3dml: A Language for 3D Interaction Techniques Specification

Pablo Figueroa, Mark Green and H. James Hoover

{pfiguero,mark,hoover}@cs.ualberta.ca
Department of Computer Science, University of Alberta, Edmonton,
Alberta, Canada

Abstract

We present 3dml, a markup language for 3D interaction techniques and virtual environment applications that involve non-traditional devices. 3dml has two main purposes: readability and rapid development. Designers can read 3dml-based representations of 3D ITs, compare them, and understand them. 3dml can also be used as a front end for any VR toolkit, so designers without programming skills can create VR applications as 3dml documents that plug together 3D ITs, VR objects, and devices. This paper focuses on the language features and presentation scheme designed in our website (http://www.cs.ualberta.ca/~pfiguero/3dml).

1. Introduction

3dml is a markup language that describes applications such as desktop-based 3D presentations, Virtual Reality (VR) applications, or Augmented Reality (AR) applications†; in other words, applications with different types of input devices, output devices, and 3D interaction techniques (ITs). 3dml defines a way to describe classes of 3D ITs and devices, and it allows developers to combine instances of such classes and VR objects in applications.

3dml is a uniform language to represent 3D ITs, a language that can be readable by programs, as well as by designers. It provides an instrument for comparison of 3D ITs, for the study of alternative interaction techniques in an application, and for rapid prototyping. 3dml can accelerate the production of VR applications, because it is a higher level language than the languages traditionally used in this area. 3dml also provides a way to hide unnecessary details from designers such as device configuration, interaction technique implementation, scene graph details, and so on.

This paper is organized as follows: Related representation techniques are presented in Section 2, followed by the basic concepts of this representation in Section 3, and the main features with examples in Section 4. The presentation aids designed in our website – http://www.cs.ualberta.ca/~pfiguero/3dml – are presented in Section 5, the future work in Section 6, and finally some conclusions in Section 7.

† From now on VR applications

2. Related Work

Here we analyze the main two areas that 3dml is trying to address: readability and rapid development. The resources that a developer has to understand an IT are basically two: the available documentation in the toolkit that implements such technique, and papers that present its rationale and purpose. However, it is usually difficult to understand the implementation of an IT in a toolkit without a deep knowledge about the toolkit's architecture, and it is difficult to compare an implemented IT with non-implemented alternatives, because of the large diversity of presentation techniques in research papers. 3dml aims a solution to this.

In terms of application development, current VR toolkits define ITs in programming languages such as C++ or Java, so programming skills in such languages are necessary to develop any application. VRML[6] and its successor X3D[18] are descriptive languages that require less programming skills than C++ or Java, and can define 3D ITs as a set of SCRIPT nodes connected by ROUTEs; however it is not easy to identify particular ITs, to change the default set of devices, and to reuse such ITs in other programs.

X3D Components[19] define a way to extend the functionality of X3D by using BML[4], a markup language for Java beans integration that is very powerful but exposes much of the bean complexity to the user. 3dml proposes a simpler component model to the one used in BML‡.

Newsgroups such as 3dui[1] have been discussing a way to describe 3D ITs, either to capture their design and intention in formalisms such as UAN[15], or to capture their main algorithm and code generation mechanisms by pseudocode or scripting languages such as Python[17]. This paper presents a markup language that can be shown to the user in a more friendly way, and that can be also parsed in order to make an executable application.

3. Basic Concepts

3dml uses a dataflow architecture as the general structure of an application. VR objects, devices, and ITs are the components of this dataflow, and there is a simple component model that defines how such elements are plugged together.

The component model used in 3dml differs from standard models such as Java Beans[10], CORBA[8], or COM[9] in the following ways:

- 3dml can be considered an interface definition language (IDL) based only on the concepts of typed input and output (I/O) ports.
- It is, in principle, a simpler model than the others. It defines only the concepts required to specify a VR application at a high level of abstraction.
- Only a specific and known subset of components are allowed to make changes to the scene. This feature makes explicit the modifications to the world, so the overall behavior of an application is easy to understand.
- Connections between components allow concurrent and enriched fan-in – several components send a message to a component – and fan-out – a component sends a message to several components. For example, if several components try to change the position of an object at the same time (fan-in), different policies can apply, such as to add all concurrent values or to pick one at random.
- It is possible to define default values for connections in a component. At runtime, the state's component is initialized with such values.

