Visual Programming for Virtual Reality Applications Based on InTml

Pablo Figueroa∗, Santiago Gil†, Raúl Oses∗, Juan Toro∗, Catalina Rodriguez∗, Christian Benavides* and Esteban Correa
∗Computing and Systems Engineering
Universidad de los Andes, Colombia
†Heinsohn Business Technology
Colombia

Abstract—This paper presents our work on a visual programming environment (VPE) for portable, implementation-independent, virtual reality (VR) applications. Previously, we have defined InTml, the Interaction Techniques Markup Language, a domain specific language for VR applications, and some initial, command-line based development tools. By using the concept of Model Driven Development (MDD) and with the aid of tools from the Eclipse Graphical Modeling Project (GMF), we built an IDE for VR applications, that allows the visual description of components, application creation, and code generation to targeted runtime environments in C++, Java, and ActionScript. We report some advantages and shortcomings in this approach for tool development, some results from our preliminary user studies and lessons learned. In general, an MDD based approach to VPE is challenging both in terms of learning curve and usability of the final IDE.

I. INTRODUCTION

Virtual Reality (VR) has been selected as one of the Grand Challenges for Engineering by the National Academy of Engineering [1], due to its potential in areas such as training, phobia treatment, or advanced video conferencing, to name a few applications. VR applications provide users with a novel interface that may allow to practice real world situations in the virtual world, learn from simulations, and transfer such learning to their real environments. VR can also be entertaining, so there is great potential to include more VR based experiences in future theme parks or home entertainment setups.

When it comes to VR application development, there are several issues that should be addressed. A good VR application requires powerful and fast 3D graphics, distributed access, good sound, and integrated and possibly redundant devices, which define a complex development scenario. There are libraries and frameworks on top of generic programming languages, which are very powerful in general but may be quite complicated and therefore preclude non-programmer development [2]. Obsolescence is also an issue in this direction, since new libraries are appearing in order to get worth of novel hardware technologies, and old technologies are either abandoned or not actively supported. As consequences, old developments can not be improved and new developments should start in new technologies, which precludes reusability. Several high level tools for VR development have resulted from research, although their users are still limited and sometimes are still work in progress. There are also interesting commercial tools, either for any type of VR application [3], [4] or for specific domains [5], but these environments have not yet facilitated the solution to known challenges (i.e. reusability, portability, or rapid prototyping) or extended the use of VR to other interesting domains.

We believe there is a need for a widely available, community supported, reusability driven environment for VR application development. Such an environment should foster developments in new areas, and at the same time reuse previous results. Developments in such an environment should also be resilient to technological changes, in order to facilitate the execution of previous applications on top of new technologies (i.e. scene graphs, libraries, toolkits, new generations of devices). Such an environment should also facilitate application migration from one VR interface to another, so prototypes can be easily tested and tuned. Finally, it should facilitate programming by non-programmers, so domain experts could explore the use of this technology for their own needs.

This paper presents our IDE for VR development based on the InTml language. InTml is a dataflow based solution that provides a technology agnostic description of VR applications, and possibilities for multiple implementations on top of various technologies and generic programming languages. This work is divided as follows: first, we present related work. Then, we present our IDE, starting the the domain specific language it supports, the tools we have developed, and the development process we encourage. Finally, we present lessons learned and results from preliminary user studies, conclusions, and future work.

II. RELATED WORK

Our work is inspired in novel Software Engineering Approaches and around 20 years of development in the field of User Interface Management Systems (UIMS). In the case of Software Engineering, we based our developments in the concepts of Software Product Lines (SPLs) [6] and Model Driven Development (MDD) [7]. SPLs aim to facilitate development of families of applications, by clearly describing what is common for all applications in such a family and their variability. MDD proposes a development method based
on model transformations and high level description of the solution space. To our knowledge, there are no other IDEs in the VR field that have these foundational concepts.

In terms of UIMS, there are several systems that could be taken into account. We mention here a sample of systems with a brief description of its main features and in chronological order.

Garnet [8], [9] is a large system for the development of WIMP interfaces. Developers can define the look and feel of any interface element, both by a programming language and by a tool that allows programming by example, where developers graphically define the before and after states of an interaction. They count with a visual programming environment for widgets, where the look and feel and the behavior of a widget is defined. They also count with a library of 2D interactors, and debugging facilities. It is based on the LISP programming language, although it seems it was possible to use the high level tools without using LISP code.

