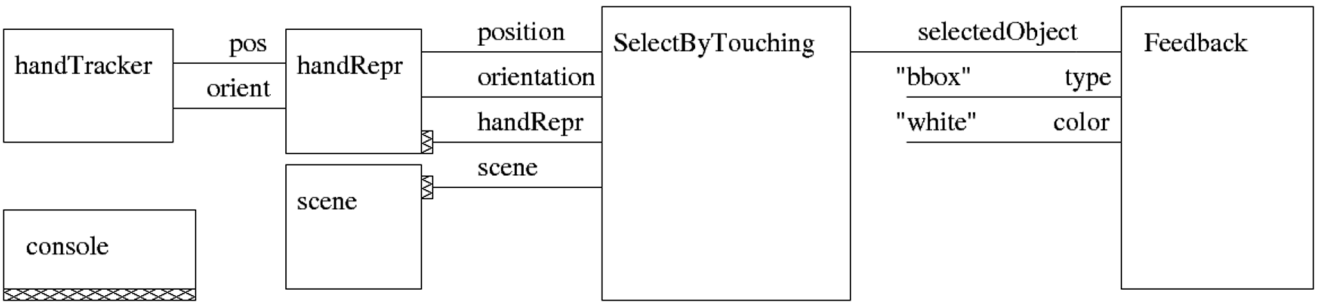

Figure 2. *Simple application. Touching objects with a virtual hand.*

- Data collection. 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 on 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.

For example, in the first step, *SelectByTouching* could receive one object through *handRepr* and one through *scene,* which may be considered parameters. Later steps will receive events through *position* and *orientation,* which trigger the filter's computation of a new selected object.

VR objects represent identifiable pieces of content in the virtual environment, elements that can be seen, heard, or touched by the user. An application is a set of interconnected filters that meet certain user requirements. Figure 2 shows a simple application that allows a user to move a virtual hand with a tracker and touch virtual objects. Filters can also be sent as events through the data flow, which is shown as an output port with a special decoration (two examples are *handRepr* and *scene* in Figure 2). An *input device* is a filter that sends events of a certain type, which come from a physical device, to the data flow. An *output device* is a filter that receives objects in the scene and renders them. We also use a special decoration for an output device (e.g., *console*) in order to avoid the clutter of lines from all objects to the output device.

In this example, a device (*handTracker*) gives position and orientation information to an object (*handRepr*). *SelectByTouching* receives the actual *handRepr* and *scene* objects, and any changes in position or orientation from *handRepr*. Once a collision is detected, the collided object is passed to *Feedback,* which shows a white bounding box around the object. At the end of each execution step, *console* will render all objects in the scene (both *handRepr* and objects inside *scene*). Filters and applications are independent of any particular software framework and hardware, so the designer does not have to be limited by platform specific elements, and the developer is free to reorganize the implementation in order to improve the behavior of the application on a particular platform.

3.2 Operational Semantics

A formal description of the InTml model has been developed in the Z notation (Spivey, 1992), which is also language and platform independent. In Figueroa, Hoover, and Boulanger (2004), we describe, using this notation, the concept of a filter, how filters can be composed into filters that hide complexity, how filters process information at any time step, how information gets propagated thoughout a data flow of filters, and controlled ways to change the data flow at runtime. From this model one can derive 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 the typical restrictions imposed by stan-

dard function calls, where parameters are always mandatory. For example, the filter *SelectByTouching* in Figure 2 may have input ports for updating the scene, but since it is not required in this example, they are not shown, nor they are connected.

- Different filter types, such as devices, interactive content, and behavior, are first class citizens of this description. Apart from some details in the execution of object holders and content objects, all filters appear equal from an execution point of view. Again, in Figure 2, devices are at the same level and look similar to both content and behavior filters.
- A time step defines a time interval in which all events from input devices are considered simultaneous, independent of the particular generation rates of each device. For example, with a 200 ms interval, a filter receiving information from both a PHANToM running at 1000 Hz and a tracker running at 120 Hz could receive up to 200 PHANToM events and up to 24 tracker events, all to be handled simultaneously.
- Filters execute at most once in a time step, and all information they produce is considered simultaneous. A topological sort is used to find a sequential execution order. For example, the sequence [*wand, scene, timer, 6DOF2Ray, SelectByRay, MoveToObject, viewport*] in Figure 10, covered below, could be parallelized into the sequence [{*wand, scene, timer*}, {*6DOF2Ray*}, {*SelectByRay*}, {*MoveToObject*}, {*viewport*}], without affecting the inputs and outputs of any filter.
- Data flow cycles are allowed in an application, but they are broken temporally so back-flowing data is delivered at the next time step. For example, if the filter *MoveToObject* in Figure 10 starts changing the position of the viewpoint at time step i, it will receive such changes in its p and q ports at time step $i + 1$. This cycle in Figure 10 allows *MoveToObject* to receive the actual viewpoint position, which may be also affected by other filters.
- An object holder is a placeholder that defines the connections that a content object should have, once it is assigned to such a position in the data flow. For example, consider the object holders *current* and *previous* in the right-hand part of Figure 9, covered below. An object holder has a special input port that binds to the contained object (i.e., *currentObject* and *previousObject*). Events received in other ports (i.e., events from *setBBCurrent* or *setColorCurrent*) are propagated to the contained object, and events generated from the contained object are propagated to registered filters (not used in this example).
- Content objects can be related in structures that are not evident from the InTml data flow (i.e., in a scene graph), which may require rule checking and change validation. For this reason, all changes in objects are queued until the end of a time step. For example, in Figure 2, *handRepr* can be attached to an avatar inside *scene* that may restrict its movements. Although each filter executes at most once in a time step, output events from *handRepr* and *scene* will appear one time step later than its outputs. In other words, all inputs are collected in the first time step, and all outputs are generated in the second time step. The time between time steps can be used for rule checking at the system level.

3.3 InTml Tools

We have prototyped a set of basic tools for both InTml designers and developers. This set provides basic support for the construction of new InTml applications, and it represents the foundation for a future InTml IDE. In the following, we describe the functionality of the translator to HTML, the InTml compiler, the InTml checker, the InTml visual editor, and the InTml library.

3.3.1 The Translator. The InTml-to-HTML translator is a set of XSLT and AWK scripts that generate indexed HTML documents from a set of InTml documents. It provides an easy-to-read reference of InTml documents, which describe filter classes and applications. We organize filter classes by either file name or user-defined categories, so there are several ways to find a concept. Figure 3 shows how the HTML output appears for a particular filter class and for the view by file name.