The following paragraphs define the main concepts in 3dml. A *filter* is the smallest process unit in the dataflow, composed of input and output ports – which carry typed events –, and state information. Figure 1 shows a way to represent a filter – SelectByTouching – in a diagram that shows its input ports on the left of a box and output ports on the right. Its output is a selected object from the scene. Its inputs are the VR object 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 added or deleted objects from the scene. Some of these values are redundant - i.e. the hand representation has also the information about position and orientation - but this redundancy allows flexibility in the modeling of the filter, since there are several ways to connect its input, as follows:

- If the hand representation can change, the hand representation input port will inform the filter of such a change.
- If objects can be added to or removed from the scene, the filter can know it by connecting the addObject or removeObject ports.
- If the hand representation is fixed throughout the application, the filter can receive just the expected changes - position or orientation - and avoid the cost of receiving all the object at every change.
- If the scene is fixed, the filter does not need to connect the addObject and removeObject input ports.

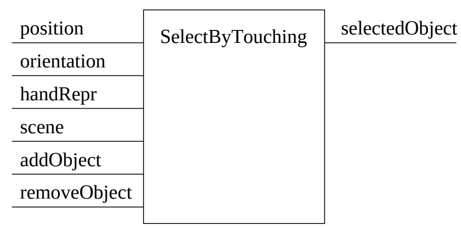

Figure 1: *Select by Touching. An Example of a Filter*

It is up to the IT designer to define the correct level of generality for each filter, in order to maximize flexibility and avoid unnecessary performance penalties. SelectByTouching is stateless, but a filter can also have an internal state. For example, *FeedbackOne* in Figure 3 remembers the last selected object and its previous color, and changes it back when required §.

VR objects represent identifiable pieces of content in the virtual environment: elements that can be seen, heard, or touched by the user. An *object holder* is a special type of filter that allows changes on a VR object. An *input device* is a filter with just output ports that sends events of a certain type to the dataflow. An *output device* is a placeholder that describes where the output of the application will be shown – it is internally related to the VR objects, but the details are hidden to the VR designer.

An *interaction technique* is a set of interconnected instances of filters and objects , that is identifiable and reusable. An IT has the same interface as a filter in terms of input and output ports, so it can be used in an application

‡ Our component model is analyzed in Section 3.

§ Feedback is actually an interaction technique, but its implementation has filters with internal state.

without knowing details about its implementation. This reduces the complexity that VR designers have to deal with. Figure 2 shows the Go-Go interaction technique[14] – an IT to lengthen the user's virtual arm for reaching distant objects– as it is used by VR designers, and Figure 3 shows all the filters and objects involved, for somebody interested in the details of how Go-Go works.