The work by Bastide and Palanque [10] shows a model for event driven programming based on the general concept of petri nets, and several examples on how particular widgets can be expressed in such a formalism. Although there are not enough details about tools, this work is important since it shows language alternatives for interface modeling.

Amulet [11] is an evolution of Garnet. It presents a programming model for WIMP interfaces based on a textual programming language, collection of small components which collaborate to give behavior to a particular interface element and the concept of prototypes, which allows creation of novel elements by example. It also presents a behavior library, an interface builder, debugging facilities, and several widget examples.

Lightning [12] is a VR programming framework that features both Tcl and C++ programming. It also sits on top of basic APIs such as Performer, so it can reuse their functionality. Its architecture is inspired in VRML, and it is accessible from both programming languages.

PMIW [13] presents first a dual model for post WIMP interfaces, based on state machines and dataflows for continuous signals, such as the ones from trackers. It then presents several examples, model editors, and implementation issues.

VRJuggler [14] is a framework for VR development. It offers an abstraction level for devices and scene graphs, with a runtime environment that allows configuration changes at runtime.

Contigra [15] is an XML language for the description of 3D interaction components, which proposes a language on top of X3D that extends its functionality.

Ubit [16] presents a system based on a subset of C++ that provides a library of components for building widgets, in a similar manner than prototypes already mentioned.

ICARE [17] presents a set of components to handle multimodal applications, a sketch for an Integrated Development Environment (IDE), and addresses several implementation issues.

The Input Configurator tool (ICON) [18] presents first a model for input devices, and then a model for simple applications based on a 3 layered dataflow of components. MaggLite [19] complements this development with an environment for interface creation by drawing, and an integration of the visual model and the interaction model in ICON.

UsiXML [20] presents a description language for portable widgets and interface elements, with code generation to X3D and VRML.

FlowVR [21] is a VR/AR framework that proposes a dataflow based programming model, with interesting computer vision based demos.

Avango [22] is a VR framework that started to be developed around 1996, but with recent improvements. It has support for both Python and C++, with an architecture similar to Open Inventor, a VRML predecessor.

OpenInterface [23] is a large development for the development of multimodal applications. It features integration of components from several programming languages, tools to support design at different levels of detail, and a database of components, mostly related to devices.

Finally, Squidy [24] presents a dataflow based language for multimodal applications, which features a runtime environment in iPhones and a visualization tool for signals through channels.

A. Addressed Issues

Table I presents a subjective analysis of the issues addressed by each system, according to the cited bibliography, and how our system based on InTml compares to those developments. Issues in the table are organized relative to the number of systems that mentions them, from the highest mentioned to the lowest. An alphabetical list of issues is the following:

- Architecture (AC). Issues related to components, types of components, relationships, and in general software architecture in the implementation of VR applications.
- Particular Applications (AP). The development environment shows a real, complex application as example.
- Code Generation (CG). The system generates code or stubs, as part of its work towards VR application development.
- Programming by Constraints (CT). The programming language at hand allows the definition and manipulation of constraints, a declarative programming style.
- Devices (DE). Device modeling is presented as part of the language or the system.
- Debugging Facilities (DG). The system provides high level debugging, at the level of the proposed programming language.
- Examples (EX). The documentation show several simple examples of certain parts of the system and/or the programming language.

---

1 An example of constraints in WIMP interfaces is the concept of a constraint based layout manager, where elements can be located relative to others instead of using absolute positions.
• Graphics (GR). The system includes a model for graphical objects, or tools that facilitate the creation of graphical content.
• Implementation Issues (IM). Particular implementation issues over a generic programming language are presented.
• Interaction Components (IN). There are components dedicated to the description of the interaction in the application, apart from devices and graphic components.
• Multimodality (MM). The system/language is explicitly addressing multimodality.
• Programming Language (PL). The system shows a new programming language or a way to use a generic one (i.e. the declarative C++ in Ubit).
• Runtime issues (RT). There is an in-depth description of the runtime environment in which applications are run.
• Tools (TO). Tools for certain development purposes are presented.
• Widgets (WD). The system provides support for widget development, or interface elements that allow users to input information to the VR application.

In summary, we can notice that InTml concentrates in a very consistent way on the most common issues from this list. Although it is just and after thought on several years of development, it also shows some rationale about our course of action.

