A framework to develop VR interaction techniques based on OpenInterface and AFreeCA

Diego Martínez[1] , J-Y. Lionel Lawson[2], José P. Molina[1], Arturo S. García[3], Pascual González[1], Jean Vanderdonckt[2], Benoit Macq[2],

[1] Laboratory of user Interfaces and Software Engineering, University of Castilla- La Mancha, Spain. Email: {diegomp1982, jpmolina, pgonzalez}@dsi.uclm.es
[2] Laboratoire de Telecommunications et Teledetections, Université Catholique du Louvain, Belguique. Email: {jean-yves.lawson, jean.vanderdonckt, benoit.macq}@uclouvain.be
[3]SymbiaIT, Parque Científico y Tecnológico, Albacete, Spain.
Email: arturo@symbiaIT.com

Abstract. Implementing appropriate interaction for Virtual Reality (VR) applications is one of the most challenging tasks that a developer has to face. This challenge is due to both technical and theoretical factors. First, from a technical point of view, the developer does not only have to deal with non-standard devices, he has to facilitate their use in a parallel a coordinated way, interweaving the fields of 3D and multimodal interaction. Secondly, from a theoretical point of view, he has to design the interaction almost from scratch, as a standard set of interaction techniques and interactive tasks has not been identified. All these factors are reflected in the absence of appropriate tools to implement VR interaction techniques. In this paper, some existing tools that aim at the development of VR interaction techniques are studied, analysing their strengths and, more specifically, their shortcomings, such as the difficulties to integrate them with any VR platform or their absence of a strong conceptual background. Following that, a framework to implement VR interaction techniques is described that provides the required support for multimodal interaction and, also, uses experience gained from the study of the former tools to avoid previous mistakes. Finally, the usage of the resulting framework is illustrated with the development of the interaction techniques of a sample application.

Keywords: Novel User Interfaces and Interaction Techniques,Tools for Design, Modelling, Evaluation, Multimodal interfaces, Virtual Reality

1 Introduction

The WIMP (Window-Icon-Mouse-Pointer) metaphor has been broadly accepted and this has provided a standardized set of Interaction Techniques (ITes) and Basic

Interactive Tasks (BITas) for these applications. As a result, stable and mature GUIs have been produced, rules to guide interaction have been formalized [1] and integrated development environments (IDEs) are available to design and implement them. On the other hand, the current status of Virtual Reality (VR) interfaces is far less mature. Conceptually, it has been long accepted that a set of universal BITas exists [2] and many possible classifications have been suggested [3], [4], [5], [6]. However, a standardized set of BITas and ITes has not been identified [5] or, rather, agreed.

The importance of defining this set of universal tasks and techniques has been deeply discussed in the 3DUI mailing list [7] (see Figueroa in August, 2004), as they would be the starting point for the development of toolkits to implement 3DUIs. As a result of this situation, the tools available do not provide the required support to deal with the complexity of 3D interaction. In some cases, developers will have to implement their own solutions mostly from scratch, relying on their previous experience and personal criteria. Taking into account the complexity and the multidisciplinary nature of 3D interaction –3DUIs, multimodal interaction, ergonomics, human factors, etc.-, designing interaction can result one of the most challenging tasks that the developer has to face. In some other cases, the support provided forces developers to be bound to a specific VR platform and programming paradigm. This factor restricts developer's freedom, limiting him/her to the BITAs and ITes considered when creating the platform, which may not fit the requirements of the system implemented.

This paper describes a framework to assess VR developers in the implementation of VR ITes. To achieve our goals, a selection of previous proposals, identifying their weaknesses and strengths, is studied in the next section. Based on the weaknesses and strengths identified, a detailed list of goals is provided. A conceptual model to describe ITes has been proposed and, once the conceptual bases have been established, a framework to implement VR ITes has been implemented on top of OpenInterface [10] and AFreeCA [11]. Finally, a case of study describing some ITes developed with this framework is presented.

2 Related Work

Most of the VR platforms that have appeared during the last years have provided some support for user interaction. At a lower level of support, some systems such as, DIVE [18] or MASSIVE [19], provide integrated support for the most popular VR devices. Besides supporting VR devices, MR Toolkit [25] includes the concept of decoupled simulation, that allows device management and VR simulation to be executed in separate computers. This factor is especially useful when resource consuming processes -i.e. visual tracking- are involved.

VRJuggler [26] and VRPN [27] extend this kind of support. Besides providing support for many devices and a decoupled simulation model, they rely on the concept of *abstract input device*. Applications are implemented using a set of abstract devices to gather user input. At run-time, any set of concrete devices that satisfies the requirements of those *abstract input devices* can be used to run the application.

However, in all these approaches, the processing required to adapt the information gathered from the devices to the requirements of the application –smoothing filters, algorithms, etc- is up to the developer. Some platforms, such as InTML [8] or OpenTracker [28], have been proposed that provide appropriate support for this task. These systems use a graphical notation to describe both the devices and the algorithms that process those data. Using these data-flow graphs includes some benefits, such as improving the comprehensibility of the ITe by depicting it as a diagram or the possibility to reuse the components in other diagrams. These two factors encourage fast prototyping and the interaction of the developers with experts from other domains –i.e. human-computer interaction experts-. However, InTML does not provide an execution platform to generate executable code from the diagrams. OpenTracker, provides an execution platform but is does not provide a graphical editor for the diagrams. Besides, this platform is discontinued nowadays.

There are, however, some platforms, such as NiMMIT [9] and the IFFI [30], inspired in the usage of data-flow models to describe ITes and that do provide an execution platform to create their diagrams and transform them into executable code. Other platforms, such as CHASM [29] or StateStream [31], even extend this concept, mixing data-flow graphs with state-based models in order to provide better support for event-based systems. However, these approaches tend to exceed the limits of the ITes. Many of these platforms consider collision detection, selection or modifications of the appearances of objects -feedback or highlight- in their diagrams. These elements are considered BITas by many authors and, as a result, these diagrams end up covering the whole description of the logic of the application. As a result, the execution platforms of these platforms are not just tools to describe ITes, but they are closed development systems to implement VR applications.