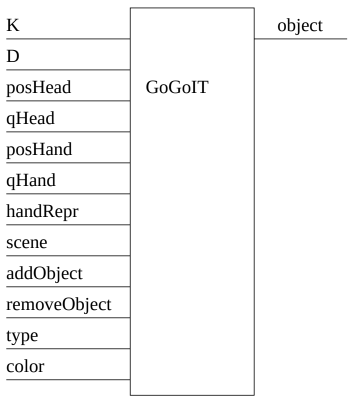

Figure 2: *Go-Go IT. General view.*

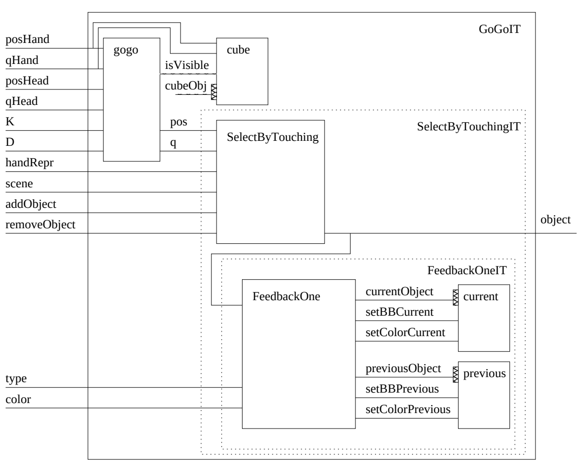

Figure 3: *Go-Go IT. Details.*

An *application* allows a designer to plug together all the previous elements in order to meet certain user requirements. Figure 4 shows a simple application, that allows an user to move a virtual hand with a tracker and touch virtual objects. In this example we have one filter (SelectByTouching), one IT (Feedback), one input device (handTracker), one output device (console), two VR objects (scene and handRepr), and one object holder (handRepresentation).

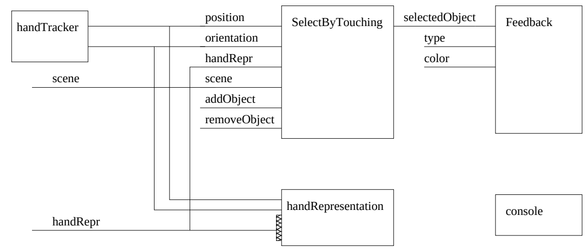

Figure 4: *Simple Application. Touching Objects With a Virtual Hand.*

Filters, ITs, and applications are documented in 3dml by a short description, a long description, indexes (explained in Section 5), and references to papers where they are defined. This information is used in the presentation scheme presented in Section 5.

4. Main Features and Examples

3dml is a XML application that defines all the basic concepts described before. Listings 1, 2, and 3 are 3dml documents that represent the examples in Figures 1, 2, and 4, respectively ¶. These examples are used to describe the most important features of 3dml in the following paragraphs.

List 1 Select by Touching. XML Code

```
<FilterClass id="SelectByTouching">
  <IPort id="p" type="Pos3D">
   <ShortDesc>Change of position
   </ShortDesc> </IPort>
  <IPort id="q" type="Quaternion">
   <ShortDesc>Change of rotation
   </ShortDesc></IPort>
  <IPort id="handRepr" type="VRObject">
   <ShortDesc>
     Object that represents
     the users' hand
   </ShortDesc></IPort>
  <IPort id="scene" type="Scene">
   <ShortDesc>Selectable objects
   </ShortDesc></IPort>
  <IPort id="addObject" type="VRObject">
   <ShortDesc>Dynamically added objects
   </ShortDesc></IPort>
  <IPort id="removeObject"
         type="VRObject">
   <ShortDesc>Dynamically removed objects
   </ShortDesc></IPort>
  <OPort id="object" type="VRObject">
   <ShortDesc>Selected object
   </ShortDesc></OPort>
</FilterClass>
```

¶ For space reasons, only important elements are presented here. For more details look at the 3dml website.

List 2 Go-Go IT. XML Code

```
<ITClass id="GoGoIT">
  <Filter id="gogo" type="GoGo"></Filter>
  <IT id="select"
      type="SelectByTouchingIT"></IT>
  <VRObject id="cubeObj" primitive="Box"/>
  <ObjectHolder id="cube">
    <Input id="setVisible" target="gogo"
           port="setVisible"/>
    <Input id="object" target="_self"
           port="cubeObj"/>
  </ObjectHolder>

  <Binding target="gogo" port="pos">
    <Port target="select" port="pos"/>
  </Binding>
  <Binding target="gogo" port="q">
    <Port target="select" port="q"/>
  </Binding>

  <IPort id="K" type="float"
         defValue="0.167">
    <Port target="gogo" port="K"/>
  </IPort>
  <IPort id="D" type="float"
         defValue="0.6">
    <Port target="gogo" port="D"/>
  </IPort>
  <IPort id="posHead" type="Pos3D">
    <Port target="gogo" port="pHead"/>
  </IPort>
  <IPort id="qHead" type="Quaternion">
    <Port target="gogo" port="qHead"/>
  </IPort>
  <IPort id="posHand" type="Pos3D">
    <Port target="gogo" port="pHand"/>
    <Port target="cube" port="p"/>
  </IPort>
  <IPort id="qHand" type="Quaternion">
    <Port target="select" port="qHand"/>
    <Port target="cube" port="q"/>
  </IPort>
  <IPort id="handRepr" type="VRObject">
    <Port target="select"
          port="handRepr"/>
  </IPort>
  <IPort id="addObjectToScene"
         type="VRObject">
    <Port target="select"
          port="addObjectToScene"/>
  </IPort>
  <IPort id="type" type="String">
    <Port target="select" port="type"/>
  </IPort>
  <IPort id="color" type="Color">
    <Port target="select"
          port="colorSelection"/>
  </IPort>
  <OPort id="object" type="VRObject">
    <Port target="select" port="object"/>
  </OPort>
</ITClass>
```

