A Model-based Approach for Developing 3D User Interfaces

By Juan Manuel González Calleros

A dissertation submitted in fulfillment of the requirements for the degree of

Certificate of in-Depth Studies (DEA)
in Management Sciences
Option “Information Systems”

of the Université catholique de Louvain

Committee in charge:

Prof. Jean Vanderdonckt, Advisor
Prof. Karin Coninx, Examiner
Prof. Jaime Muñoz Arteaga, Examiner

Summer 2006

Table of Contents

Acknowledgement

I would like to express my thanks to:

- My advisor, Professor Jean Vanderdonckt, for his constant support and enthusiasm regarding my work.

- Professors Karin Coninx and Jaime Muñoz Arteaga for accepting to participate to the jury of this dissertation.

- My family who encourage me every day.

- My friends who always believe in me.

- My wife, Josefina Guerrero García and our child, who has been a constant sustain in this adventure.

This dissertation was supported by:

- The Program Alßan, a high level scholarship program specifically addressed to Latin America.
- The Mexican scholarship program CONACYT to support studies abroad.

Abstract

Three-dimensional (3D) interaction is an exciting field of research. Today, the development life cycle of 3D user interfaces (UIs) mostly remains an art more than a principled-based approach. Several methods have been introduced to decompose this life cycle but rarely provide the design knowledge that should be typically used for achieving each life cycle step. In addition, the development life cycle is more focusing directly on the programming issues than on the design. Model-driven development is a development paradigm that relies on model engineering i.e., in the power of models to build computer systems. This thesis applies such approach to develop 3D UIs to cover the lack of methodologies in this domain. First on the transformational development paradigm that consists in the progressive refinement of abstract models into concrete models, until program code, we propose an ontology of concepts defining various viewpoints that can be maintained on a 3D UI system. Viewpoints are hierarchically structured depending on their level of abstraction. They describe user tasks, classes of objects, presentational and behavioral aspects of UIs, context of use, and a set of mappings between these representations. The underlying mathematical formalism, is a graph structure (directed, identified, labeled, constrained, and typed graphs) that transform one viewpoint into another by the application of conditional graph rewriting rules gathered in graph grammars. These enable expressing a wide variety of transformational heuristics so as to be able to express multiple development paths, in which we include the 3D UIs path. Our proposal extends UsiXML (User interface eXtensible Markup Language), which already covers 2D and vocal UIs. Ontologies and transformations may be stored in an XML format, called UsiXML, that will be transformed to code for 3D UIs.

Chapter 1 Introduction

Nowadays, software development is evolving exponentially, with the hand of hardware and technological innovations. New languages or paradigms are required to provide solutions to those new technologies. This contributes to frequent changes in focus and as result of new areas of deployment, [Khaz00].

The Computer Human Interaction (CHI) field studies how to evaluate, analyze, design and develop usable and useful software. CHI field is supported by many disciplines such as: psychology, sociology, cognitive disciplines, among others. One of the concerns in this discipline is the development of User Interfaces (UI)

UIs development for software developers, designers and maintainers is a complex activity. Each time that a technological innovation appears they have to learn new skills that could be: a new programming language, the manipulation of new hardware architectures or new design methodologies.

With the hand of the innovations we found a new emerging approach: 3-dimensional User Interfaces (3DUI). Nowadays, many resources exist as 3-dimensional virtual reality scenes or worlds for informational, public, pedagogical, and rehabilitation purposes. Normally those worlds are devoted to show virtual environments, sophisticated animations, games, but not User Interfaces, which traditionally are developed in 2-dimensions.

Three-dimensional (3D) interaction is an exciting field of research that promises to allow users to perform tasks freely in three dimensions rather than being limited by the 2-dimensional (2D) desktop metaphor of conventional graphical interfaces. For some computer-based tasks, pure 3D representations are clearly helpful and have become major industries: medical imagery, architectural drawing, computer-assisted design and scientific simulations, (Shneiderman, 2003). Those systems traditionally are associated to complex and expensive technologies. Games industry is leading the market and showing the potential of rendering 3D graphics in a desktop computer.

1.1 Research Motivation

Several methods [Bowm04, Cele01, Fenc01, Geig01, Neal01] have been introduced to decompose this life cycle into steps and sub-steps, but these methods rarely provide the design knowledge that should be typically used for achieving each step. In addition, the development life cycle is more focusing directly on the programming issues than on the design and analysis phases. This is sometimes reinforced by the fact that available tools for 3D UIs are toolkits, interface builders, rendering engines, etc. When there is such a development life cycle defined, it is typically structured into the following set of activities:

- The **conceptual phase** is characterized by the identification of the content and interaction requests. The meta-author discusses with the interface designer to take advantage of the current interaction technology. The interface designer receives information about the content. The result of this phase is the production of UI schemes (e.g., written sentences, visual schemes on paper) for defining classes of interactive experiences (e.g. class Guided tour). Conceptual schemes are produced both for the final users and the authors. The meta-author has a deep knowledge of the content domain and didactic skills too. He/she communicates with the final user too, in order to focus on didactic aspects of interaction.
- In the **implementation phase**, the UI designer builds the final user interface and the author interface on the basis of the UI schemes. The results of this phase are available as tools for the authors, which can be manipulated without a deep knowledge of computer science world. It is important to note that this implementation phase can be a personalization or a sub-setting of existing tools, rather than a development from scratch.
- In the **content development phase**, authors choose among the available classes of interactive experiences and instantiate the one that fits their particular needs (e.g. Guided tour, paths). The take advantage of a number of complementary subjects: editors (writer, 2D graphic artist), 3D modeler, world builder.

- In the **final user interaction phase**, the final user interacts with the contents of the 3d world, composed by the author, through the interface implemented by the interface designer. The final user interaction is monitored in order to improve both the usability of the interface and the effectiveness of content communication.

As opposed to a content-centric approach, some other authors advocate a user-centered approach; hence, involvement of users in the requirements analysis and evaluation are essential for achieving a usable product. They also argue for separating the conceptual part from the rest of the life cycle to identify and manage the Computing-Independent Models (CIM as defined in the Model-Driven Engineering –MDE) from the Computing-Dependent part. This part is in turn typically decomposed into issues that are relevant only to one particular development environment (Platform-Specific Models –PSM) as opposed to those issues which remain independent from any underlying software (Platform-Independent Models–PIM). In the MDE paradigm promoted by the Object Management Group (www.omg.org), it is expected that any development method is able to apply this principle of separation of concerns, is able to capture various aspects of the problem through models, and is capable of progressing moving from the abstract models (CIM and PIM) to the more concrete models (PSM and final code). The goal of this dissertation is to demonstrate the feasibility of a MDE-compliant method that is user-centered as opposed to contents-centric for developing 3D UIs.