In contrast to these approaches, the intention of the tool proposed in this paper is to provide a framework to implement VR ITes that includes the features discussed in this section, but that can also be integrated in any custom VR development, letting the developer choose the most appropriate platform to describe the rest of the aspects of the system. The specific list of goals that guide the definition of the framework are:

- The framework must identify the borderline between the ITe and the BITas. The theoretical basis of 3DUIs will be reviewed, and the required models proposed, in order to identify the elements that must be addressed.
- The framework must support the relevant features identified during the study of previous proposals, such as the management of devices, a decoupled simulation model and abstract input devices.
- The framework must support the easy and fast prototyping of VR ITes - provide graphical notations when possible, reuse previous components, provide support to implement these components, etc.-.
- The framework must be easy to integrate in any VR development.

3 Conceptual Model

The framework proposed in this paper is intended to provide support to implement ITes, independently of other issues of the VR system, such as the contents or the way

the logic of the application is described. However, given the lack of maturity of the 3D field, identifying the functionality that it should and should not cover was one of its main concerns. With this purpose in mind, the theoretical bases of 3DUIs have been studied, proposing a model of VR ITe that can be used to identify the functionality that must be addressed.

The model proposed is inspired in [13] and reuses its definitions and vocabulary to describe the process required for an ITe in the context of VR applications. This model has been chosen as it successfully summarizes relevant previous models [14] and it provides a common vocabulary that removes some existing ambiguities in the field.

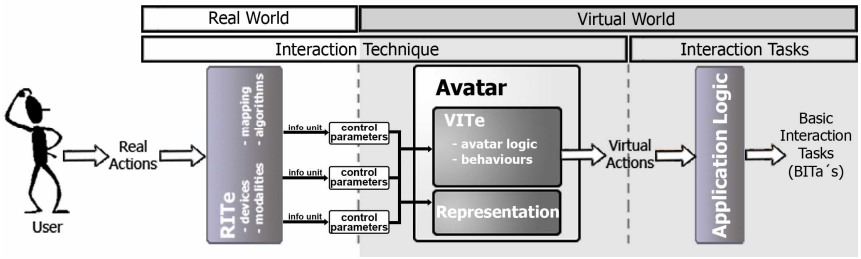

Figure 1: Elements in the model and their role in the process of an ITe.

In [13], the author describes two main functionalities for ITes: 1) to translate user's actions, produced in the real world, into the space of virtual actions, and 2) to generate the information units required by the BITas to be executed.

The first definition includes the idea that ITes cover all the process since the user performs an action in the *real world*, this real action is measured and the *avatar* transforms it into a virtual action that is processed by the *virtual world*. The three main elements in this definition will receive special attention in this model: the *real world*, the *virtual world* and the *avatar*.

The second definition assumes that the ITe has to generate the information units used by an existing set of BITas. This definition implicitly describes two stages: a first stage in which the required information units are generated -see *RITe* bellow- and a second stage that encapsulates and sends these information units to the BITas -see *VITe* bellow-.

The elements identified from the first definition –the *real world*, the *virtual world* and the *avatar*-, will be the key elements to divide the ITe in the two stages demanded from the study of the second definition. As a result, six main concepts -see Fig. 1- build this model of ITe: the three elements in the first definition -the *real world*, the *virtual world* and the *avatar*-, the two stages identified -*RITe* and *VITe*- and the borderline between these stages –*control parameters*-.

3.1. Virtual World

VR applications create a synthetic world that users can explore and interact with (the Virtual Environment (VE)). The virtual 3D space and the domain of the tasks to perform in the system characterize this *virtual world*.

3.2. Real World

The goal of a VR application is to create the *virtual world* described above. However, there are many elements outside the scope of this *virtual world* that participate in the task of letting a user experience it. The combination of all these elements outside the *virtual world* -the real 3D space, the user, the input devices, the algorithms that process the data, the presentation devices, etc.- is defined as the *real world*.

3.3. Avatar

The *avatar* is usually defined as the user embodiment in a VE [6]. Apart from representing the user in the VE, the *avatar* is the entity that allows her to interact with the objects of the VE. Thus, it can be seen as an intermediary existing in the *virtual world* but controlled by the user from the *real world*.

Under this perspective, the *avatar* provides three main functionalities. First, the *avatar* generates the *virtual actions* that allow the user to perform her tasks. Secondly, the *avatar* provides a *control interface*, so that the user can exploit its capabilities in the VE.

Finally, the *avatar* acts as the representation of the user in the VE. This representation must help the user -or other users in the case of a multi-user VE- to understand its capabilities -what it can do and how it can do it-. As it is defined in [16], this representation must be consequent to the metaphor it represents, its affordances and its constraints.

The study of the functionality of the *avatar* is relevant for the goals of the work presented, as it identifies the extents of the ITe. ITes describe how the *avatar* generates its *virtual actions* or how it is controlled. However, ITes should not affect other virtual objects –i.e. 3D widgets-. Interaction with these objects would be the result of executing BITas using the information units contained in the *virtual actions* created by the *avatar*.