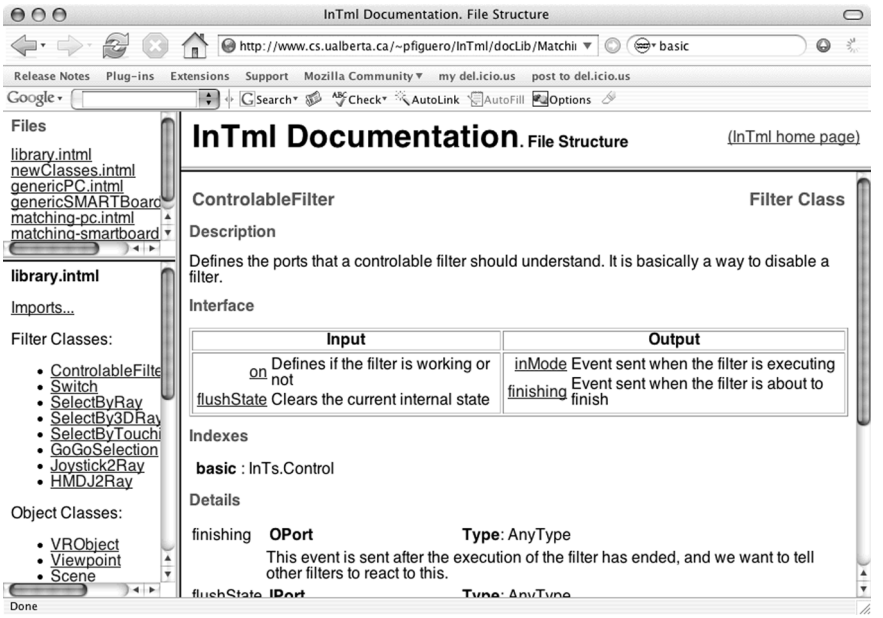

Figure 3. *InTml translator to HTML.*

3.3.2 The Compiler. The InTml compiler is a translator that produces Java or C++ code from InTml files. Each InTml class becomes a programming language class, which fulfills contracts from the core framework and allows localized editing and generation of repetitive code.

3.3.3 The Checker. The InTml checker validates a set of InTml documents and reports semantic problems to developers. Currently, it is implemented as the first stage of the compiler. Some of the problems currently detected are:

1. Referenced types that are not found;
2. Name convention errors. For example, an application's name cannot include dots in order to avoid problems with conventions for package names;
3. Type correspondence problems in filter-to-filter connections;
4. Input or output ports not found in a filter.

3.3.4 The Editor. The InTml visual editor (Mejia, Figueroa, & Hernández, 2005) integrates two applications in order to allow creation of InTml applications (InTml Application Editor) and novel filter classes (InTml Filter Class Creator). The application editor shows the current state of an application and provides an easy way to add or remove filters, to create or remove connections, and to run the application. The filter creator allows designers to graphically define novel filter classes. Such classes will later be completed by developers, so designers do not have to worry about the specific details of the filter's execution code, and they can concentrate on the high-level decisions related to input and output ports. Figure 4 shows the visual editor.

3.3.5 The InTml Library. A library of filters facilitates development, both at the level of a family of applications and at the level of the entire development environment. At present, our generic library contains 20 filters that support the input and output devices in our lab, 3D objects that may be changed through rigid transformations, and simple animations. Each family of applications defines its own set of filters, which may be reused by applications within the family, and eventually become part of the generic library. Reuse ratios within a family are considerable, as will be covered in Section 5, and we hope to improve our generic library as more applications are developed in order to improve further reuse levels.

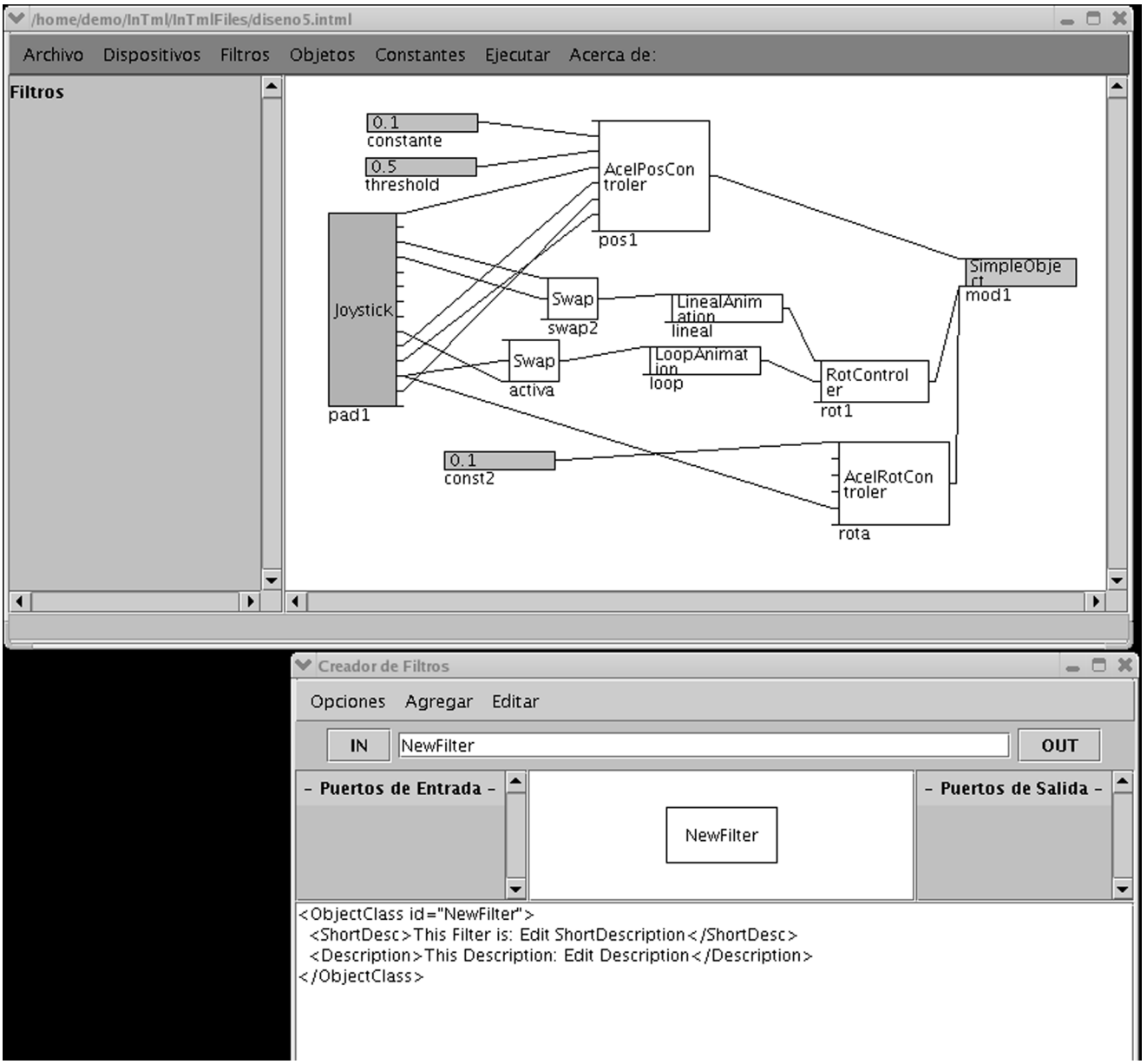

Figure 4. *InTml visual editor.*

3.4 Implementation Details

We currently have two implementations of the InTml runtime environment, one in Java and one in C++, although most of our current applications are on top of the C++ implementation that is available at sourceforge (Figueroa, 2007). There are three packages in C++, one with the core framework classes, one for the loader, and one with the dynamically loaded filters.