What is more, it is necessary to clearly identify which are the problems related to 3D solutions. The question for some authors is whether doing 3D software development or not? This Shakespearian dilemma deals with the design decision of presenting 3D User Interfaces because is necessary or just because is attractive. [Cock01] tried to distinguish between the real necessity of using 3D representations and the overuse of this kind of user interfaces development just because they are in vogue. In [Shne02] the same subject was discussed, for general purpose applications. Both surveys concluded that 3D presentation is not just more attractive for the users but also provide a best option for developers to manage the information visualization issue. In their review of applications, [Cock01] offer some examples of user performance in 3D applications. They found that user preferences are on the use of 3D systems, as users found them more natural to use. Also, that 3D user interfaces are better to use for cognitive reasons, as it exploits the spatial memory and cognition of humans.

Even so, we consider that we need, clear guidelines for designing 3DUI in a coherent way. Some advises to choose the appropriate representation for the problem instead of adapting an existing solution to the problem. In this concern, the user task plays an important role, as they are helpful to identify the true nature of the task, so, identify if its 3D in nature or not. Task models are not considered during the design phase of 3DUI.

Similarly, user performance in 3D is an important aspect to evaluate. Until now the vast majority of 3D applications were created for a specific group of users and specific application domains; for instance, for psychological therapy, airplane and car training, natural science simulations, among others. In all these tools it is possible to measure the user performance in front of the system by analyzing some attributes such as: time to learn its use, speed of performance, rate of human errors, and human retention over time. Even that the previous variables could be measured with experiments, modern UI development require to take into account the system context of use, whether social, organizational and cultural, so as to consider individual differences among potential users, [Shne02].

Actually, the common attitude for 3DUI development is to start from scratch rather than reuse components, as is the common way to generate 2D software, [Poup00]. Some problems on the usability field; are provoked by the lack of methodologies to evaluate (quasi)automatically the developments. The acceptance of virtual environments (VE) technology requires scrupulous optimization of the most basic interactions in order to maximize user performance and provide efficient and enjoyable virtual interfaces. There is a need for a methodology that could help also to evaluate the development of 3DUI.

In addition to the problems above described, our last motivation for conducting this research is the lack of a software tool, toolkit, for developing 3DUI. Different toolkits exits to create 2D user interfaces that help developers in doing this task easier, the counterpart in 3D exist but not for the purpose of creating UI. So we need a software Framework capable to group the solution of the described problems, so as to be easy to use.

1.2 The scope of the research

1.2.1 Human Computer Interaction

This research is located in the *Information Systems* Management area, in particular in the discipline of *Engineering* for Computer-Human Interaction (CHI). This discipline is the crossroad of *software engineering* (the application of a systematic, disciplined, quantifiable approach to develop, operate, and maintain software; the application of engineering to software) and *CHI* (concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them), [Hewe92].

The *User interface* (UI) is the aggregate of means by which people (the users) interact with a particular machine, device, computer program or other complex tool (the system). The user interface provides means of: Input, allowing the users to control the system, Output, allowing the system to inform the users (also referred to as feedback), [Wiki05], see in Figure 1-1 a 3DUI which a special case of UI.

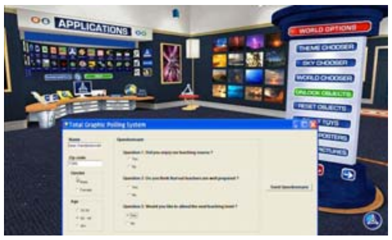


Figure 1-0-1 Windows 3DNA environment

1.2.2 Model Driven Architecture Development

Model-driven development is a development paradigm that relies on model engineering i.e., in the power of models to build computer systems. It advocates that software development should be guided as much as possible by the construction, and refinement of software models at various levels of abstraction. Most of current development methodologies have been influenced by, can be affiliated to, or are totally in debt with, this paradigm, for instance: object-oriented methodologies, database engineering, or agent-oriented methodologies.

Model-based design purpose is to identify high-level models that allow designers to specify and analyze interactive software applications from a more semantic-oriented level rather than directly pass to the implementation level.

More recently, along with the *Model Driven Architecture (MDA)* proposal [Mill03], model processing and transformation have gained particular importance in the software engineering literature, [Limb04c]. The main motivation of these works is to tackle the problem of computing platform heterogeneousness. For this purpose MDA defines a set of abstraction layers able to factor out specificities of implementation platforms. In this context, explicit model-to-model transformations enable the realization of the development process.

The software engineering *transformational development*, a paradigm consisting in the progressive refinement of abstract models into concrete models, until program code [Somm99]. This research expects to apply transformational development concepts until code for 3D User Interfaces (3DUIs), enlarging the 2D Ontology already specified in *UsiXML*.

UsiXML (which stands for USer Interface eXtensible Markup Language) is a XML-compliant markup language that describes the UI for multiple contexts of use such as Concrete User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. In other words, interactive applications with different types of interaction techniques, modalities of use, and computing platforms can be described in a way that preserves the design independently from peculiar characteristics of physical computing platform [Limb04a].

An *engineering approach for model-based* design should address at least four main issues. First, support to flexible and expressive notations. Second, build systematic methods to support the specification during the design. Third, it must give support to reuse the models. Finally, support to add or remove detail from the task model during the designing phase [Pate00].

Model generation, in software development different kinds of models could be used. However, two categories have been identified to abstract user interfaces, the design centered on the user and the design centered on the environment. In general, the common mistake is that developers design software based on the target or the environment, rather than taking into account the user as the center of the design. This shortcoming creates problems of usability of the software and low performance of the users.

Different kinds of models imply different representations as they have different points of view and analysis. Task models are centered on the user an help developers to understand the main activities of an application (domain model), represent the agreement between all the agents involved in the development (users, designers, etc), design based on the user (conceptual model), is useful to evaluate the usability of the system and gives support to generate help bars or tools based on the task. In other words, task model is useful for both designers and final users. But task model is just the starting point for the design.

1.2.3 Virtual Reality

Virtual reality describes an environment that is simulated by a computer. Most virtual reality environments are primarily visual experiences, displayed either on a computer screen or through special stereoscopic displays, but some simulations include additional sensory information, such as sound through speakers or headphones. Some advanced and experimental systems have included limited tactile feedback. Similarly, [Krei01] defines virtual reality as a kind of reality that is computer-generated, and at least at present consist in replacing the normal sensory environment by another environment that mimics as closely as possible the normal sensory environment without being the normal sensory environment.

1.3 Research Goals

The goals of this research consist in specifying, designing, and developing a Model-Based Approach for Developing 3D User Interfaces (3DUI), a software environment aimed at:

1) Provide the developer the chance to describe a 3DUI in a XML-based language, UsiXML.
2) Help the developer to create a 3DUI in a graphic editor or using as a starting point a UsiXML file description and render it in the tool.
3) Contribute to the creation of 3DUI instead of traditional 2D, easily.
4) Provide developers a set of predefined 3D widgets that helps them to create the UI.
5) Testing the usability and the accessibility of virtual reality worlds contained in Web sites or stand-alone against empirically validated guidelines and design rules.
6) Providing developers with assistance in improving detected usability and accessibility deviations while designing.
7) Repairing deviations under the supervision of the developer with a mixed-initiative process.
8) Enhancing the world by adding hooks and hints for supporting the navigation in the world through multiple and alternate sensory modalities for disabled people. Such modalities should foster feeling, sensing, and hearing the world through appropriate interaction devices. For example, the tool should prepare the world for screen sonification to facilitate navigation.