3.4. Control parameters

A *control parameter* is a piece of information that is somehow relevant to control the *avatar*. They can be seen as the cords we pull to move a puppet [15] and they separate two spaces, the real world -where the *RITe* generates updated values for the *control parameters*-, and the *virtual world* -where the *avatar* processes these updated values-.
Two main functionalities are considered. First, *control parameters* can contain information to affect the *avatar's* representation –describe positions of parts of the avatar (head, hands, etc.), its status (standing, crouching, sitting, etc.)-. Secondly, *control parameters* can contain information to make the *avatar* produce a given *virtual action* -see VITe in 3.6-.
The set of *control parameters* of an *avatar* defines its *control interface*. Any user will be able to control the *avatar* by producing the appropriate values of this set of *control parameters* -see *RITe* in 3.5-. As a result, it can be seen as an extension of the concept of *abstract input device*, describing how the *avatar* is controlled. However, this

description is not strictly bound to the devices used, but can include domain-specific concepts, such as status (i.e. run, crouch), actions (i.e. SELECT, POINT, CUT), etc.

3.5. Real-world Interaction Technique (RITe)

As it was described above, during the first stage of an ITe, the information units that the ITe will transmit are gathered. This part of the process, since the user performs an action in the *real world* until it is transformed into one or more of the information units required by the *avatar* -see 3.4- is known as the *Real-world Interaction Technique* (*RITe*). It receives this name because it deals with all the elements of the *real world* that affect the ITe -the user, the input devices or any other real objects and the spatial relationships among them (i.e. the relative position of a user to a screen, which real world object the user is holding), etc.-. Real world values are measured -positions, temperatures, key-presses in a device, etc.- and processed to generate meaningful information for the *avatar* -values for a certain *control parameter*-.

Many kind of tasks can take part during this process: algorithms -digital signal processing, filtering or algorithms to transform real world coordinates into virtual world coordinates-, processes typical from multimodal interaction, such as multimodal fusion -combining different input modalities into the required information units [17]-, etc.

3.6. Virtual-world Interaction Technique (VITe)

The *Virtual-world Interaction Technique* (*VITe*) is the last stage of the ITe and covers all the process that occurs since the *control parameters* of the *avatar* are updated until it generates one or more *virtual actions* -to grab an object, or to point and select it-. It receives this name because it happens in the context of the *virtual world* -coordinates refer to the *virtual world*, messages are related to the domain of the tasks to perform in the application, etc.-. The *real world* is completely unknown during this stage.

One issue that has a big impact in the definition of this stage is that, just like it happens with ITes or BITas, there is neither universal format for *virtual actions* nor a standardized way to transmit them. These factors will vary according to the VR platform used, but three main paradigms can be identified [32]:

- **Data-flow graphs**: Some systems, such as VRML [20] or Avocado [32], describe a logical graph that traverses the 3D scene, interconnecting objects and allowing them to exchange events –virtual actions-.
- **Callback subscription**: Some systems, such as DIVE [18] or SecondLife, describe a set of high-level events –virtual actions- and provide a process that detects when any of these events occur. Objects can subscribe call-backs to each of these events, which will be invoked every time the event occurs.
- **Spatial model of interaction**: Some systems, such as AFreeCA[11] or MASSIVE [19], use the space as the layout that rules object communication. Objects define volumes in the 3D space where they can receive information -Focus [19] or receivers [11]- and volumes where the virtual actions can be received -Nimbus [19] or messages [11]-.

The nature (and format) of the *virtual actions* used is different, according to the paradigm chosen. Given that the *VITe* has to generate these *virtual actions*, as long as a unified format for *virtual actions* is not proposed, this stage of the ITe will be dependent on the VR platform chosen and, thus, *VITes* will not be reusable between VR platforms.

4 Prototype framework

The framework implemented will be explained in this section. As it was discussed in section 3.6., as long as a unified format for virtual actions is not available, it will not be possible to create a fully reusable ITe. It will be necessary to adapt the last stage of the ITe, according to the specific VR platform used.

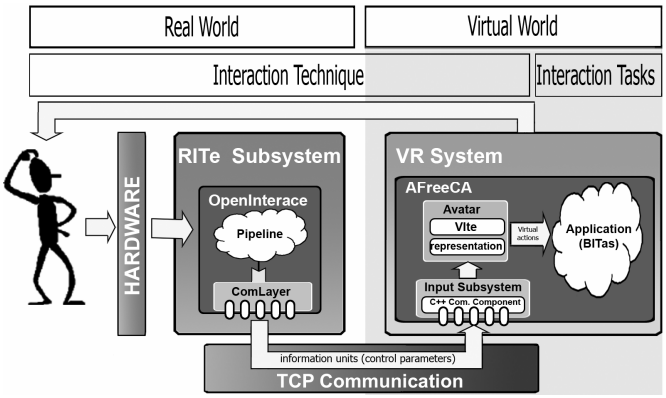

Figure 2: Structure of the framework and its relation with the conceptual model proposed

In order to limit the impact of this restriction, and according to the model of ITe proposed, the framework has been divided into three main components –see Fig. 2-. The first two components –*RITeSubsystem* and *CommunicationLayer*- are in charge of satisfying most of the requirements and goals of the framework and can be reused in any VR development. The third component –*VRSystem*- deals with the *VITe* stage and, thus, will need to be modified, according to the VR platform chosen by the development team. In order to accomplish this sample implementation of the framework, AFreeCA, a CVE development platform created at the LoUISE group [32], has been chosen.

Comentario [W1]:

The implementation details of each of these components will be detailed in the following subsections.

4.1. RITe Subsystem

This component is in charge of transforming the user real actions into the information units -values of the *control parameters*- required by the *avatar* (see section 3.3) and it has been implemented using the OpenInterface Platform [10].

OpenInterface aims at providing an extensible open source design and development workbench for supporting the rapid development of multimodal interactive systems, and it provides support for many of the features desired.

First, is uses a graphical notation that describes programs in terms of components and connections between components. Components can encapsulate anything, from a device to a processing algorithm, and they describe their communication interface with other components using input and output ports. Connections can be used to connect components' ports, describing the flow of data and messages (data-flow graph) among the components of the program.