III. INTML IDE

This section presents our developments, divided in language concepts, development processes and roles, and available tools. Our main purpose is to support uniform development across several runtime environments, so designers can create their applications at the level of InTml and developers can collaborate with details about the overall architecture and particular component’s implementation.

A. The InTml Language

InTml is a dataflow based language for high level description of VR applications. Each component counts with several input and output connection ports, which may receive or send zero or more events each frame. More details can be found elsewhere [25], [26].

B. Tools

Based on Eclipse and the GMF, we have implemented the following tools: a library editor, an application editor, a code generator, and several runtime environments, in C++, Java, and ActionScript. More details follow.

1) Library Editor: The library editor allows the creation of new component classes, in terms of the description of their ports and a unique name. Ports should select a type from a list of basic types, shown in Table II, which could be translated to equivalent types in particular implementations. Figure 1 presents a common view of such an editor, which is a plug-in in Eclipse. Apart from the standard areas in an Eclipse perspective, such as the package explorer, the properties tab, and the task list; this interface provides a canvas for the set of classes in the library, a palette with available operations (create input port, output port, or filter class), and an outline of the entire library.

2) Code Generator: Component classes in a library should be translated into specific code in each available runtime environment. Developers can do so by selecting the option "Generate FilterClass Implementation" in the popup menu of each component, or "Generate Library Implementation" of the entire canvas. These functions execute code written in one of the GMF’s languages. Advanced developers take these stubs and write any missing parts, targeted to a specific runtime environment.

3) Application Editor: An application is a set of interconnected component instances. From the application editor it is possible to import libraries of component classes, create new filters, or instances of component classes, create constants and

| Category               | Type Names                                      |
|------------------------|-------------------------------------------------|
| Occurrence of an event | Signal                                          |
| Basic types            | Integer, Boolean, Float, Double, Long, Byte     |
| Spatial types          | Pos2D, Pos3D, Orientation, Orientation3D, Quaternion, Matrix4 |
| Id plus type           | IdFilter, IdBoolean, IdDouble, IdLong, IdByte, IdPos2D, IdPos3D, IdOrientation, IdOrientation2D, IdOrientation3D, IdQuaternion, IdString, IdFloat |
| A Component's instance | Filter                                          |
| Others                 | String                                          |

Fig. 1. The Library Editor
Table I

| System     | EX | PL | AC | TO | DE | IM | IN | CG | RT | WD | MM | GR | CT | DG | AP |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Garnet     | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| Petri Nets based | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| Amulet     | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| Lightning  | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| PMW        | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| VRJuggler  | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| Contigra   | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |
| Ubit       | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| ICARE      | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| ICON       | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| MaggLite   | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| UsiXML     | X  | X  |    | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |
| FlowVR     | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| Avango     | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| InTml      | X  | X  | X  | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |
| OpenInterface | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |    |    |
| Squidy     | X  | X  | X  | X  | X  |    |    |    |    |    |    |    |    |    |    |

Table III

| Name         | Technologies                                |
|--------------|---------------------------------------------|
| C++ (1)      | VRJuggler[14], OpenSG[27], VRPN[28]        |
| C++ (2)      | Scenix[29], VRPN                           |
| Java         | Java3D[30], VRPN                           |
| ActionScript | PaperVision, VRPN                          |

We created first the Java runtime, as a proof of concept for the development of VR application families and its advantages in reusability and application evaluation [31]. Although this runtime didn’t count at the beginning with the VPE described here, it was re-engineered to fit the code generation schemes available. Lately, it has been used to develop a family of applications for training on machine tools kinematics [32], both in a cluster based, 9m wide by 3m high large screen and in a web based environment.

The C++ versions follow, as a way to get advantage of all the performance available in large visualization facilities. Several small demos were developed in the first runtime, and we decided to implement a new runtime on top of Scenix in order to improve the visual quality of our applications.

Finally, as a way to facilitate development in popular environments such as Flash, we implemented a version in ActionScript [33]. Our purpose was to facilitate designers a VPE outside our lab environment, which may be used to define entire InTml applications in a simulated working space, which could be ported later on to the VR setups with the real devices. This environment introduced also the concept of device recording, so designers could save events of a real devices and then play them in their Flash based environment in order to test their applications during development.