By improving and ensuring the quality of virtual reality worlds, it is expected that A Model-based approach for developing 3D user interfaces will contribute to a social goal of making worlds more usable for traditional visitors and more accessible for disabled ones. Three-dimensional environments are greatly appreciated by some users and are helpful for some tasks. They have the potential for novel social, scientific, and commercial applications.

1.4 Research Beneficiaries

Software development in its last phase, which is implementation, could be divided in two sub-phases, the UI programming (static part) and system functionality programming (dynamic part). Considering some studies, that have revealed that the time devoted to create User Interface is around 44% [Boeh88] and 48% [Myer92] of the total time required in the implementation phase, we notice that UI development is not just a complex task but also time consuming. We identified four actors of the software development cycle that would be benefited with this research: the software maintenance, the developer, the designer and the user of 3DUI.

1.4.1 Software Maintenance

Software maintenance refers to the process of enhancing and optimizing deployed software (software release), as well as remedying defects. Updating the UI is a mandatory task when new requirements appear. Requirements are linked with the organizations necessities, and the organizations are constantly changing, in structure, in the way they process their products, in the hardware or software that they used, etc. As a consequence, they "change user's requirements", software maintenance is required. Actually, the dynamicity of the development of UI represents difficulties when the same User Interface should be developed for multiple contexts of use such as multiple categories of users (e.g., having different preferences, speaking different native languages, potentially suffering from disabilities), different computing platforms (e.g., a mobile phone, a Pocket PC, an interactive kiosk, a laptop, a wall screen), and various working environments (e.g., stationary, mobile).

Updating the UI becomes so hard, as previous specifications could not be applied to the new context, and the time required to do that becomes a constraint. Maintainers using this approach will have the support required to facilitate their work when they need to change the UI. Actually, UsiXML language support multi-context, multi-platform, multi-device and multi-modal software development. Our research would increment the capacity of the language to 3DUI.

1.4.2 Software Designer

Software Designers are in charge on designing the system structure. With the lack of methodologies for design 3DUI, Designers will be benefited with this research as we will provide a Framework with models and guidelines that will support their task of designing solutions in 3D.

1.4.3 Software Developer

Software Developers are sometimes called programmer and are concerned with the implementation part in the software development process. It would help developers to be mere productive but it won't eliminate them from the software development process. They would be able use their time better, to use their creativity in other issues but not the 3DUI development. Developers would have productivity gains, so as the enterprise in which they work.

1.4.4 End Users of three-Dimensional User Interfaces

The last, but not less important, beneficiary of this work is the user. Considering the key role that the user played in the success or failure of software programs, developers, designer and software maintainers have to think carefully on the user. This aspect is covered by this research, as the development of UI starts from specifying the user task so as the domain in which the task is done. The user will be benefited if the development, the design or the maintenance of UI is done following our methodology, as the results will be based on how the user task is done actually. If our guidelines cover the enhancing of the world by adding hooks and hints for supporting the navigation in the world through multiple and alternate sensory modalities for disabled people. Such modalities should foster feeling, sensing, and hearing the world through appropriate interaction devices. For example, the tool should prepare the world for screen sonification to facilitate navigation. This will benefit the disabled people so as the common users.

However, this work does not ensure the complete satisfaction of users, as the functionality of the systems is not accomplished by the methodology. Users could be happy with the presentation, i.e. the 3DUI, but not with the functionality, phase two of the implementation step, that is straightforward of our work.

1.5 Thesis

Thesis statement

This dissertation addresses the shortcomings previously outlined for achieving transformation-driven development of 3D user interface. This dissertation provides an:

(1) ontological framework based on an explicit and rigorous representation of concepts relevant to 3DUI development.

(2) methodological framework based on the ontological framework previously introduced. This methodological framework introduces a new paradigm for 3DUI development called ***model-based approach for the development of 3DUIs*** that is characterized by the following principles:

Transformation driven: a development method is composed of development stages. A development step is a transition from one stage to another one. Development steps rely on explicit and rigorous transformation catalogs.

Multiple-path: The context of development projects may involve variable arrangements of development steps. A development path refers to a particular arrangement of steps. Multi-path development refers to the capacity of a method to accommodate to various development paths.

Validation

Two kinds of validation are provided to assess the validity of this thesis. Firstly, an internal Validation which includes: a) theoretical validation that confronts the methodological framework introduced by this thesis to the requirements identified after a state of the art of existing transformation-driven development methods; b) a practical validation is provided by illustrating how the methodological framework can be instantiated on two case studies. Secondly, an external validation, where the system developed would be evaluated either with end users, testers.

1.6 Reading Map

In addition to the introduction and the conclusion, this dissertation is organized in four chapters.

Chapter 2 reports on some significant pieces of related work to the paradigm of model-based development and 3D applications. We identify the different approaches to the development of 3D User Interfaces. A set of observations and shortcomings is raised in conclusion of a comparative analysis. From these observations, we establish a list of requirements for addressing the observed shortcomings. This list of requirements will help us to assess the appropriateness of our solution.

Chapter 3 introduces a Taxonomy of 3D User Interfaces. With this taxonomy we identify the different components used in virtual application, the context of use, the nature of the task.

Chapter 4 presents a structuring of concepts identified in viewpoints, capturing various levels of abstraction that can be maintained on a 3DUI. After that, we present the abstract syntax that has been used to represent our concepts, namely: directed, identified, labeled, and typed graphs. We present two concrete syntaxes (i.e., graphical and textual) used to represent our concepts. Finally, we introduce the supporting software tool for this research

Chapter 5 illustrates the principles of model-based transformational development for one case study. The first one concerns the development of an on-line polling system. We conclude this chapter by an evaluation of the three case studies.

Chapter 6 concludes by discussing the validation and appropriateness of the solution proposed in this dissertation. Our contributions are summarized and future works are proposed.

Chapter 2 State of the Art

2.1 Description of 2D User Interface development

Two elements define the smallest common denominator of what a User Interface (UI) is [Limb04c], whether 2D or 3D: the *presentation* or *look*, and the *dialog* or *feel*. In the context of 2D UI, various approaches have been followed to develop UI. Diana methodology [Bart88] characterizes these approaches in: *internal view,* what is relevant for the UI developer; the *external view,* the UI perceived by the end user; and the *conceptual view,* what is relevant for the UI in designer.