Figure 5 shows a class diagram of the InTml core framework. The abstract class *Filter* defines the basic behavior of an application's filter. *Device, InT,* and *GObject* are the main *Filter*'s subclasses and they can be

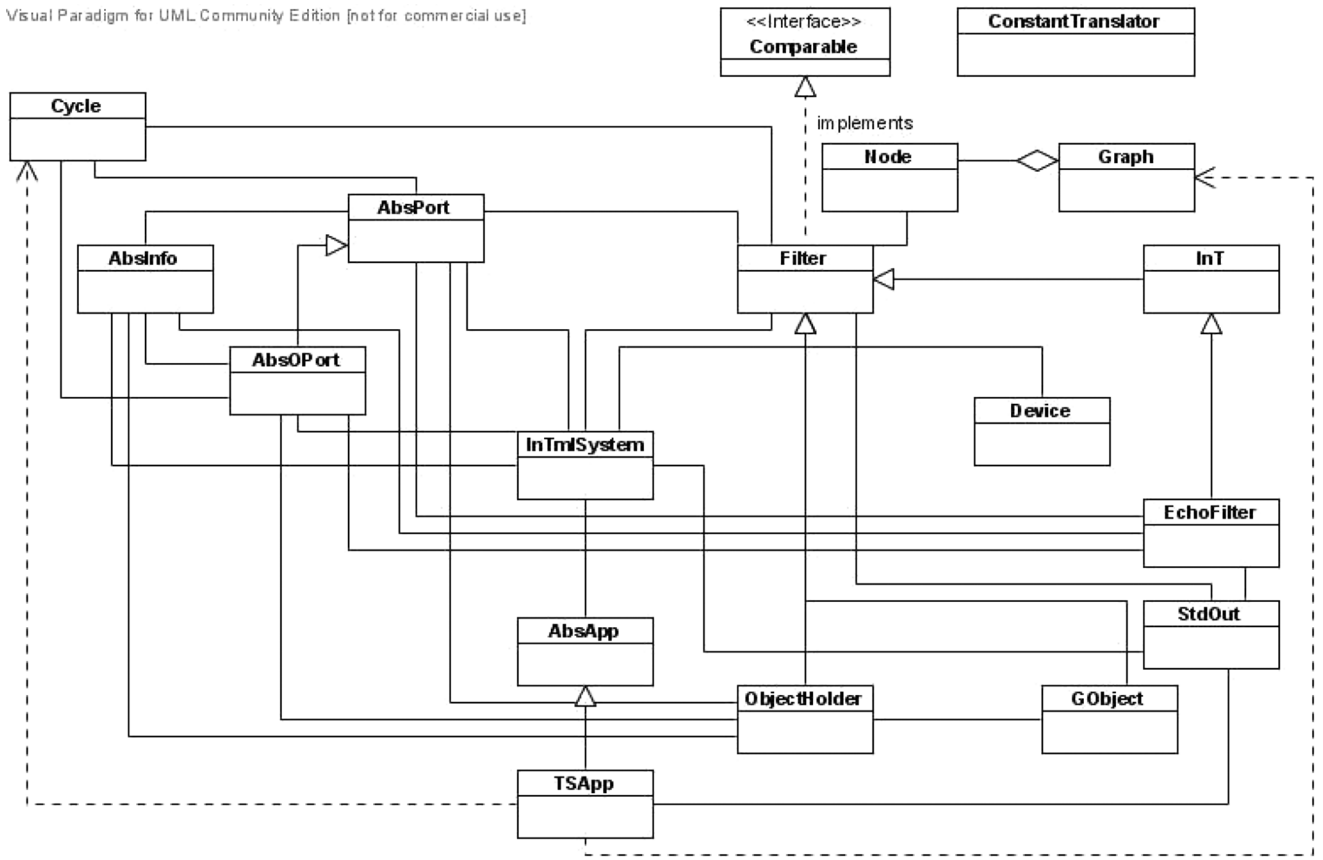

Figure 5. *InTml framework classes.*

subclassed in order to support novel devices, behaviors, or content types. The basic behavior of an application is composed of the *TSApp* and *InTmlSystem* classes. *TSApp*, or Topologically Sorted Application, represents an application that runs in just one thread and organizes the execution of its filters by means of a topological sort. This order guarantees that no filter is executed before its predecessors in the data flow. *InTmlSystem* is an abstract class that provides a standard interface for platform-dependent issues. Ports (*AbsPort* and *AbsOPort*) transport instances of *AbsInfo* and its subclasses, which handle different information types for events and their generation time. *ObjectHolder* defines the InTml mechanism with the same name, and contains a *GObject* and a set of ports for this purpose. *ConstantTranslator* is a generic class that allows marshalling and unmarshalling of simple types. This behavior is useful when an InTml application is being read from its XML-based representation.

Figure 6 shows a class diagram of the loader. There are three main classes: *InTmlDynamicApp, AbstractFactory,* and *MainInTmlLoader*. *InTmlDynamicApp* represents an application that may be dynamically loaded from its XML-based description. *AbstractFactory* implements the design pattern (Gamma, Helm, Johnson, & Vlissides, 1994, p. 87) that allows us to abstract the particular type of scene graph in use. It uses abstract classes (*Base3DObject*, *BaseScene*, *BaseViewpoint*, and *BaseInTmlSystem*[1]) that hide implementation-dependent details, which are represented by subclasses in the diagram. Geometry represented by *Base3DObject* is loaded from a

[1]We could create a *BaseApp*, but we decided to reuse *TSApp* from the framework for this role.