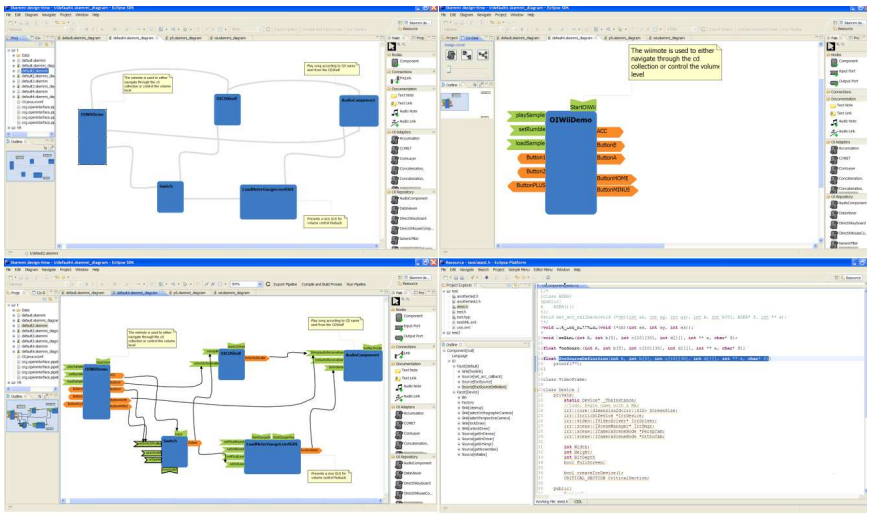

Figure 3: OpenInterface includes a visual editor called SKEMMI that can be used to design both the diagrams and the components that participate in an OpenInterface application. Diagrams can be visualized at several levels of detail and executable code can be authomatically generated from them.

Secondly, OpenInterface provides an IDE for both components and programs called SKEMMI. This IDE is integrated as a plug-in in Eclipse and it supports all the development life-cycle of an OpenInterface application –see Fig. 3-. Components can be graphically designed, and the skeleton of its source code –basically its input and output methods- generated both in C++ or Java. Developers can then implement the body of the components using Eclipse and start using them for their OpenInterface programs –called pipelines-. Pipelines can also be designed graphically, compiled and run by the OpenInterface kernel with a very low overhead.

Besides providing integrated support for the developers, OpenInterface diagrams can be visualized at diferent levels of abstraction, which facilitates interaction with experts from other fields of knowledge or final users, which can be useful given the multidisciplinary nature of these interfaces.

Thirdly, OpenInterface provides an on-line growing repository of reusable components to implement multimodal systems. These components range from components to control input devices, algorithms to filter data, and even low-level algorithms to support multimodal fusion. This repository can be extended by any developer willing to share its components with the rest of the members of the research community.

All these features make OpenInterface an almost perfect choice to implement the *RITe Subsystem*, as it can be used as an integrated tool to implement *RITes*, including graphical notations, reusability and fast prototyping capabilities. There is, however, one important issue that needed to be addressed. OpenInterface assumes that its execution kernel will lead execution, and all the elements in the program will be OpenInterface components. This philosophy was conflicting according to out goal to allow the framework to be integrated with any external VR system. Instead of forcing VR developers to encapsulate their systems as OpenInterface components, it was decided to include an OpenInterface component that allowed programs to send the values of the *control parameters* calculated –the result of the *RITe*- to an external VR application. This design allowed interoperability but, also, provided support for a decoupled simulation model, that can be important as the complexity and processing requirements of the *RITes* increases.

4.2. Communication Layer

This component is in charge of getting the updates of the *control parameters* generated by the *RITe Subsystem* and deliver them to the *VR System*, where they are encapsulated into *virtual actions* and used to execute BITas.

The data types that these components can send and receive are described in Table 1. This set of data types has been selected so that it can satisfy the requirements of any BITa –it can send the information units that any BITa can possibly require-. In order to identify the data types that should be supported, the most accepted classifications of BITas have been studied [5], [3], [6], identifying the types of data they can require –see Table 1-.

Two pieces of software have been implemented to cover this functionality, an OpenInterface component –*ComLayer*- to send information via TCP/IP to an external program and a generic C++ component to receive this information. Thus, developers simply hava to integrate this C++ component in their VR systems to receive the updates of the *control parameters* sent by the OpenInterface pipeline.

CommunicationLayer uses the OpenSoundControl formatting to transmit these updates, given its high performance. Even this decision can be technically correct, most VR platforms do not support this protocol. A second implementation of this component is currently being tackled, this time using the VRPN protocol for communication. This will allow straightforward integration of the framework with many VR systems.

Table 1: Data types supported and BITas that could support them, acording to several authors.

Name	C-style type	BITas that require this data type per author
Vector3	`double[3]`	wayfinding, exploration, selecting, positioning, rotating [5]
		navigation [3]
		movement and navigation, object manipulation and interaction [6]
Matrix44	`double[4][4]`	wayfinding, exploration, selecting, positioning, rotating [5]
		navigation [3]
		movement and navigation, object manipulation and interaction [6]
Discrete	`int`	system control [5]
		data input [3]
		object manipulation and interaction, conversation with agents [6]
Continuous	`float`	system control [5]
		data input [3]
		object manipulation and interaction, conversation with agents [6]
DataStream	`char*`	system control [5]
		data input [3]
		object manipulation and interaction, conversation with agents [6]

4.3. VR system

This component encapsulates all the details about the VE –the BITas, functions and behaviours of the objects-. Regarding the ITes, it encapsulates an application dependent implementation of the *avatar* and its associated *VITes* -see *avatar* in Fig. 2-. AFreeCA is the VR platform chosen for this part of the framework. AFreeCA aims at providing an open source platform to implement immersive and highly interactive Collaborative Virtual Environments (CVEs). The main factor that differentiates this platform from other current VR platforms is its interaction model [11, 12]. This model uses space, time and the collaborative structure of the CVE to rule the communication and interaction among the virtual objects. This allows the platform to provide native support for some interesting features, such as immersive ITes - virtual-hand, go-go, ray-casting, etc.-, collaborative manipulation, subjective perception of the VE -depending on users' locations, their collaborative contexts, etc.-. Additionally, it provides many of the technical features required in a CVE application nowadays -stereo visualization, 3D audio, haptycs, avatars with inverse kinematics, device abstraction, etc.-.