List 3 Simple Application. XML Code

```
<App>
  <Scene id="scene" filename="scene.wrl"/>
  <Object id="handRepr"
          filename="hand.wrl"/>

  <IDevice id="handTracker"/>
  <ODevice id="console"/>

  <Input id="pos" target="handTracker"
         port="pos"/>
  <Input id="q" target="handTracker"
         port="q"/>

  <ObjectHolder id="handRepresentation">
    <Input id="setTranslation"
           target="_self" port="pos"/>
    <Input id="setRotation"
           target="_self" port="q"/>
  </ObjectHolder>

  <Filter id="select"
          type="SelectByTouching">
    <Input id="handRepr"
           target="_self" port="handRepr"/>
    <Input id="pos"
           target="_self" port="pos"/>
    <Input id="q"
           target="_self" port="q"/>
    <Input id="scene"
           target="_self" port="scene"/>
  </Filter>
  <IT id="feedback" type="FeedbackOneIT">
    <Input id="object"
           target="select" port="object"/>
  </IT>
</App>
```

Class Declarations

Filter and IT classes are defined in 3dml with the elements `FilterClass` and `ITClass`, respectively. For example Listing 1 defines the filter class `SelectByTouching` and Listing 2 defines the IT class `GoGoIT`. Filters and ITs classes define their interface, in terms of input and output ports. Each input and output port has a name and a type, and an input port can have a default value. For example, the line `<IPort id="D" type="float" defValue="0.6">` in `GoGoIT` defines the input port called `D`, of type `float` and with default value `0.6`, and `<OPort id="object" type="VRObject">` defines an output port named `object` of type `VRObject`.

Application Definition

The element `App` is used to create applications. Listing 3 shows how instances of objects, devices, filters, and ITs can be defined and connected together in order to show selection by touching.

Instantiation

IT classes can contain instances of objects, filters, and other ITs. For example, in lines 2–5 of Listing 2, we define the object `cubeObj` as a simple box, the filter `gogo` of class `GoGo` that computes the lengthening behavior of this technique, and the IT `select` of class `SelectByTouchingIT`, that finds an object that collides with the virtual hand and changes its color.

Applications can contain all instances that an IT can, plus instances of devices. In Listing 3, the application instantiates the `handtracker` and the `console` as its input and output devices, respectively.

Variable and constant declaration

Variables and constants can be created inside an application with the elements `Input` and `Constant`. Variables are initialized with a port name, constants with a value, and they can be used wherever an output port is used. The application in Listing 3 creates two variables, `pos` and `q`, and they are used to separate the platform-dependent part of the application – I/O devices – from the platform-independent – filters and ITs.