Part of our implementation philosophy is to adapt components to the particular capabilities and limitations of each runtime, a process we call VR Retargeting [25]. For example, connect filters and constants 3. Connections are type checked, and it is possible to create several connections in or out a port. Figure 2 shows a view of this editor.

4) Runtime Environments: We have created 4 runtimes for InTml programs, based on several technologies and programming languages, as it is shown in Table III. Each environment accommodates to particular characteristics, given by its underlying technologies, and at the same time allows us to consider application ports if necessary, since all its description is in InTml.

OpenSG, Scenix, Java3D, and PaperVision are scene graphs, APIs for 3D graphics management. VRPN is a device abstraction layer, widely used in the VR community, with bindings for several programming languages. VRJuggler is a C++ based framework for VR development, which provides us with some output device abstraction.

3The interface also allows editin filter holders, which is an InTml concept for a filter reference.
collision detection in a cluster is more precise than the same functionality in Flash. However, we want to present designers with a uniform experience, based on the InTml language and its standard library, with no details regarding the runtime capabilities.

C. Development Process and Roles

Our development process is depicted in Figure 3. 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 [31], a matching test that shows three objects and three copies of such objects, to be executed in four VR hardware environments available at our lab [34] and similar installations.

Our ultimate objective is to facilitate access to the VR environments available at our lab [34] and similar installations, so designers could concentrate on application functionality and forget all issues related software integration and learning curves in particular technologies. We believe we are advancing towards this goal, although there are several issues we have to address, as the following studies and lessons suggest.

In our first experiment with final users, 15 students from several specialties in an extension program in Multimedia received a three hour training in InTml. After training, they were asked to design MR applications in groups of about 4 people, in order to promote discussion and self tutoring on the basic concepts of this new language. After a week, designs were shown to all class, and a discussion about each design and the overall experience was performed. Although most of them developed their designs just in paper due to installation and usage issues with the VPE, we could observe that concepts were understood and that our DSL allowed them to express their designs. We also observed that they prefer to learn new languages and concepts with approaches such as trial–and–error and learning–by–examples, which could be supported through an IDE. Some issues they made explicit.

Fig. 3. InTml–Based Development Process

- **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 have not implemented yet Task Views as part of our IDE.

- **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 the IDE 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.

- **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.

IV. STUDIES AND LESSONS LEARNED

This VPE has been developed and maintained during the last 4 years by Master students of our Computing and Systems Engineering Department, and used by them and by graduate students in an extension program of the Faculty of Arts. Our ultimate objective is to facilitate access to the VR environments available at our lab [34] and similar installations, so designers could concentrate on application functionality and forget all issues related software integration and learning curves in particular technologies. We believe we are advancing towards this goal, although there are several issues we have to address, as the following studies and lessons suggest.

In our first experiment with final users, 15 students from several specialties in an extension program in Multimedia received a three hour training in InTml. After training, they were asked to design MR applications in groups of about 4 people, in order to promote discussion and self tutoring on the basic concepts of this new language. After a week, designs were shown to all class, and a discussion about each design and the overall experience was performed. Although most of them developed their designs just in paper due to installation and usage issues with the VPE, we could observe that concepts were understood and that our DSL allowed them to express their designs. We also observed that they prefer to learn new languages and concepts with approaches such as trial–and–error and learning–by–examples, which could be supported through an IDE. Some issues they made explicit.
were related to the restrictions that they could feel from IDEs for other languages, and how such restrictions helped them to understand the way to use such a language.

Our second experiment was performed a year later with 26 students of the same course. This time, we had 6 hours of training in both InTml and the VPE, and we asked groups of about 4 people to design InTml an application they have already defined in storyboards. They had a week for this assignment, 3 hours more of coaching regarding issues related to their design in InTml or the use of our tool, and finally they presented their results. In this case we could notice that limited examples did not allow students to fully understand the semantics of both the language and the example components they had at hand, although such problems were solved in the coaching session. Their final designs were interested, although their required too many new components and therefore we did not have the time to implement all of them (i.e. generate stub code for them, include new code, and include such new component to the final runtime environment for its execution).

In summary, a more rich set of implemented components is required in order to facilitate development.

Apart from the user studies, we would like to report some insights on the development of this VPE for InTml. In general, component development is very difficult, due to the sheer amount of technologies that should be mastered. On top of the technologies involved in each runtime platform, developers should learn inner details related to the InTml implementation in such language, and the way eclipse, GMF, and their code generation tools work. This scenario is overwhelming, and requires either lengthy learning curves or very specialized professionals. Since students are usually more interested in applications on top of this development environment than in the environment itself, there is little interest in taking the time for learning all these technologies.