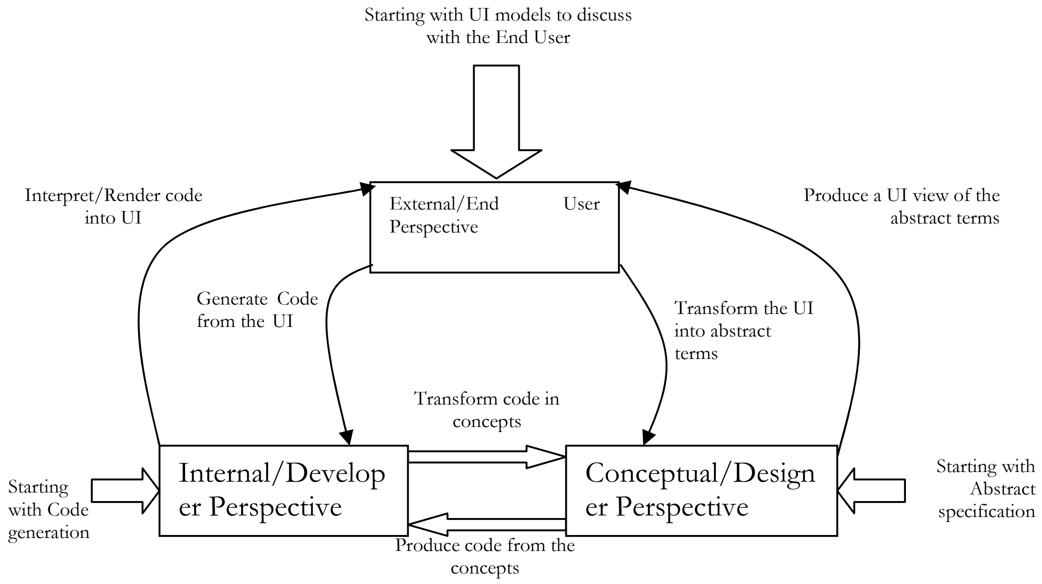

Starting with UI models to discuss with the End User
Interpret/Render code into UI
External/End User Perspective
Produce a UI view of the abstract terms
Generate Code from the UI
Transform the UI into abstract terms
Transform code in concepts
Starting with Code generation
Internal/Developer Perspective
Conceptual/Designer Perspective
Starting with Abstract specification
Produce code from the concepts

Figure 2-1 Development of 2D UIs

These three views define three possible points where the process of UI construction can be initiated, internal, external and conceptual, see Figure 2-1.

2.1.1 Exploratory Approach

In the *exploratory approach*, a developer firstly provides an external representation of the UI (e.g., with a graphical editor like those found in Integrated Development Environments like Visual Basic, or Visual C++, or a mock-up produced by a drawing tool such as Microsoft Visio) [Limb04c], the result is expected that will be analyzed the end-user.

The mock-up is traditionally used, consists of either a hand-drawing or a mock-up constructing of the UI, software tools for this purpose are: Corel Draw, Microsoft PowerPoint, Visio.

An easy way to show a UI is using *Visual Programming,* which is typically based on UI toolkits. Visual programming is the most popular way to construct a UI because is easy and ready to be discussed with end-users.

2.1.2 Programmatic Approach

Interface development practices have significantly evolved with programming languages development. In the *programmatic approach* the internal representation is obtained by directly coding the UI in its target computer language, e.g., HTML for a markup language or Basic, Pascal for imperative languages and Java as object-oriented language [Limb04c].

Traditionally two levels are found in the literature for programming languages, *Low level programming* that consists in providing instructions in machine or assembly language, requires a high computing knowledge, advanced programming skills and is time consuming; and *High-level programming* that develops a UI faster than low level programming [Limb04c].

With the improvement of high-level programming, *Toolkits,* which are UI program libraries, contain common widgets used to build the interface. They also provide support functions for manipulating widget like events and I/O handling. The main advantage of toolkits is that they provide a great flexibility and an improved control over the UI elements while maintaining a relative ease of use [Limb04c].

Finally, *Mark-up languages* [Luyt04] are at the edge of programming approach and specification-based approach; see more detail in next section. Mark-up languages are declarative languages. They describe what a UI is rather than what to do to produce it and are generally complemented with scripting languages [Limb04c].

2.1.3 Specification-Based Approach

In software engineering, specification-based (or model-driven) approach relies in the power of models to construct and reason about software systems. This development approach starts with an abstract representation of a UI (i.e., any UI model) [Limb04c]. In this approach models are the basic element.

A model is a simplified and intentional view of real-world things. Real world concepts can be abstracted away in different ways. In other words, modeling is not a deterministic process resulting from observation of the real world [Limb04c].

The goal of specification-based or *model-based approach*, for user interface development is to propose a set of abstractions, development processes and tools enabling a engineering approach of user interface development. The characteristics of an engineering approach are its systematic (development based of rational principles), its reproducibility, its orientation towards quality criteria [Limb04c].

The components required to adopt this approach are the following:

- **Abstractions.** [Limb04c] found three abstractions related to the UI development: *Computing-independent abstractions* that encompass task models and domain models; *UI focused abstractions* that are gathered in two models: a presentation model and a dialog model; finally, the *Context of use abstractions* that are abstractions corresponding to the context model or a user model.
- **Task Model.** A task model is often defined as a description of an interactive task to be performed by the user of an application through the application's user interface. They are also used to achieve a range of objectives [Boms98, Boms99a]: to inform designers about potential usability problems, as in HTA [Anne67]; to evaluate human performance, as in GOMS [Card83]; to support design by providing a detailed task model describing task hierarchy, objects used, and knowledge structures exploited while interacting, as in TKS [John92] or CTT [Pate00].

- **Domain Model.** Domain modeling comes from software engineering [Dsou99]. It represents an essential ingredient to UI engineering methods as it describes its informational content. The domain model is usually developed by software engineers e.g., *Entity-Relationship-Attribute model* (ERA), which seeks to represent real-world objects as *entities* equipped with *attributes*. A second example is a *class diagram* that is an extension of ERA model in the context of Object-Oriented (OO).
- **User Interface Model**. The real-world objects abstracted away in this case concern all manifestations of a UI in the real world i.e., UI appearance (i.e., presentation model) and behavior (i.e., dialog model) [Limb04c]. Three *levels of abstraction*, and corresponding model, are recurrently mentioned in the literature: abstract UI model, concrete UI model and final UI (also called implementation or code level) [Limb04c]. A *final UI*, is composed of two sub-levels. The *rendering level* and the *code level*. This implementation is realized using a programming language. This implementation could be *toolkit dependent* or *independent specification* [Limb04c].
- **Presentation model.** A presentation model is a description of the appearance of a user interface [Limb04c]. Most presentation models found in the literature concern graphical, 2-D, widget-based UIs that is to say WIMP interfaces (Windows, Icons, Menus and Pointing device). *Type of elements* in the scope of the model. *Layout mechanism* exploited. They bring the advantage of a precise interpretation as they are totally unambiguous while omitting any reference to absolute coordinates i.e., the layout declaration stays logical.
- **Dialog model.** Dialog models enable to reason about the behavior of a UI system. Backus-Naur Form (BNF) *grammars* are typically used to specify command languages. *State Transition Diagrams* like statecharts provide a mean for specifying the dynamic behavior of the interface. *Statecharts*, similar to state transition diagrams, support a graphical representation of dynamic aspects of systems. *Petri Nets* is a graphical formalism associated with a formal notation. Petri nets are best suited to represent concurrency aspects in software systems. *Event-Response Languages* treat input stream as a set of events. Events are addressed to event handlers. Each handler responds to a specific type of event when activated.