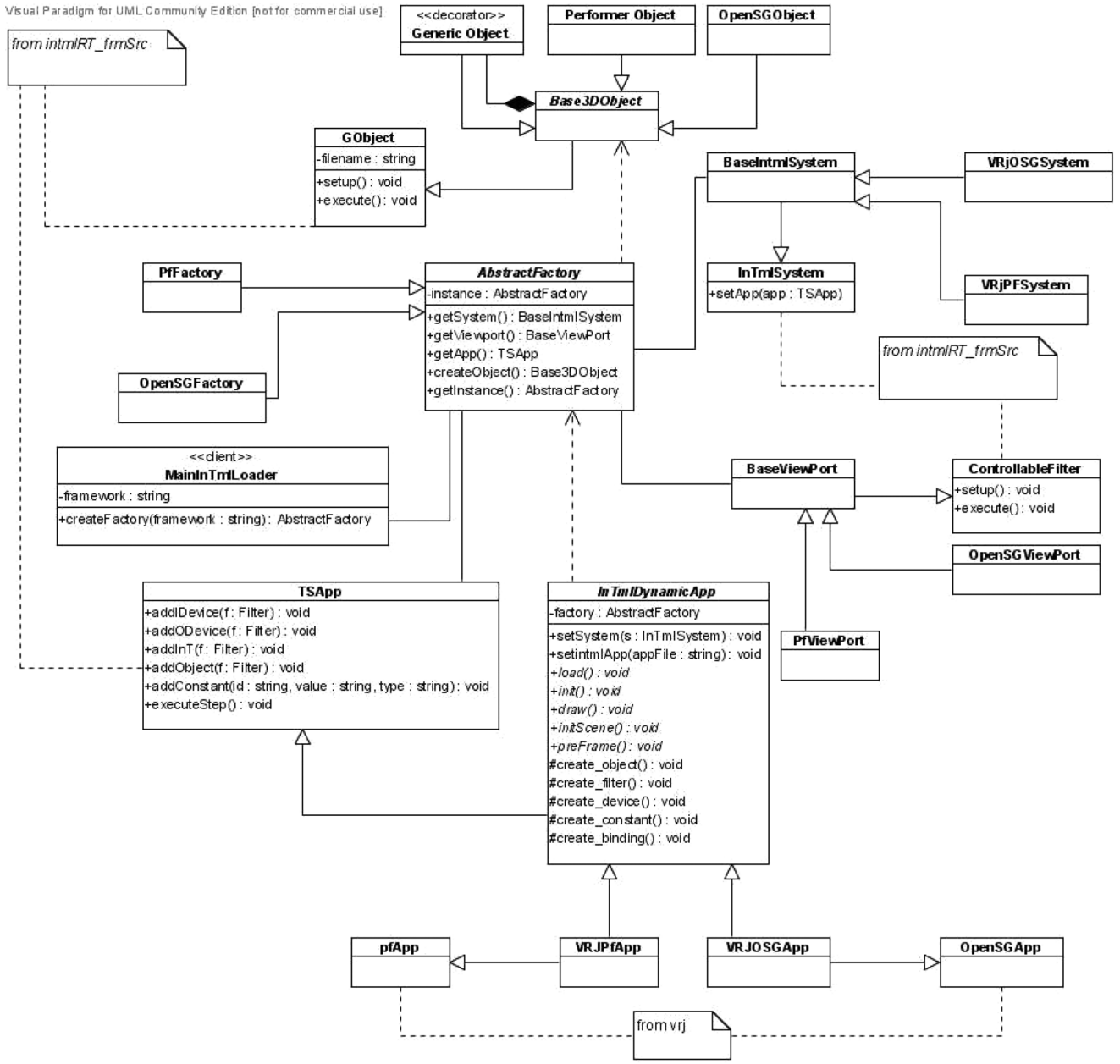

Figure 6. *InTml loader classes.*

file, in formats supported by the underlying scene graph. Currently each object has to be in a separate file, but it is easy to load several objects from a file by reading their ID's from such a file.[2] *MainInTmlLoader* is in charge of loading and creating relationships between the other two classes.

Dynamically loaded filters are subclasses from the core framework, usually from *Device, InT,* or *GObject,* and their implementations are platform-dependent. For example, in our C++ version of InTml, the *Fob-*

[2]Formats such as VRML support IDs at certain nodes in the scene graph.

Tracker2 class inherits from *Device* in the core framework, represents a Flock of Birds tracker with two tracking devices, and it is implemented by using VRPN (Taylor et al., 2001). Among others, we have implemented filters for devices (*FobTracker2, Gamepad, Glove5DT, Joystick, Keyboard, Mouse, Wheel, Timer*[3]), simple navigation techniques (*WalkNavigation*), and simple object animation (*LinearAnimation, LoopAnimation*). Devices are currently implemented in several operating systems, and we use the distribution capability of VRPN in order to access them from one application in Linux.

3.5 Implementation of a New Filter

A filter is implemented as a dynamically loaded module that inherits basic behavior from classes in the InTml Framework, and reuses third-party code as necessary. In order to implement a new filter in the current system, it is necessary to perform the following steps: define the interface, create initial code stubs, redefine important methods, identify third-party libraries, and add support in makefiles. The following paragraphs give some details on this process.

Initially, an interface of a filter is defined in the InTml language. The interface consists of all required input and output ports, with clear names and types. Usually, types in input and output ports are defined for an entire set of filters, so it is customary to include filters inside a library in order to share definitions and a name space. It is also customary to fill specific documentation tags that describe the filter's purpose and use of its ports.

Once the interface is defined, it is possible to create the initial code stubs by two means, either by generating a basic stub with the provided tools, or by copying and modifying an existing filter implementation (header and source files for a filter's class). Once the stubs are created, a developer should redefine the following methods: A constructor without arguments, `createPorts`, `setup` for initial setup and values in input ports, `execute` for the actual behavior, and optionally `loadProperties` and `saveDefaultProperties` for loading parameter values from configuration files. Two extra C functions control dynamic loading and should also be modified, but they are almost identical in all filters and their modifications are trivial.

Code in the `execute` method could be based on third party libraries, in order to reuse preexisting solutions. If some initialization is required and the library is not used in any other filter, it is possible to write such code in the `setup` method. If several filters require the same initialization code, it should be written at the application or loader level. However, we have found only two examples of such a requirement, for the scene graph creation and for socket initialization, so most of each filter's code is embedded inside its class. Finally, makefiles should include new dependencies if necessary. Our current implementation supports CMake and autoconf tools, and in the case of autoconf it is possible to define the dependencies at the level of each filter.

3.6 Filter Dependencies

Table 1 shows dependencies per filter and main modules in an InTml application. An application has at least an instance of `MainInTmlLoader`, `InTmlDynamicApp`, and a scene graph, so it requires at least VRJuggler (Bierbaum et al., 2001), xerces-c (Apache, 2007), and either OpenSG (Reiners & Voss, 2007) or Performer (SGI, 2003). Other filters add more libraries and dependencies, such as VRPN and boost (Rivera, Dawes, & Abrahams, 2007). Some of these libraries could be replaced or complemented with others; for example, several devices require VRPN, but others could use other libraries, such as X11 for keyboard and mouse handling. It is also interesting to notice limitations of the current implementation: `WalkNavigation` is a technique defined just for OpenSG, which may be extended to the other supported scene graphs. Finally, this table shows the necessary work to be made if a library should be replaced.