Filter and IT connection

Filters and ITs can be connected in order to create new applications and ITs. Basically, connections are valid if the origin is an output port, and the destination is an input port of the same type as the origin. The examples show the following types of connections:

- Explicit connection. The element `Binding` allows us to connect the output port of a filter/IT to the input port of another one. Two examples are shown in Listing 2, lines 13–18
- IT input. Input ports of an IT are redirected to input ports of the contained object holders, filters and ITs by using the element `Port`. This mechanism defines how the IT will use its internal implementation, so users of such IT do not have to worry about this detail. For example, the input ports of GoGoIT in Listing 2 are redirected to input ports in `cube`, `gogo`, and `select`.
- IT output. The output of any IT is defined as a subset of the output of its filters and ITs. For example, the output of GoGoIT in Listing 2 is just the selected object, that is taken from the IT `select`, part of its implementation.
- Object holder ports. An object holder is connected to filters and ITs by using the element `Binding`, or by defining a `target` and a `port` in each defined `Input`. For example, Listing 2 defines the object holder `cube`, and connects its input ports: `object` – the object that will be modified – and `setVisible` – in order to show or hide the object.

Geometry of VR objects

3dml does not has elements to define object geometry. Instead, a `VRObject` can import its geometry from a file, or be defined as a primitive: `Box`, `Cone`, `Cylinder`, and `Ellipse`. Listing 2 defines a Box in line 5, and Listing 3 defines `handRepr` as the geometry in `hand.wrl`, and a set of objects called `scene` as the geometry defined in `scene.wrl`.

Embeded documentation

All important elements in 3dml have textual descriptions. A short description is sometimes mandatory as for the ports of `SelectByTouching` in Listing 1.

Element overloading

Several elements in 3dml are overloaded, in order to represent similar concepts in the same way. For example, I/O ports in filters and ITs are represented with the elements `IPort` and `OPort`, despite the fact they only require `Port`s in an IT. `Input` is also overloaded, because it can represent variables in an application, or input bindings in filters. We think this mechanism enhances the similarities of such elements and avoids possible doubts of its use.

IT Reuse

All IT and filter classes can be reused in new ITs and applications, just by creating instances of such classes. For example, Listing 3 creates a filter of class `SelectByTouching` and an IT of class `FeedbackOneIT`, and connects them in an application.

Complexity hiding

The information required to use an IT is only its description and its interface. Details about the IT's implementation can be hidden, so the overall application can be easier to understand.

5. Presentation Scheme

A presentation scheme for 3dml documents have been developed in our web site, and it is shown in Figure 5. These webpages present 3dml documents in a more readable way, for comparison and understanding purposes. The main elements in this scheme are:

- File view. Applications and ITs are classified by the file they are contained in. It is the basic view for the library contents and the simpler way to browse the information.

- Category views. It is possible to create several hierarchical indexes for ITs and applications. In this way ITs can be classified by several criteria, which can help designers to understand easier the library and choose the best ITs for a particular application. For example, all ITs we have described in 3dml are classified by the paper they appear in and by this basic classification: travel, selection, manipulation, control, and feedback ‖.
- Filter details. It presents a general description of the filter, its interface – input ond output ports –, its position in other categories, details about its ports, and bibliography.
- IT details. It presents a general description of the technique, its interface, its implementation – object holders, filter, and IT instances –, details about ports, and bibliography.
- Application details. It presents a summary of the tasks – ITs and filters –, objects, and devices that the application uses, with a detailed information of connections and purposes of each element.

These HTML pages have been generated from the 3dml documents using XSLT[20], and can be used to show any document that follows the 3dml definition rules. The specification contains in this moment the most important 3D ITs presented in papers in the last 10 years: selection by raycasting, selection by touching, aperture-based selection[7], techniques based on proprioception[12], image-plane based techniques[13], walking techniques[16].

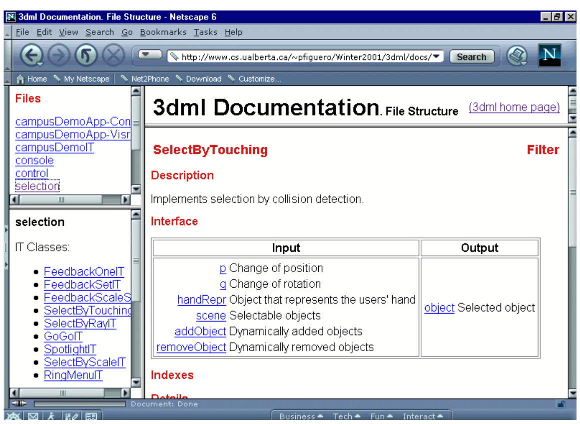

Figure 5: *Documentation view of 3dml.*

6. Future Work

This is the first stage of our project, in which we define the language and the documentation mechanisms of 3D ITs and VR applications. It is possible to design tools to view 3D ITs in more readable formats –i.e. diagrams automatically generated from the description. 3dml is also designed to be a front end for development in any toolkit, and two implementations are in development, one in Java3D[11] and one in VR-Juggler[3].