2.2 Current approaches in 3D user interfaces development

Different categories of software exist to support the rendering of 3D UIs ranging from the physical level to the logical level. At the lowest level are located APIs such as OpenGL, Direct3D, Glide, and QuickDraw3D, which provide the primitives for producing 3D objects and behaviors. They offer a set of powerful primitives for creating, manipulating 3D objects, but these primitives are located at a level that does not allow any straightforward use for rendering higher level widgets. Several 3D desktop replacements for Microsoft Windows XP exist taking the known concept of three-dimensional desktops to its own level.

We will review the low level of developing 3D UIS which is the programmatic one in section 2.2.1; then we review the toolkit approach in section 2.2.2, followed by the render engines which support many of the results generated by the programming languages or the toolkits in section 2.2.3, ending with the Specification based approach which involves methodological alternatives for developing 3D UIs in section 2.2.4.

2.2.1 Programmatic approach

Some hints could be taken from the game industry that is leading the 3d industry [Shne03]. Most games will be written in C++ although, some may use C to try to get even more speed (at the cost of not having built in Object Oriented support) [Beke06a]. Therefore languages such as Java, C#, Basic, and Managed C++ are not used for mainstream games because they tend to run slower. This may possibly change in the future and factors like development time and the ability manage complexity may become more important. However, at the moment, there is not really a viable alternative to C++ for writing high speed games.

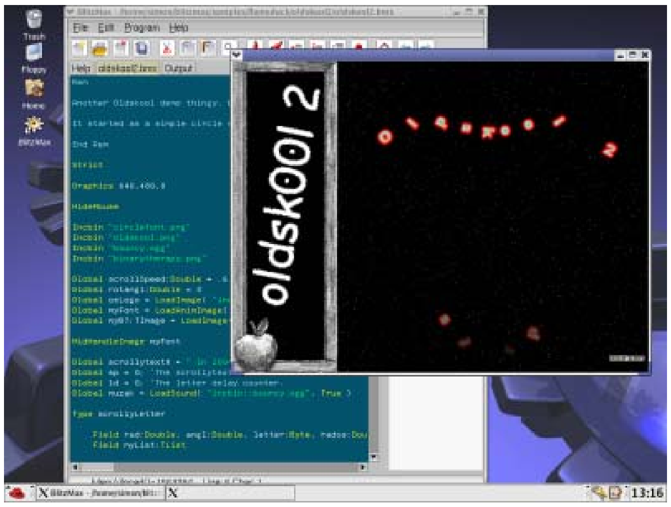
2.2.1.a OpenGL

On the other hand OpenGL [Open04], the premier environment for developing portable, interactive 2D and 3D graphics applications, has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API). Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

2.2.1.b Blitz

Blitz Research Ltd [Blit06a] is a software development company dedicated to develop game creation tools and utilities. Three programming tools based on visual basic programming is their offer:

- BlitzPlus for simpler 2D game and application programming.
- Blitz3D provides a simple yet powerful environment for game creation - simple, because it is based around the popular and easy to use BASIC programming language; and powerful, thanks to a highly optimized underlying 2D/3D engine [Blit06b].
- BlitzMax, a cross platform programming language based on BASIC, but with many weird and wonderful additions [Blit06c]. In this tool Linux and MacOS are supported apart from Windows system. Even that keeps BASIC programming; there is also the possibility to program directly in OpenGL.



```
Rem

Another Oldskool demo thingy. [illegible]

It started as a simple circle [illegible]

End Rem

Strict

Graphics 640,480,0

HideMouse

Incbin "circlefont.png"
Incbin "oldskool.png"
Incbin "bouncy.ogg"
Incbin "binarytherapy.png"

Global scrollSpeed:Double = .[illegible]
Global rotangl:Double = 0
Global osLogo = LoadImage( "in[illegible]
Global myFont = LoadAnimImage([illegible]
Global myBT:TImage = LoadImage[illegible]

MidHandleImage myFont

Global scrollytext$ = " In [illegible]
Global sp = 0; 'The scrollytext[illegible]
Global ld = 0; 'The letter delay counter
Global muzak = LoadSound( "incbin::bouncy.ogg", True )

Type scrollyLetter

    Field rad:Double, angl:Double, letter:Byte, radss:Dou[illegible]
    Field myList:TList
```

Figure 2-2 BlitzMax Software development editor

In Figure 2-2, we could see the Blitz editor which is similar in all their tools. In this case it corresponds to BlitzMax. In Table 2-1 mjbWorld Supporting typesTable 2-1 are summarized their main characteristics.

BlitzMax	Characteristics
openGL	Yes
Basic	Yes

DirectX	Yes
Modularity	Yes
Reuse of Components	Yes
MacOS	Yes
Windows	Yes
Linux	Yes

Table 2-1 mjbWorld Supporting types

2.2.1.c Multi Content Natural Language

Multi Content Natural Language (MCNL), known as Alambik Script [Alam02a], was developed to provide an indispensable resource for producers of digital interactive content working on a wide variety of electronic media. MCNL combines a complete set of text-based programming instructions with a wide collection of text-based audiovisual commands (video, audio, 3D, 2D, etc.). These two worlds, once distinct, are united synergistically under Alambik within a simple, uniform programming environment that functions in real-time [Alam02b].

Consider its basic structure: every Alambik instruction starts with the name of an object and ends with an action to be performed on that object. The object is always a noun, the action always a verb [Alam02b]. For instance, to convert a numerical value into text:

```
String = Text.convert (Number)
Or the traditionally used in programming languages
String = Str (Number)
```

Sometimes a third word (adjective, noun, or verb,) can be inserted between the object and its action, and specifies a property for the object, keeping this structure:

```
Variable = Object.Property.Action (Parameters)
           picture.position.set (@ObjectID,0,0)
```

This basic structure aside, all Alambik instructions end in parentheses () in order to provide space for the passing of parameters. Multiple parameters for a single instruction must be separated by commas [Alam02b].

2.2.1.d VRML

The Virtual Reality Modeling Language (VRML) is neither virtual reality nor a modeling language, [Care97]. On the one hand, virtual reality (VR) is assumed that at some level refers to immersive environment, VRML neither requires nor precludes immersion, is just that is not something mandatory, so this is way can be consider the language as a 3D representation language. On the other hand a modeling language should have a robust set of geometric modeling features but VRML just provides a bare minimum of geometric modeling features.

Even that its name maybe does not correspond exactly with its meaning, the purpose of the language have been achieve. Accordingly to [W3C95] is the intention of its designers to develop VRML as the standard language for interactive simulation within the World Wide Web.

So, as a W3C recommendation several efforts has been done around the language such as plug-ins to the most used and commercial modeling engines such as: Maya, Max 3D, CAD, among many others. Also some plug-ins has been developed for internet browsers which let developers to have to create Web content for the Internet but also for standalone applications.

VRML is not a programming library for application developers. Since VRML is based on the Open Inventor [Sili03] file format. VRML is an extended subset of Open Inventor's file format and does not define an application programmer interface (API), [Care97]. Even, that VRML includes scripting language, this is for authors who need more power or integration.