In order to integrate the VR system with the *RITe* Subsystem, an *InputSubsystem* -module used in AFreeCA to gather information from input devices- was implemented using the C++ communication component described in 4.2.

At this point, the possibility to improve the framework, providing a toolkit of reusable implementations of the most common *VITes* -virtual hand metaphors, ray-casting techniques, world in miniature, etc. [16]- was considered. Actually, this was the approach taken in some previous proposals. However, the fact that this toolkit would have been platform dependent, together with the fact that a set of standard ITes does not exist discouraged us from doing so. Instead, reusing the native implementations of these techniques that AFreeCA provides was found a more sensible approach.

5 Example usage of the framework

The following subsections illustrate the usage of the framework proposed to a sample VR system. The application reproduced a building game that allowed users to build different figures by joining LEGO-like pieces. The development followed the user centred process described in [21], interleaving four evaluations -two heuristic evaluations, a user centred evaluation and a summative comparative evaluation- with five implementation iterations.

Some initial efforts were necessary during the user task analysis -see 5.1- to identify the BITas, the virtual actions and the control parameters required. However, once this effort was done, the development of the system was facilitated.

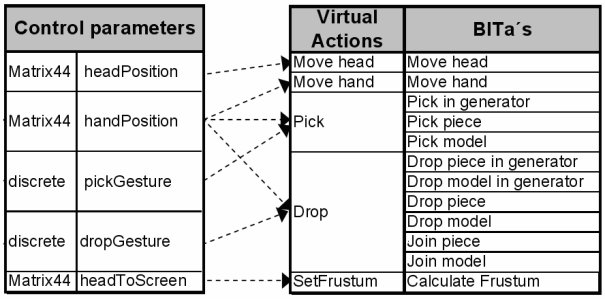

Figure 4: BITas identified (right), virtual actions that trigger them (centre) and relationships with the control parameters required to generate the actions (left).

First, once the *control parameters* required to control the *avatar* had been fixed, the development of the *VR System* –contents, tasks, avatar, etc- could be independent of the implementation of the *RITes*.

This independence together with the graphical editor and fast prototyping capabilities of the *RITe Subsystem* allowed interaction developers to easily test new techniques or refine the existing ones. This was especially useful after each of the four usability evaluations. After each evaluation, modifications on the ITes, or even new ITes, were proposed and prototyped according to the data and observations gathered from the evaluations.

This resulted in the implementations of up to eleven different prototypes, each of them using different devices and techniques. Different presentation devices–monitors, head-mounted-devices, rear-projection screens-, input devices –mouse, P5 data glove, Wiimote and Fakespace pinch gloves- and techniques –composite and increased manoeuvring, composite positioning [24]- were compared during this process. Some prototypes even evaluated the usage of voice as an input modality, but its performance was very low, abd this approach was discarded. The four most promising ITes were evaluated in a summative comparative evaluation. The detailed description of these ITes, and the relevant conclusions gathered from this study can be found in [24]. However, the current paper will only focus in one of these prototypes, in order to provide enough details about its implementation using the framework presented.

5.1. The Virtual World

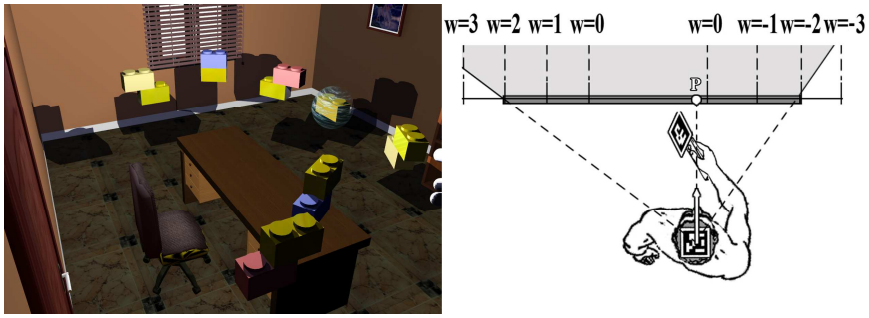

Figure 5: Screenshot of the Virtual Environment created and sketch of one of the ITes implemented.

The VE implemented reproduced a building game that allowed users to build different figures by joining LEGO-like pieces (see Fig. 5, left). Users could create pieces in the piece generator and join them together to create figures. These figures could be dropped, destroyed by dropping them in a piece generator or joined to other figures. The user task analysis performed during the first stage of the methodology highlighted that positioning -control of the position of the virtual hand [22]- and manoeuvring -precise control of the viewpoint [22]- would be the most challenging tasks that the user would need to perform in the VE. Besides, this study was very relevant for the design of the ITes in the system. This information was used to identify the BITas that had to be performed in the system -see the third column in Fig. 4-, the virtual actions required to trigger them -second column and the information units these actions would encapsulate –first column-.

5.2. The Real World

Several real world factors were considered that affected interaction:

- Space availability: An empty room of 8x3 meters was available to deploy the prototype. This allowed the usage of rear projection screens and visual tracking.
- Devices available: Several devices were available: A P5 data glove, a wiimote, mice and keyboards as input devices; Head-mounted devices, flat monitors and rear-projection screens as presentation devices; Finally, even though a commercial tracking system was not available, a visual tracking system was implemented using fire-wire cameras.
- Users: The profile of the users was studied and their abilities taken into account. The way this profile affected the ITes is described into more detail in [24].
- Ergonomics: The position and movements that the users would have to perform to use the ITes was considered, to avoid discomfort or tiredness.