We have started a process of debugging the 3dml documents that describe our library of 3D ITs. In the future we plan to incorporate more ITs, offer the designer better tools to handle this representation, and incorporate more documentation and usability data.

‖ Inspired from [5] and [2]

7. Conclusions

3dml defines a uniform way to represent 3D ITs that is high-level, toolkit-independent, component-based, reusable, and extensible. Designers of VR applications can understand and compare several 3D ITs described in the same language, and with an uniform documentation paradigm. This language also allows designers to represent VR applications at a high level of abstraction, allowing them to easily use 3D ITs.

References

1. 3d user interfaces newsgroup. http://www.mic.atr.co.jp/~poup/3dui.html.
2. Jon Barrileaux. *3D User Interfaces With Java 3D*. Manning Publications, August 2000.
3. Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. Vr juggler: A virtual platform for virtual reality application development. In *Proceedings of IEEE Virtual Reality*, pages 89–96, 2001.
4. BML: Bean Markup Language. http://www.alphaworks.ibm.com/tech/bml.
5. Doug A. Bowman and Larry F. Hodges. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *The Journal of Visual Languages and Computing*, 10(1):37–53, Februrary 1999.
6. Rikk Carey and Gavin Bell. *The Annotated Vrml 2.0 Reference Manual*. Addison-Wesley, 1997.
7. Andrew Forsberg, Kenneth Herndon, and Robert Zeleznik. Aperture based selection for immersive virtual environments. In ACM, editor, *UIST*, pages 95–96, 1996.
8. Object Management Group. CORBA: Component Object Request Broker Architecture. http://www.omg.org/corba/.
9. Microsoft. COM: Component Object Model. http://www.microsoft.com/com/.
10. Sun Microsystems. Java beans home page. http://java.sun.com/products/javabeans/.

11. Sun Microsystems. Java 3d home page. http://java.sun.com/products/ java-media/3D/index.html, 1997.

12. Mark R. Mine, Frederick P. Brooks Jr., and Carlo H. Sequin. Moving objects in space : Exploiting proprioception in virtual-environment interaction. In ACM, editor, *SIGGRAPH*, pages 19–26, 1997.

13. Jeffrey S. Pierce, Andrew Forsberg, Matthew J. Conway, Seung Hong, Robert Zelenik, and Mark R. Mine. Image plane interaction techniques in 3d immersive environments. In ACM, editor, *Symposium on Interactive 3D Graphics*, pages 39–43, 1997.

14. Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique: Non-linear mapping for direct manipulation in vr. In ACM, editor, *Symposium on User Interface Software and Technology*, pages 79–80, 1996.

15. A. C. Siochi and H. R. Hartson. Task-oriented representation of asynchronous user interfaces. In ACM, editor, *Proceedingsof the SIGCHI conference on Wings for the mind*, pages 183–188, 1989.

16. Martin Usoh, Kevin Arthur, Mary C. Whitton, Rui Bastos, Anthony Steed, Mel Slater, and Jr. Frederick P. Brooks. Walking > walking-in-place > flying, in virtual environments. In *Proceedings of the SIGGRAPH 1999 annual conference on Computer graphics*, pages 359–364. ACM, 1999.

17. Guido van Rossum. Python language website. http://www.python.org/.

18. Web 3d consortium. http://www.web3d.org.

19. X3d components specification. http://www.web3d.org/ TaskGroups/x3d/lucidActual/ X3DComponents/X3DComponents.html.

20. XSLT: XSL Transformations. http://www.w3.org/TR/xslt11/.