We show an example of a button, Figure 2-3, in VRML. This button then can be reused in a Menu, Figure 2-4. Both examples are rendered using the Cortona player of parallel graphics, available at http://www.parallelgraphics.com/products/cortona/.

This first section of the code corresponds to the button shape.

```
DEF Menu Transform {
    children [
            DEF Menu1 Transform {
                children [
                        DEF switchMenu Switch { whichChoice 0
                              choice[
                                    Shape {
                                          appearance Appearance {
                                                material Material {
                                                      ambientIntensity 0.15
                                                      shininess 0.20
                                                      diffuseColor 0.20 0.20 0.20
                                                      emissiveColor 0.2 0.2 0.2
                                                      specularColor 0.2 0.2 0.2
                                                      transparency 0.2
                                                }#End Material
                                          }
                                            geometry Box {size 2 1 0.1}
                                          }
                                     Shape {
                                          appearance Appearance {
                                                material Material {
                                                      ambientIntensity 0.15
                                                      shininess 0.20
                                                      diffuseColor 0.5 0.5 0.5
                                                      emissiveColor 0.2 0.2 0.2
                                                      specularColor 0.2 0.2 0.2
                                                      transparency 0.2
                                                      }#End Material
                                                }
                                          geometry Box {size 2 1 0.1}
                                          }#End Shape Square Menu1
                                   ]#end choice
                        }#end Switch
              ]#End Children Menu1
            } #Menu1
```

We used a switch grouping node for the button, with the switch different shapes can be defined and on run time select the one desired. In this case we can use this mechanism to easily provide the user a feedback when the pointer is over the button, i.e. change the color, similarly as buttons react on 2D UI. This can be done with a sensor on the shape and making a called to the function isOver, which sends an event each time that the mouse pointer is over the shape and when is not. So the method defined below shows how to do a script in VRML for such definition.

```
DEF PassOn TouchSensor { }
                DEF script Script {
                        eventIn SFBool isOver
                   field SFBool enabled FALSE
                   eventOut SFBool onOff_changed
                   eventOut SFInt32 which_changed
                   url "javascript:
                            function initialize() {
                              // Initialize to off state.
                              onOff_changed = false;
                            }
                            function isOver( value ) {
                              enabled = !enabled;
                              onOff_changed = enabled;
                              which_changed = enabled;
                            }"

ROUTE PassOn.isOver TO script.isOver
ROUTE script.which_changed TO switchMenu.whichChoice
```

Routing the sensor, PassOn event isOver to the script of the shape method will modify the variable which_changed, which has been defined as a output event, so each time this variable is modified it will launch an event, that is connected to the switchMenu, which is our button, so each time that which_changed change the switch choice will change.
Finally The code corresponding to the text is divided in two parts, the first, shown above for the word "Virtual" and the second for the word "3D GUI", in VRML as in any 3D toolkit, text is one of the most complicated to handle and to situate in the virtual space. Apart from the string there is also a need to translate the text, for the second string the translation values are -0.8 -0.35 0.055.

```
#text definition for the menu Agmented Reality
DEF Text1 Transform {
              children [
                   Shape {
                   appearance Appearance {
                            material Material {
```

```
                                        ambientIntensity 0.15
                                        shininess 0.20
                                        diffuseColor 0.0 0.0 0.0
                                        emissiveColor 0.0 0.0 0.0
                                        specularColor 0.0 0.0 0.0
                                        transparency 0
                                        }#End Material
                                }
                        geometry Text {
                                length []
                                maxExtent 0
                                string ["Virtual"]
                                }
                        }
                ]
                translation -0.7 0.1 0.055
                scale 0.5 0.5 0.5
                                        }
```

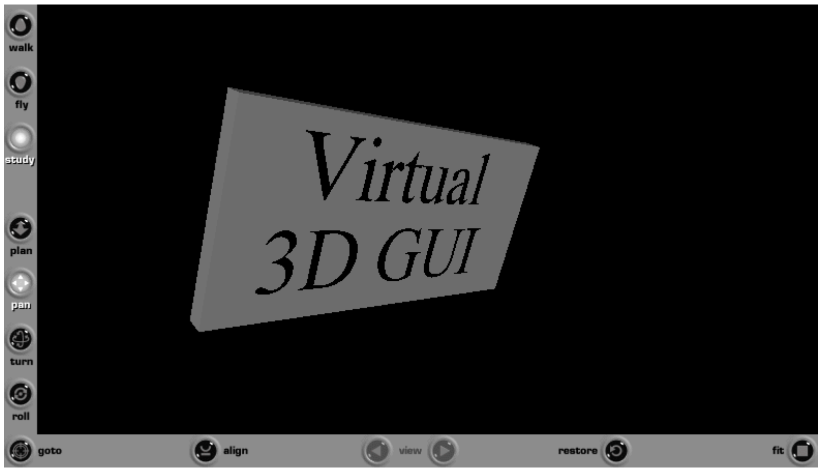


Figure 2-3 VRML Virtual GUI Button rendered in Cortona

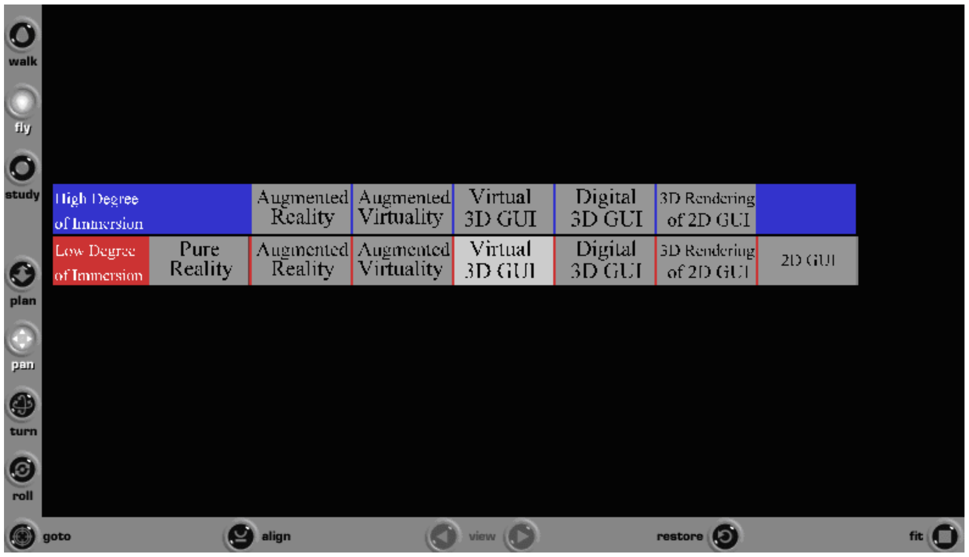


Figure 2-4 Menu using buttons in VRML

2.2.2 Markup Languages

Nowadays, XML-based languages are used in a variety of applications. The description of virtual applications is not the exception. As a XML-based language could represent anything, a wide quantity of solutions has been proposed to 3D development, such as: X3D the VRML evolution. Even that there are more XML-based languages such as: InTML to describe virtual reality interaction techniques or VRIXML to define 3D UI, we prefer to describe them in section 2.2.4, as they more than a XML-based language they represent a methodology to produce 3D UI.