The relationships among the previous elements were also studied, which was relevant for the later design of the *RITes*. As an example, during the implementation of the visual tracking system, the field of view of the cameras (devices) and the space

availability determined where the cameras had to be located (attached to the roof of the room). This affected the size and location of the ARToolkit markers on the devices used. Their final location was decided taking into account user's comfort and ergonomic criteria.

5.3. Virtual-world Interaction Techniques

In order to choose the *VITes* to use in the system, the selection tool described in [23] was used. With this purpose, the BITas required in the system and the particular features of the VE were studied. The manipulation requirements of the tasks, the limited size of the VE and the required ability of the avatar to navigate while having its hands free encouraged using a virtual hand [16] *VITe* for manipulation. The ability to have a close and detailed view of the model while constructing it encouraged the usage of first person navigation. Instead of having to implement these *VITes*, the native techniques shipped with AFreeCA were used.

5.4. Avatar

Once the *VITes* to use were chosen, the time to design the perceptual representation of the avatar came. Two possibilities were considered: a human-like avatar -they are natively supported by AFreeCA- and an invisible avatar in which the only visible part was the hand. The main drawback of the human-like avatar was that it did not help users to understand the capabilities and constraints of their avatar. This avatar presented many body parts that could not really be controlled by the user -except for the reduced control inverse kinematics permitted-. Another negative point was the fact that its body sometimes occluded other objects of the environment. In contrast to the inconveniences of this human-like avatar, the invisible avatar provided a simple and functional design, representing its exact capabilities. It avoided the user to get wrong ideas or expectations and, also, minimized occlusions. As a result, - and according to the ideas described in 3.3. - the invisible avatar was chosen.

5.5. Control parameters

As it explained in section 3.4, control parameters describe the information units required for the BITas of the system and the ones required to move the representation chosen for the avatar. Given the invisible avatar chosen, no additional control parameters are required to move the avatar representation itself. As a result, the only control parameters required are the ones obtained in the study of the BITas of the system -see the third column in table 3-.

5.6. Real-world Interaction Techniques

This section describes one of the prototypes implemented. The prototype used a rear projection screen of 1.65x1.20 square meters, a wiimote and two ARToolkit markers,

one on the head and another one on the top of the wiimote –see Fig. 5, right-. The rear projection screen was used as a window to the virtual world, displaying the part of the VE that would be visible according to the position of the user's real head, the real position and size of the screen, and the location of this window in the VE -see the grey area in Fig. 5, right-. In order to see objects outside this grey area, a way to rotate the window in the VE was required. To achieve this, when the user looked to the side of the screen -see projection ***P*** *i*n Fig. 5, right-, the virtual window started rotating, showing other parts of the VE. The angular velocity depended on the point where the user was looking at -see ***w*** in Fig. 5, right-. To control the movement of the hand, a stretch go-go technique was used. Also, the A and B buttons in the wiimote were used to pick and drop elements of the VE.

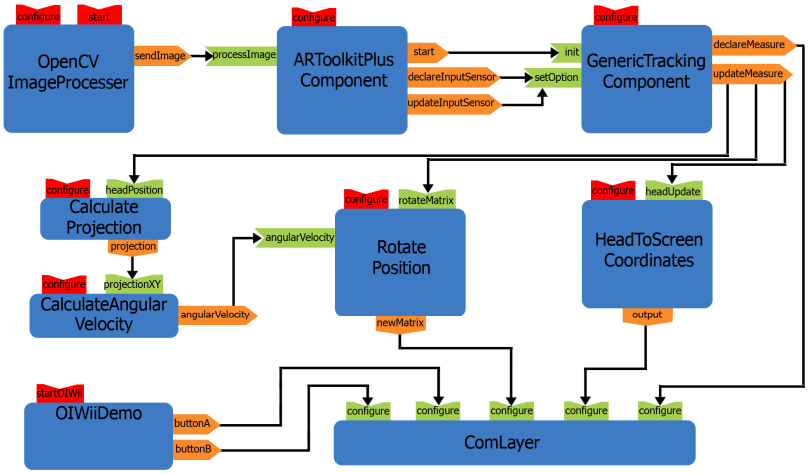

Figure 6. OpenInterface Pipeline used for prototype number 4.

These *RITes* were implemented using the OpenInterface pipeline depicted in Fig. 6. The red pins in every component identify components' services that are invoked by the execution kernel during the start-up. The services are usually used to configure the component or to start the execution of the *RITe* thread.

In the first line of the figure, three components collaborate to implement the visual tracking system. The first component (*OpenCVImageProcesser*) is used to get an image from a camera. The second component (*ARToolkitPlusComponent*) encapsulates a modified version of the ARToolkit Plus library, used to detect the 3D position and orientation of the markers in the image captured. Modifications to the library were required to allow the effective detection of several non-related markers in the same image, a feature that is not correctly supported The third component (*GenericTrackingComponent*) is used to transform the coordinates generated by *ARToolkitPlusComponent* from the system of reference of the camera to a useful system of reference for the application, that is, to *virtual world* coordinates. It also encapsulates some noise reduction filters, implemented to improve the quality of the

tracking. The component produces a pair containing the name of a *control parameter* -i.e. headPos, handPos,etc.- and a transformation matrix as a result, that are sent to other components through the *updateMeasure* port.
The second line of components implements the *composite manoeuvring* technique [24] that controls the position and orientation of the virtual head. *CalculateProjection* uses head position updates –and ignores hand updates- to calculate the 2D point of the screen where the user is looking at –see point **P** in Fig. 5.right-. This value is used by *CalculateAngularVelocity* to calculate the angular velocity of the virtual head according to the point the user is looking at and the size and location of the screen. This value is used by *RotatePosition* to affect both the position of the head and the hand. *RotatePosition* contains a transformation matrix that accumulates the rotations performed by the user using the *low-precision manoeuvring technique* [24]. This matrix is updated according to the current angular velocity and the pass of time and it is applied to any transformation that refers to the *virtual world.* This second set of components illustrate an important concept about *RITe: RITes* do not simply deal with the devices and how the information gathered from them is filtered or adapted to the requirements of the application, *RITes* can also encapsulate algorithms or rules to describe complex and rich ITes.