3.7 Software Engineering Issues

InTml has features that support software engineering concerns, such as team development, complexity

[3]A timer is considered an output device that gives information about the current time.

Table 1. *Dependencies Between Filters and Third Party Libraries*

Filters	Dependencies
MainInTmlLoader	OpenSG, Performer, VRJuggler
BaseInTmlSystem	VRJuggler
InTmlDynamicApp	xerces-c, iostream
WalkAnimation	gmtl, OpenSG
FobTracker2, Gamepad, Glove5DT, Joystick, Wheel	VRPN, gmtl
AbstractFactory, Base3DObject, BaseScene, BaseViewpoint	gmtl, boost
Keyboard, Mouse	X11, gmtl
LinearAnimation, LoopAnimation	gmtl
Timer	

hiding, and reusability. These issues may be classified as methodology-related and language-related.

From a methodological point of view, application concerns are divided between two roles, designers and developers. Designers are in charge of the overall design of the application. They know about InTml, its semantics, and available reusable components. Developers are in charge of the fine-grain details inside components; they know how components can be implemented on top of available frameworks and libraries. Each role develops complementary tasks: designers are closer to end-users, while developers are closer to the programming and hardware details of the solution.

The first version of a VR application, targeted to a particular hardware platform, is created by pursuing the tasks in Figure 7. This process guarantees a clear division between the architecture of the solution and the implementation of each component in the architecture. Once a first version is completed, new versions can be created by retargeting the application to other hardware platforms, a process similar to the one described previously.

Content components, such as geometric models for objects, special graphic effects, sound, or haptics, are designed with the aid of third-party tools. It is necessary that all created media types can be understood by the foundation framework where the InTml application will run. Since InTml can be implemented on top of several foundation frameworks, it is possible to discover platform limitations in the process of developing a new application. Such limitations can be detected by developers while trying to create new components. In this case, developers and designers can compromise on a solution that both satisfies requirements and minimizes changes in the foundation framework. However, the more mature an InTml implementation is, the less these changes will be required.

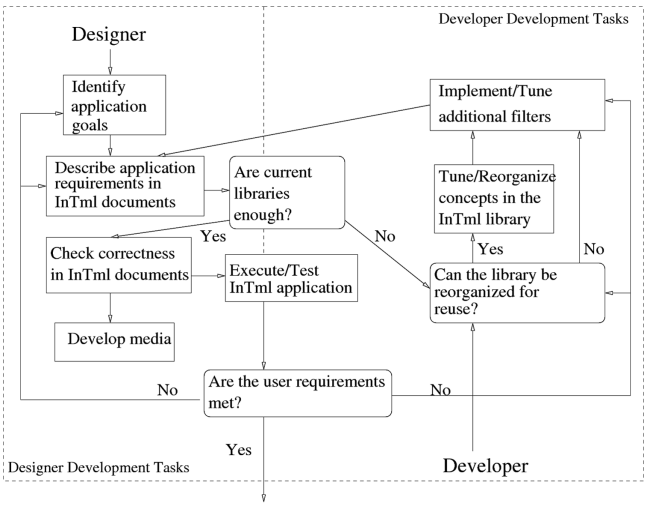

Figure 7. *InTml-based development process.*

From a language point of view, reusability and complexity hiding are accomplished by the mechanisms of filter libraries, task views, and composite filters. The concept of a library was described in Section 3.3.5, and

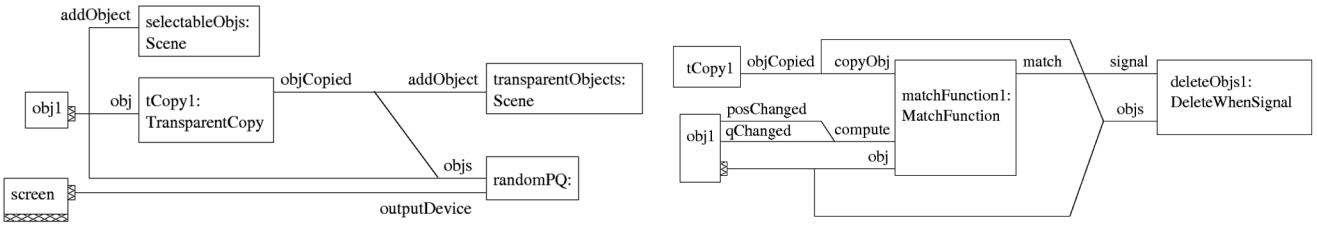

Figure 8. *Two views of a matching application: object creation and matching function.*

the following paragraphs give some details on reusability and complexity.

Complex VR applications could be represented by several dozen filters with multiple connections, which may obscure the rationale of a filter, a particular connection, or the overall architecture of an application, if presented together. For this reason, we define an InTml application in terms of Task Views, which helps to correlate requirements to design decisions. A Task View is a subset of all filters and connections in an application, which solves a particular requirement. The task description adds semantics to a view, so the rationale of filters and connections is easier to understand. For example, Figure 8 shows two possible views of an application that has been developed for a user study (Figueroa, Bischof, Boulanger, & Hoover, 2005), in which subjects should match the position and orientation of three objects by moving and rotating replicas of them. The object creation view shows how a transparent replica of an object *obj1* is created through a filter called *tCopy1,* and how both objects are randomly located and oriented on the screen by means of a filter called *ramdomPQ.* The matching view shows the filter *matchFunction1* that emits a signal once both an object and its replica are close enough, and the filter *deleteObjs1* that deletes such objects once they match. Note how the filter *tCopy1* that produces the replica is shared between views.[4]

We have also designed an InTml encapsulation mechanism called composite filters. A composite filter is a subset of encapsulated filters and connections that represent a complex interaction technique or behavior, which may be reused as a whole and treated as a black box. This mechanism allows developers to reuse filters, to hide complexity of the overall solution, and to create complex filters that may involve content and behavior. For example, each of the three objects in our matching application is associated with an instance of a composite filter that does matching for one object. Composite filters are useful for encapsulating complex interaction techniques, such as Go-Go selection in Figure 9. Go-Go (Poupyrev, Billinghurst, Weghorst, & Ichikawa, 1996) is an interaction technique to lengthen the user's virtual arm for reaching distant objects. It is composed of simpler filters for collision detection and interaction feedback by means of visual changes of the currently selected object. The behavior of Go-Go is tuned by the external parameters D as the minimum arm's distance for the lengthening effect and K as an attenuating factor between 0 and 1.