2.2.2.a X3D

X3D [Web304b] is an open standard for 3D content delivery. It is not a programming API. Combines both geometry and runtime behavioral descriptions into a single file that has a number of different file formats available for it, including the Extensible Markup Language (XML). It is the next revision of the VRML97 ISO specification, incorporating the latest advances in commercial graphics hardware features as well as architectural improvements based on years of feedback from the VRML97 development community.

As X3D is one of the recommendations standards there are many modeling tools, such as 3DMax, Maya or a CAD, with VRML97 or X3D exporter available. Many of these are still based on early versions of X3D or write invalid VRML97. To rectify this problem, the Source Working Group is in the process of writing/rewriting exporters for the most popular of the modeling packages [Web304b].

Technologies and effort of the W3D consortium tends to use X3D and not VRML anymore. However there is a lot of work to do. Browsers for X3D, such as XJ3D (http://www.xj3d.org/), Dynamic 3D (http://www.3d-online.com/), OpenVRML (http://www.openvrml.org/) or Carina (http://ariadne.iz.net/~entigo/carina/), all of them are under development. The most advanced is XJ3D, but there is still to much work to do. Similarly as open source efforts, private companies have developed plug-ins for the internet explore browser and Mozilla. Another disadvantage of this language is the lack of input devices support, just the mouse or keyboard are used actually.

The tendency, also, is to handle with more input/output devices. As, the most used languages in the web nowadays is VRML, X3D as its predecessor is expected to take its place. The future could be developing using this language to define 3D applications. X3D is broking down into profiles, including interchange, interactive, immersive and full. The interactive profile provides basic interaction with a 3D environment though various sensor nodes for user navigation and interaction, enhanced timing, and additional lighting [Grac05].

2.2.2.b XVRML

The Extended Virtual Reality Modeling Language (xVRML) Project is focused on evolving VRML into a more modern approach based on using an XML-based notation and an XML Schema -based definition. An example code is shown below related to a red box. We compare to the right how the VRML code would be for the same example.

```
<?xml version="1.0" encoding="UTF-8"?>
<World   xmlns="http://www.xvrml.net/schemas/core"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
         xsi:schemaLocation="http://www.xvrml.net/schemas/core

http://www.xvrml.net/schemas/xVRML.xsd">
    <WorldInfo>
      <title>A Glowing Red Box</title>
      <info>A simple demo file</info>
   </WorldInfo>
    <children>
       <Shape>
          <appearance>
             <Material>
                <emissiveColor red="1.0"/>
             </Material>
          </appearance>
          <geometry>
                <Box/>
          </geometry>
       </Shape>
   </children>
</World>
```

```
#VRML V2.0 utf8


WorldInfo {
        info "A simple demo file"
        title "A Glowing Red Box"
}
Shape {
        appearance Appearance {
                material Material {
                        emissiveColor 1.0 0 0
                                }
                        }
        geometry Box {
                                size 2 2 2
                        }
}
```

XVRML files can be open using the Carina viewer (http://ariadne.iz.net/~entigo/carina/). Carina is an open source viewer for the xVRML format. It includes the Carina application to load files, a library to be used by other applications to load files, and command-line tools convert between formats, it's available for Mac, Windows and Linux. The result of the above red cube Example rendered in Carina in shown in Figure 2-5.

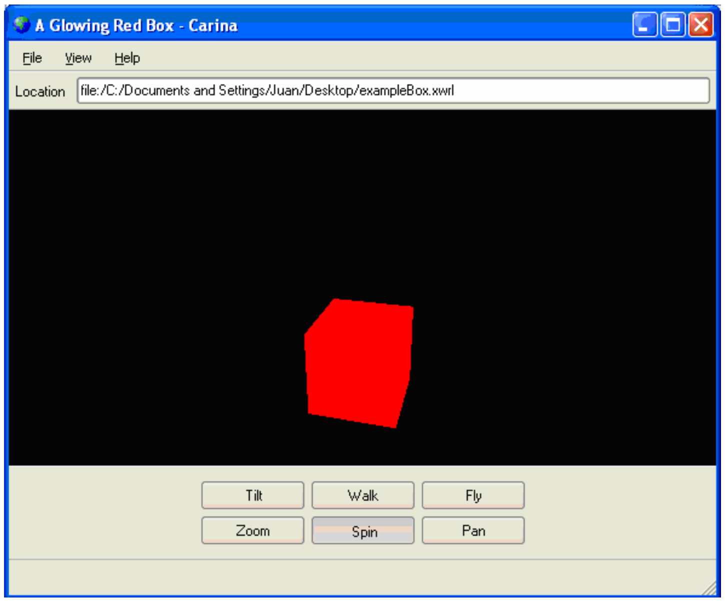

A Glowing Red Box - Carina
File
View
Help
Location
file:/C:/Documents and Settings/Juan/Desktop/exampleBox.xwrl
Tilt
Walk
Fly
Zoom
Spin
Pan

Figure 2-5 Carina Viewer rendering a XVRML file

2.2.2.c VRIXML

Virtual Reality Interaction XML (VRIXML) [Cupp04] is part of the VR-Demo (Virtual Reality: Conceptual descriptions and Models for the Realization of virtual Environments). VRIXML is a User Interface Description Language (UIDL) used to describe the User Interface of virtual application. The framework can be used not just to create the appearance but also to connect the 3D widgets to the functionality of the system.

VRIXML uses the look and functionality 2D widgets, so as the interaction was thought in 2D because the *user* takes advantage of the skills assumed of manipulating 2D UI, and, even that the language support the use of different input devices, such as speech, 3D trackers, spacemouse, they rely in haptic interaction in 2D.

The set of 2D widgets proposed by them has 3 containers, menu, toolbar and a dialog. The dialog can render all the controls that VRIXML provides and the menu and toolbar are just composed of string and buttons respectively, both with the capability of triggering actions, as much they could open a submenu or a sub dialog. Figure 2-5 shows a dialog to control object appearance.

The new syntax proposed by them was conceived, contrary as others, to describe not just the static part of the 3D UI but also the behavior. The main reason of creating a new language is because the others were conceived in a context of work which is not similar, secondly because of the lack of strict parsing (UIML vocabulary can not be validated against its UIML file), [Cupp04].

From [Cupp04] the code description corresponding to the dialog box of Figure 2-6, is described below. Notice that each widget has its own tag and also is composed of several sub tags representing the attributes of the widgets.

Each dialog has some sub-tags to define its title, a texture for the title bar.
Followed by one or more dialog items. This contains a group of items and has a relative position to the dialog.

Grouping elements that are related is good not just for visual purposes but also for code generation and event handling.