Finally, *HeadToScreenCoordinates* calculates the position of the user's head from the system of reference of the screen. This is used by the *avatar* to calculate the projection volume (OpenGL frustum) displayed in the rear projection screen. *OIWiiDemo* is used to retrieve users key presses of the wiimote. All the information unitsgenerated by this pipeline are finally sent to the *VR System* using *ComLayer*.

6. Conclusions and future work

The work presented in this paper describes a framework to assess VR developers in the design and implementation of ITes.

In order to identify the functionality that the framework had to address but, also, to guide the later design and implementation of the ITes, a novel model of ITe has been proposed. This model is the result of the study of previous models of the elements of 3DUIs, but it goes deeper in the analysis of the elements that participate in the ITes -*real* and *virtual world* and the *avatar*- and the stages required for an ITe, dividing it into two sub stages -the *RITe* and the *VITe*-. This model had an important impact during the development of the final framework. First, the stages identified in the model –*RITe* and *VITe*- influenced the components of the framework –a *RITe Subsystem*, a *VR System* and a *Communication Layer* to interoperate them-. Secondly, the study performed during the definition of the model revealed that, given the lack of both standard ITes or a standard format for virtual actions, any *VITe* would be dependent on the VR platform used. As a result, it is currently impossible to provide a toolkit with a closed set of ready-to-use ITes that can be reused by any VR platform. Figueroa stated in [7] that the definition of standard ITes and BITas would be the key element to make the implementation of these toolkits possible. Even though this statement remains true, it would be the definition of a standard format for *virtual actions* what would allow different platforms to reuse complete ITes. Once reusing

ITes was possible, the standardization of the most used ITes and BITas would be simply a matter of time.

Besides the ITe model, the second, and major contribution of this work, is the ITe development framework presented. This framework shares the strengths identified in previous proposals, but it avoids some of their deficiencies. These features are summarized in Table 2.

Table 2: Summary of the features achieved by previous existing platforms and features included in the framework proposed.

	VR Devices	Decoupled simulation	Abstract input device	Graphical notation	Reuse	Visual editor	Execution platform	Integration
DIVE	yes	no	no	no	no	no	no	no
MASSIVE	yes	no	no	no	no	no	no	no
MRToolkit	yes	yes	no	no	no	no	no	no
VRJuggler	yes	yes	yes	no	no	no	no	no
VRPN	yes	yes	yes	no	no	no	no	no
InTML	yes	no	no	yes	yes	yes	no	no
OpenTracker	yes	no	no	yes	yes	no	yes	no
NiMMIT	yes	no	no	yes	yes	yes	yes	no
IFFI	yes	no	no	yes	yes	yes	yes	no
CHASM	yes	no	no	yes	yes	yes	yes	no
StateStream	yes	no	no	yes	yes	yes	yes	no
OpenInterface +AFreeCA	yes	yes	yes	yes	yes	yes	yes	yes

The framework -implemented on top of OpenInterface Workbench and AFreeCA- facilitates the implementation of the ITes. On the one hand, the usage of OpenInterface facilitates the design and fast prototyping for the *RITe* stage, covering most of the features the ITe development platforms must contain –graphical notations, IDE, modularity, component reusability and fast prototyping capabilities-. These features, together with the on line repository of ready-to-use components can save many developing efforts. On the other hand, the usage of AFreeCA facilitated the design of the VITe stage, given that it provides a set of the most usual *VITes*. Finally, both elements can interoperate using a decoupled simulation model.

Thirdly, the prototypes developed revealed that the contributions presented in the paper can have a good impact on the development process. First, the model supports the design of the ITes, helping in the identification of relevant factors. Secondly, the framework supports their development. Besides, the architecture of the framework allows the *RITe Subsystem* and the *VR System* to be implemented and tested with a high independence of each other, once that the *control parameters* that they will exchange have been determined.

Several lines of research can continue the work presented here. First, the usage of the proposed framework in the development of other VR systems is considered as relevant. This way, its ability to adapt to the requirements of different applications could be checked. Also, as a side effect, more OpenInterface components would be

generated during these developments, components that could be reused in the future, saving development time and efforts.

Secondly, the definition of a universal format for virtual actions would be definitively interesting. As it has already been discussed, this would be a key element to produce reusable toolkits of VR ITes. A detailed study of the different formats of virtual action that the most common VR platforms use and the way they are processed by the VE would be the first step. Then, it would maybe be possible to suggest a format of virtual action that could be reused in all these platforms.

Acknowledgements

The current work has been partially supported by the Junta de Comunidades de Castilla-La Mancha (PEII09-0054-9581) and the Ministerio de Ciencia e Innovación (TIN2008-06596-C02-01).