4 VR Application Retargeting with InTml

Retargeting is a term used in the compiler community that refers to the adaptation of code to the characteristics of a particular CPU, using all capabilities of a hardware architecture. Retargeting is concerned with methods for transforming an implementation on an initial platform to the most suitable implementation on a target platform. In compiler retargeting, it is assumed that there is enough information for doing such transformation.

The concept of retargeting is similar to the concept of porting, most commonly used in the software engineering community. However, traditional porting tech-

[4]The structures in these diagrams are replicated three times in our application, in order to handle three distinct objects in the application.

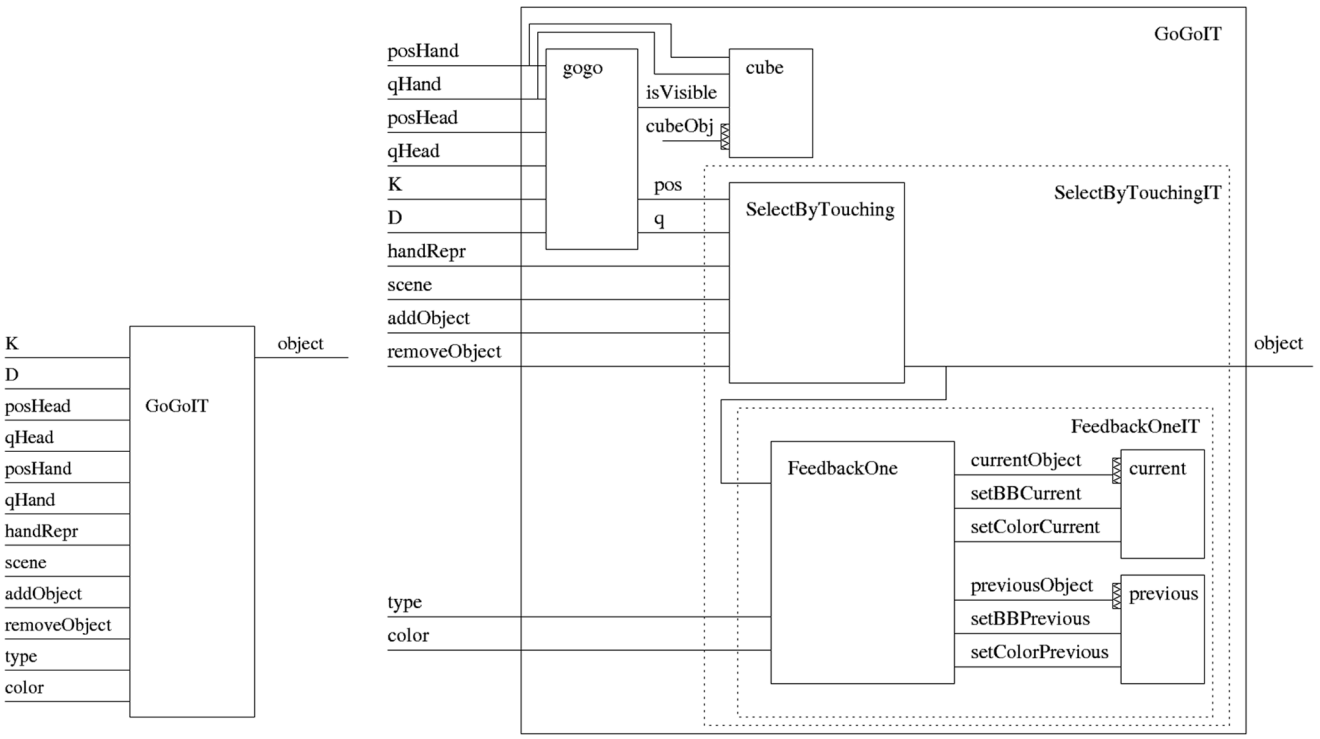

Figure 9. *GoGoIT, a composite filter.*

niques do not accommodate to the particular features of an installation. For example, object oriented frameworks allow developers to port applications to all platforms the framework has been implemented in. However, a framework usually encapsulates only features common to all implementations, so developers cannot use special features of a particular platform. Furthermore, the purpose of a framework is often to hide platform details, thus making it difficult to support platform-specific features.

VR retargeting can be understood in several ways. The most common case is based on replacing or translating events from one device to another, for example, replacing mouse events with joystick events, or translating mouse and keyboard events to 3D position and orientation from a 6 DOF tracker. Current VR development tools support this type of translation, usually with the aid of input event abstraction (Reitmayr & Schmalstieg, 2001; Taylor et al., 2001). However, these types of changes do not handle more complex and interesting cases of retargeting, such as the following:

- Replace selection techniques with ones more suited to the new hardware platform.
- Simplify or accommodate content to the capabilities of the available output devices.
- Reduce or simplify the tasks that an application can perform when retargeted to a less powerful platform.

Our approach for VR application development enables the retargeting of VR software to the particular capabilities of a VR environment. Since a VR application is described as a set of separated objects, behaviors, and devices, it is possible to accommodate its implementation to the particular characteristics of an installation. Platform specific components like devices and behaviors are easily recognized, so it is possible to transform an application from one installation to another by replacing devices and behavior while keeping support for the same tasks. This set of complex manipulations that we refer to here as retargeting is novel in VR, and it opens possibili-