Another issue is related to the generated VPE in eclipse, shown in figures 1 and 2. Although GMF implements several aids for MDD, the generated VPE still requires quite a bit of tuning to achieve a usable tool. The interface includes several menus and options that eclipse has by default, although they may not be needed in a particular domain as ours. The generated code still has to be completed in order to create a more uniform experience in eclipse, such as the way to do copy and paste or undo in a particular VPE. On top of that, eclipse users have high expectations about the type of functionality expected in their IDE, and it is still a lot for work to achieve such functionality from the basic generated code. In some way, the resulting VPE fulfills the requirements from the MDD standpoint, but it still misses to provide a usable VPE. More work has to be done within the GMF code in order to generate more user-friendly VPEs for a particular domain.

Finally, there is the issue of consistency between runtimes, both at the level of language and library support. Although there is a formal description of InTml in the Z language that aims to be a blueprint for any new runtime [35], it requires trained developers to properly understand it and use it. Even if runtimes are not that different, it is hard for developers to implement certain functionality in a consistent way in different platforms. For example, not all scene graphs provide services for identifying which objects collide with a ray, a common selection task, and those who offer such a functionality provide different results: the closest intersected object, a set of intersected objects, the closest intersected object that is selectable, to mention some options. It is an extra burden for a developer to implement the expected InTml behavior with whatever is available in the underlying implementation.

In summary, we believe we are going in the right direction to achieve our ultimate goal of portable, platform independent VR development based on MDD and SPL, although there are several issues in the underlying technologies we would like to improve.

V. CONCLUSION AND FUTURE WORK

We present a visual programming environment for the development of portable VR applications, based on the InTml language. Our development is based on the software engineering concepts of Model Driven Development and Software Product Lines. With the aid of eclipse and its Graphical Modeling Framework, we have developed a visual programming environment for InTml and integrated a set of runtime environments in different programming languages. We believe it is an interesting approach for VPE development, although more work has to be done in order to achieve a more usable interface.

ACKNOWLEDGMENT

The authors would like to thank all students involved in this project throughout the years, in particular to Daniel Mejia. Part of the funds for this project were provided by IBM Corporation, with its program of Faculty Innovation Awards, 2007.

REFERENCES

[1] National Academy of Engineering, “Grand challenges for engineering,” http://www.engineeringchallenges.org/, 2010.
[2] M. K. Annett and W. F. Bischof, “Investigating the application of virtual reality systems to psychology and cognitive neuroscience research,” Presence: Teleoper. Virtual Environ., vol. 19, pp. 131–141. [Online]. Available: http://dx.doi.org/10.1162/penj.19.2.131
[3] Virtools, “Virtools,” http://www.virtools.com/index.asp, 2007.
[4] WorldViz, “Vizard,” http://www.worldviz.com/products/vizard/, 2010.
[5] Forum 8, “Uc-win/road,” http://www.forum8.com/uc_win_road.htm, 2011.
[6] P. Clements and L. Northrop, Software Product Lines. Addison Wesley, 2002.
[7] T. Stahl and M. Voelter, Model-Driven Software Development: Technology, Engineering, Management. Wiley, 2006.
[8] B. A. Myers, “A new model for handling input,” ACM Trans. Inf. Syst., vol. 8, pp. 289–320, July 1990. [Online]. Available: http://doi.acm.org/10.1145/98188.98204
[9] B. Vander Zanden and B. A. Myers, “Demonstrational and constraint-based techniques for pictorially specifying application objects and behaviors,” ACM Trans. Comput.-Hum. Interact., vol. 2, pp. 308–356, December 1995. [Online]. Available: http://doi.acm.org/10.1145/212340.212443
[10] R. Bastide and P. Palanque, “A petri net based environment for the design of event-driven interfaces,” in Application and Theory of Petri Nets 1995, ser. Lecture Notes in Computer Science, G. De Michielis and M. Diaz, Eds. Springer Berlin / Heidelberg, 1995, vol. 935, pp. 66–83, 10.1007/3-540-60029-9_34. [Online]. Available: http://dx.doi.org/10.1007/3-540-60029-9_34