References

1. Pittarello F., Celentano A., Interaction locus: a multimodal approach for the structuring of virtual spaces. Proc. Of HCITaly Symposium, Florence, Italy September 2001.
2. Herndon K.P. , van Dam A. , Gleicher M., The Challenges of 3D Interaction: a CHI '94 workshop. ACM SIGCHI Bulletin, Vol. 26, No. 4, October 1994, pp. 36-46
3. Boyd D., Sastry L., Development of the INQUISITIVE Interaction Toolkit - Concept and Realisation. User Centred Design and Implementation of Virtual Environments (UCDIVE) Workshop, York, United Kingdom, September 1999.
4. Barrilleaux J., 3D User Interfaces with Java3D. Manning Publications, 2001.
5. Bowman D.A. ,Kruijff E. ,LaViola Jr. J.J., Poupyrev I., An Introduction to 3-D User Interface Design. Presence, Vol. 10, No. 1, February 2001, pp. 96-108.
6. Sutcliffe A., Multimedia and Virtual Reality: Designing Multisensory User Interfaces. Lawrence Erlbaum Associates, 2003.
7. 3DUI community web page. URL http://www.3dui.org/
8. Figueroa, P., Green, M., Hoover, H. J. InTml: a description language for VR applications. In Proc. of the Seventh international Conference on 3D Web Technology. Tempe, Arizona, USA, February 24 - 28, 2002. Web3D '02. ACM, New York, NY, 53-58.
9. Boeck J.,Vanacken D., Raymaekers C., Coninx K., High-Level Modelling of Multimodal Interaction Techniques Using NiMMiT, Journal of Virtual Reality and Broadcasting, Volume 4(2007), no. 2, September 2007.
10. Lawson J-Y., Al-Akkad A., Vanderdonckt J., Macq B., An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-The-Shelf Heterogeneous Components. Proc. of the First ACM SIGCHI EICS, ACM Press, USA, July 14-17, 2009.
11. Martínez D., García A.S., Martínez J., Molina J.P., González P., A model of interaction for CVEs based on the model of human communication, Journal of Universal Computer Science, Vol. 14, No. 19, November 2008, pp 3071-3084.
12. Martinez, D., Molina, J.P., Garcia, A.S., Martinez, J., Gonzalez, P., AFreeCA: Extending the Spatial Model of Interaction, International Conference on Cyberworlds, pp. 17-24, 2010
13. Molina J. P., García A. S., Martínez D., Manjavacas F. J., Blasco V., González P., An Interaction Model for the TRES-D Framework. Proc. of 13th IEEE Mediterranean Electrotechnical Conference (MELECON 2006), special session "New interaction

paradigms in Virtual Environments", Benalmdena, Málaga, May 16-19, 2006. Electronic Proceedings.
14. Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., Computer Graphics: Principles and Practice. Second Edition in C. Addison-Wesley, 1996
15. J. P. Molina, Un enfoque estructurado para el desarrollo de interfaces de usuario 3D, doctoral thesis, February 2008, University of Castilla- La Mancha.
16. PoupyrevI., Weghorst S., Billinghurst M., Ichikawa T., Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques. Computer Graphics Forum, Vol. 17, No. 3, September 1998, pp. 41-52.
17. J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, R. Young, Four Easy Pieces for Assessing the Usability of Multimodal in Interaction the CARE Properties, Human Computer Interaction, Interact' 95, pp. 115-120.
18. Frécon E., Stenius M., DIVE: A Scaleable network architecture for distributed virtual environments, Distributed Systems Engineering Journal , Vol. 5, No. 3, September 1998, pp. 91- 100.
19. Greenhalgh C., Purbrick J., Snowdon D., Inside MASSIVE- 3: flexible support for data consistency and world structuring. In Proc. of the Third international Conference on CVEs, San Francisco, California, USA. CVE '00. ACM Press, 119-127.
20. Web3D consortium. http://www.web3d.org
21. Gabbard, J.L., Hix, D., Swan, J. E., User-Centred Design and Evaluation of Virtual Environments. IEEE Computer Graphics and Applications Vol. 19, No. 6 November 1999, pp. 51-59.
22. Bowman D.A., Kruijff E., LaViola J.J., Poupyrev E I. 3D User Interfaces: Theory and Practice. Addison-Wesley, 2004
23. García, A.S. , Molina, J.P. , González, P., Exemplar VE design guidance tool for selection and manipulation interaction techniques. In proc. of 11th International Conference on HCI, HCI International 2005, Las Vegas, Nevada, USA, July 2005.
24. Marítnez, D., Kieffer, S, Martínez, J., Molina, J. P., Macq, B., González, P., Usability evaluation of virtual reality interaction techniques for positioning and manoeuvring in reduced, manipulation-oriented environments. The Visual Computer, Volume 26, Numbers 6-8, pp. 619-628.
25. Mark, C.S., Green, M., Liang, O. , Sun, Y., Decoupled simulation in virtual reality with the MR Toolkit. ACM Trans. Inf. Syst. 11, 3 (July 1993), pp. 287-317.
26. Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., Cruz-Neira, C. VR Juggler: a virtual platform for virtual reality application development. In SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses, pp. 1-8, ACM, New York, USA
27. Russell, M., Hudson, T.C., Seeger, A., Weber, H., Juliano, J., Helser, A.T. VRPN: A device-independent, network-transparent vr peripheral system. In Proceedings of the ACM symposium on Virtual reality software and technology (VRST '01). ACM, New York, USA, pp. 55-61.
28. Reitmayr, G., Schmalstieg, D. (2001). OpenTracker-an open software architecture for reconfigurable tracking based on xml. In VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01), 285, IEEE Computer Society, Washington, DC, USA.
29. C. Wingrave and D. Bowman. "CHASM": Bridging Description and Implementation of 3D Interfaces. In Proc. of IEEE VR Workshop on New Directions in 3DUIs, pp. 85–88, 2005.
30. Ray, A., Bowman, D. A., Towards a system for reusable 3D interaction techniques. In Proc. of VRST '07, pp. 187–190, 2007.
31. Haan, G., Post, F. H., StateStream: a developer-centric approach towards unifying interaction models and architecture. In Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems (EICS '09). ACM, New York, USA, pp. 13-22.
32. Tramberend, H., Avocado: A Distributed Virtual Reality Framework. In Proceedings of the IEEE Virtual Reality (VR '99). IEEE Computer Society, Washington, DC, USA.