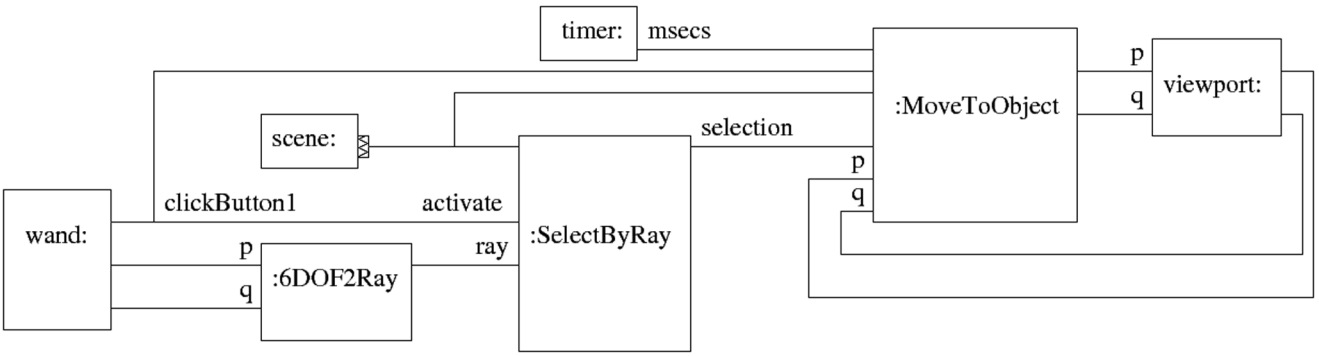

Figure 10. *InTml design for navigation in a CAVE.*

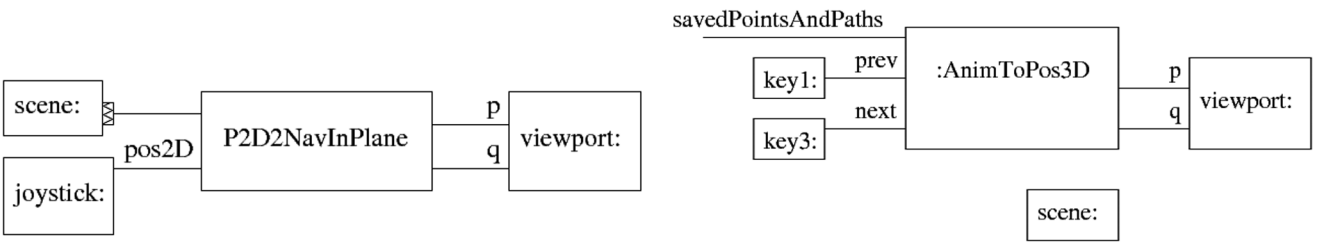

Figure 11. *InTml design for navigation on a PC and a cell phone.*

ties for VR development where several hardware configurations can be tested and compared.

For example, suppose that we are interested in navigating a small but complex VR office, showing available information about near objects. We want to use 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 of interesting and different implementations for this task on each platform, as follows:

- In a CAVE, a user can navigate to an interesting object by pointing to and selecting it. The system should compute a path from the current viewpoint position to a position in front of the object. This technique is similar to fixed-object manipulation (Bowman, Kruijff, Joseph, LaViola, & Poupyrev, 2004, p. 215) with extra behavior for path planning.
- On a PC with a joystick, one could use a navigation technique that resembles the WALK mode in VRML. This mode features collision detection between the avatar and objects in the environment.
- On a cell phone, due to computation restrictions and limitations on the input device, it is more convenient to select prerecorded viewpoints and paths. It could be also important to reduce the complexity of the scene as much as possible.

Figures 10 and 11 show the InTml diagrams of these navigation techniques. Although the design rationale for interaction techniques in each platform can be formalized further, in order to identify the most suitable one for a particular application and platform, this example allows us to show some features of our work. First, the formal description of an interaction technique is more precise than plain text. Second, we can describe very different interfaces with the same formalism, which may be used for comparison at early stages of the development process. Third, it is possible to describe complex retargeting of interaction techniques, much more complex than simple device replacement. Fourth, imple-

mentation details such as scene level of detail or actual algorithms are hidden at the architectural level, making the design easier to understand.

Since VR applications are still very diverse, and 3D interaction techniques are not standardized, we cannot automatically decide retargeting options for a particular hardware setup. The next best thing is to make it easier for designers to manually make retargeting decisions. Since the variations from one platform to another are unknown, the retargeting process takes place at design time, reshaping an InTml-based application from one platform to the particular features of a new environment. This process also requires support for InTml in diverse environments, which is why we created both Java and C++ ports, in order to create executable programs from InTml descriptions and, at the same time, to cover a wide range of hardware platforms.

5 Examples and Lessons Learned

We have designed 15 families of simple VR applications, some of them with up to four different hardware platforms, giving us insight on how we can use our notation. We have implemented 10 applications, which are now part of our Lab demos, and we have around 10 more in development. Most of these applications were created with academic goals in mind, so further work is required to improve robustness and overall software quality. Nevertheless, InTml has proved to be an excellent environment for reuse and application retargetability.

In Figueroa et al. (2005), we developed a simple matching test application, as a proof of concept for a methodology for partial (i.e., hardware and interaction techniques alternatives) exploration of the design space of a virtual reality application, based on the creation of reusable components and a standard evaluation of alternatives. This application displays three objects and their copies. The user moves the copies so that they match the 3D position and orientation of the originals. Four interfaces were implemented that use the following devices: a keyboard and mouse, a SMARTBoard (SMART, 2007), an HMD and a joystick, and a SpaceMouse (3D Connexion, 2007). The four flavors of the application that we developed shared over 60% of their Java code, despite the differences between the interfaces they provide. It was also possible to create application-specific metrics among interfaces, so formal user studies could be performed in order to compare heterogeneous implementations. Results showed that the interface affected the user's performance and preferences, as could be expected.

In Figueroa and Mejia (2004), we illustrated our development methodology and environment with a threshold application for medical data visualization. In this case, we designed an application in InTml and we implemented it in Java and C++, in order to show the portability and platform-independence of InTml. In Mejia, Figueroa, Hernández, & Rosa (2004), we illustrated the implementation of the InTml framework of several simple applications with high levels of reuse between them that use two trackers to accomplish the following tasks: orbit around a Performer-based or a VTK-based object, save the current viewpoint, and zoom the current object.

We are currently migrating a 3D visualization application to InTml (Boulanger, Garcia, Badke, & Ryan, 2006), in order to support distributed users with heterogeneous interfaces. We continue to evaluate InTml in diverse domains, such as architectural walkthroughs, medical data visualization, and edutainment.

6 Conclusions and Future Work

The InTml comprises results in several areas, such as event-based architectures, formal models, team development, and high-level languages for VR. We define VR application retargeting as the process of modifying an application in a particular VR setup in order to fit the advantages and limitations of a second hardware environment. These results allow us to design applications without the limitations in interaction of a particular development environment, and to explore more thoroughly the design space in a particular VR solution. We believe our approach allows us to create a rich development environment for VR applications, which encour-

ages collaboration between VR programmers and VR designers. This collaboration allows the creation of more complex and compelling applications in novel fields and contributes to the spread and maturity of the VR technology. We have identified several areas of future development, described in the following paragraphs.

Some VR designers, such as artists, are interested in a stable version of capabilities, instead of the potential growth that a tool can have. We plan to define a documented library of filters that this type of users can use for their own purposes.

Rapidly changing VR hardware also presents challenges for VR development. The advantages and limitations of specific VR devices are not well known, and users often make incorrect assumptions about their use. Moreover, some devices are very fragile and difficult to set up, which makes some platforms cumbersome to use. Such problems have led designers to prefer some devices, despite limitations in their functionality or degrees of freedom. We are working on more reliable VR-related devices, so our design space for VR applications can grow effectively.

We have found that programmers have difficulty adhering to the rules related to the development of new filters, in particular those that encourage filter reuse. An inventive developer can create several filters that work together to build a novel application, but that cannot be easily reused to build a different application. We are working on the methodology and guidelines for filter development and its support in development tools.

Finally, we would like to expand the domain of InTml applications to applications in the mixed-reality spectrum, such as augmented reality and tangible interfaces. This will require a more comprehensive library and more work on the integration of novel devices and existing libraries.

Acknowledgments

Parts of this work were developed with funding from NSERC, iCore (Canada), and Universidad de los Andes (Colombia). We would like to thank the students Carlos Diaz and Jose Ferreira for their collaboration in some figures of this paper, and our reviewers for their constructive comments and suggestions.

References

3D Connexion. (2007). *Spacemouse plus*. Available at http://www.3dconnexion.com/product/3a3.php.

Alias Wavefront. (2003). *Maya*. Available at http://www.aliaswavefront.com/en/products/ maya/index.shtml.

Allard, J., Gouranton, V., Lecointre, L., Limet, S., Melin, E., Raffin, B., et al. (2004). Flowvr: A middleware for large scale virtual reality applications. *Euro-par 2004 Parallel Processing,* 3149, 497–505. Heidelberg: Springer.

Apache. (2007). *Xerces C++ parser*. Available at http://xerces.apache.org/xerces-c/. (The Apache XML Project).

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

Avango: A Distributed Virtual Reality Framework. (2000). Available at 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*. Berlin: Springer.

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,* 89–96.

Blach, R., Landauer, J., Rosh, A., & Simon, A. (1998). A flexible prototyping tool for 3D real-time user interaction. *User-interaction, Proceedings of Virtual Environments,* 54-1-54-10.

Blender.org. (2003). *Blender*. Available at http://www.blender.org/.

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)*. The Netherlands: TU Delft. Available at http://www.eccomascfd2006.nl/.

Bowman, D., Kruijff, E., Joseph, J., LaViola, J., & Poupyrev, I. (2004). *3D user interfaces: Theory and practice*. Reading, MA: Addison-Wesley.

Carey, R., & Bell, G. (n.d.). *The annotated VRML 97 refer-*

ence, chapter 2.10. Reading, MA: Addison Wesley Professional.

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

Dachselt, R., Hinz, M., & Meiner, K. (2002). Contigra: An XML-based architecture for component-oriented 3D applications. *Proceedings of the Seventh International Conference on 3D Web Technology,* 155–163.

DaimlerChrysler. (2006). *The DaimlerChrysler virtual reality center*. Available at http://www.daimlerchrysler.com/dccom/0-5-7154-1-14250-1-0-0-0-0-0-8-7145-0-0-0-0-0-0-1.html.

Discreet. (2003). *3D Max*. Available at http://www.discreet.com/products/3dsmax/.

Disney. (2006). *Disneyquest indoor interactive theme park*. Available at http://disneyworld.disney.go.com/wdw/entertainment/entertainmentDetail?id=DisneyQuestIndoorInteractiveThemeParkEntertainmentPage&bhcp=1.

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,* 443–454.

Figueroa, P. (2007). *InTml development tools*. Available at http://sourceforge.net/projects/intml.

Figueroa, P., Bischof, W. F., Boulanger, P., & Hoover, H. J. (2005). Iterative design of virtual reality interfaces. *International Journal on Human Computer Sciences, 62*(1), 73–103.

Figueroa, P., Green, M., & Hoover, H. J. (2002). InTml: A description language for VR applications. *Web3d '02: Proceedings of the Seventh International Conference on 3D Web Technology,* 53–58.

Figueroa, P., Hoover, H. J., & Boulanger, P. (2004). *InTml concepts* (Tech. Rep.). Edmonton, Canada: University of Alberta. Computing Science Department.

Figueroa, P., & Mejia, D. (2004). Separation of concerns in immersive applications: An example. *Immersive Projection Technology Workshop*. Ames, IA: Iowa State University.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns. Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

HITLab. (2006). *VR therapy for spider phobia*. Available at http://www.hitl.washington.edu/projects/exposure/.

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,* 129–140.

Kim, G. J. (2005). *Designing virtual reality systems. The structured approach*. Berlin: 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.

MÄK. (2006). *MÄk stealth*. Available at http://www.mak.com/products/stealth.php.

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,* 169–178.

Mejia, D., Figueroa, P., & Hernández, J. T. (2005). Interactive support for VR rapid prototyping based on intml. *International Workshop on Methods & Tools for Designing VR Applications*. Ghent, Belgium: WISE Web & Information Systems Engineering Lab.

Mejia, D., Figueroa, P., Hernández, J. T., & Rosa, F. de la. (2004). Infraestructura de realidad virtual multiplataforma. *XXX Conferencia Latinoamericana de Informatica (clei2004),* 949–956. Arequipa, Peru: Sociedad Peruana de Computación.

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,* 231–240.

Neale, H., Cobb, S., & Wilson, J. (2002). A front ended approach to the user-centered design of VEs. *Proceedings of IEEE Virtual Reality,* 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,* 365–372.

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,* 79–80.

Reiners, D., & Voss, G. (2007). *OpenSG home page*. Available at http://opensg.vrsource.org.

Reitmayr, G., & Schmalstieg, D. (2001). An open software architecture for virtual reality interaction. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology,* 47–54.

Rivera, R., Dawes, B., & Abrahams, D. (2007). *Boost. C++ libraries*. Available at http://www.boost.org.

Sastry, L., Boyd, D., & Wilson, M. (2001). Design review and visualization steering using the inquisitive interaction tool-

kit. *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*. Available at http://www.sense8.com/products/index.html.

SGI. (2003). *IRIS performer home page*. Available at http://www.sgi.com/software/performer.

SGI. (2006). *Networked visualization in the oil and gas industry*. Available at http://www.sgi.com/features/2001/jun/statoil/.

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,* 321–328.

SMART *(SMART Technologies)*. (2007). *SMART board interactive whiteboard*. Available at http://www.smarttech.com/Products/smartboard/index.asp.

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

Spivey, M. (1992). *The Z notation: A reference manual* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Sun Microsystems. (1997). *Java 3D home page*. Available at 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. *Proceedings of the ACM Symposium of Virtual Reality Software and Technology,* 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,* 55–61.

Virtools. (2007). *Virtools*. Available at http://www.virtools.com/index.asp.

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

Web3D Consortium. (2003). *Extensible 3D (X3D™) Graphics*. Available at http://www.web3d.org/x3d.html.

Web3D Consortium. (2005). ISO/IEC FDIS *19777-1:2005. extensible* 3D (X3D) *language bindings part 1:* ECMA-S*cript*. Available at http://www.web3d.org/x3d/specifications/ISO-IEC-19777-1-X3DLanguageBindings-ECMA-Script.

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,* 85–88.