When an event is trigger the items that are in a group are analyzed to see if they have the any response to the event trigger in any of the members of the group. This option even that is good for code generation purposes in run time could produce a slow performance, as there is no need to iterate among all the items always each time that one of them is triggered.

```
<UIDialog >
   <Texture >
      <Name> tex_Properties .png </ Name >
         <Color R="1.0" G="1.0" B="1.0"/>
   </ Texture >
   <Title >Object Properties </ Title >
   <DialogItem >
      <UIGroup >
         <GroupItem >
            <UIStatic >
               <Text >Diffuse Color
               </ Text >
            </ UIStatic >
            <Position >
               <X>1.0 </X><Y>0.0 </Y>
            </ Position >
         </ GroupItem >
         <GroupItem >
            <UIStatic >
               <Text >R</ Text >
            </ UIStatic >
            <Position >
               <X>0.0 </X><Y>1.85 </Y>
            </ Position >
         </ GroupItem >
         <GroupItem >
            <UISlider paramID ="10">
               <Value min ="0"
                     max="255"/>
               <Tickstyle
                  orientation =" horizontal "
                  position =" both "
                  frequency ="16"/>
            </ UISlider >
            <Position >
               <X>1.0 </X><Y>1.5 </Y>
            </ Position >
         </ GroupItem >
         ...
         <Event >14 </ Event >
      </ UIGroup >
      <Position >
         <X>0.0 </X><Y>0.0 </Y>
```

```
        </ Position >
    </ DialogItem >
    <Position >
        <X>0.0 </X><Y>0.0 </Y><Z> -20.0 </Z>
    </ Position >
    <Metrics Horizontal =" middle "
            Vertical =" middle "/>
</ UIDialog >
```

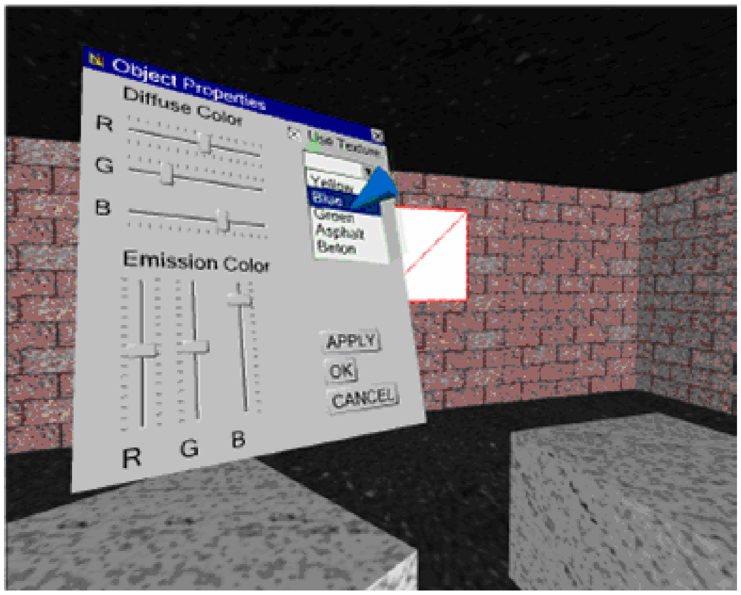


Figure 2-6 VRIXML object property dialog. Source [Cupp04]

VRIXML imitate the 2D UI, as they try to take advance of this well know by computers users representation. They claim that some others UIDL are specific purpose and are difficult to adapt for their goals. Similarly, they fell in the same problem, as they are not proposing a general UIDL as the attributes and the way they handle the events. We classify this solution as a 3D rendering of 2D UI.

2.2.2.d InTML

Interaction Techniques Markup Language (InTML) of [Figu02] is a XML-based language for defining VR content, especially for Interactions Techniques. He consider a VR application a dataflow of interconnected filters, which are the building blocks that describe the standard connections for any of the following entities: input or output devices, interaction techniques, object behavior, animations, geometric objects, and other media objects [Figu02]. InTML is a black box for details about gathering information from devices or about object behavior, as is "described" with code of programming languages. Also, geometry or other media types related to VR objects are produced in any of the available tools for that purpose, such as Maya, 3D Max, or Blender.

"*InTml is then an integration language for all elements involved in VR applications. It enables the designer to concentrate on the architecture of the application, without dealing with too many details"*. [Figu04]

As an example, while dataflow–based languages such as VRML focuses on description of geometry and animation, InTml focuses on the integration of application–specific behavior, object behavior and events from input devices, which is a tedious task in VRML, less complicated actually in X3D, see section 2.2.2.b. Geometry is something that is described at a lower level, in a loadable format, and InTml refers to it as a reference to an object. The same can be applied to sound or haptic content.

2.2.3 Toolkit Programming

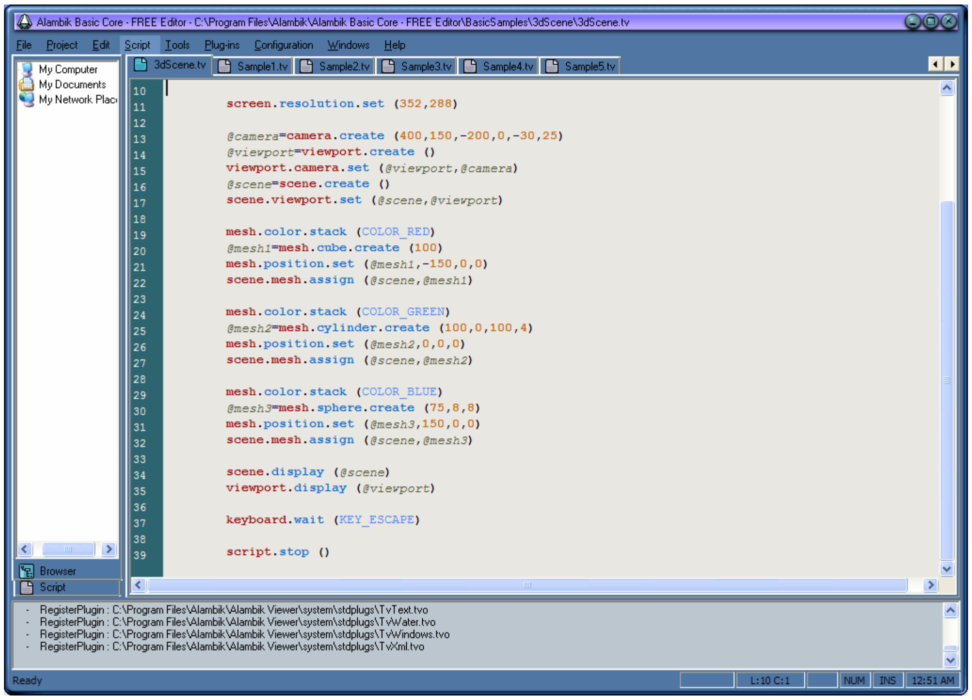
Toolkit programming is one of the most used mechanisms used to develop 2D UI, with interface builders. We will refer to a toolkit as the set of basic building elements for graphical User Interfaces that can be implemented whether in a library, such as Open Inventor, or an application framework such as Alice.

2.2.3.a Open Inventor

Open Inventor is built on top of OpenGL [Open04]. Open Inventor [Sili03] is an object-oriented 3D toolkit offering a comprehensive solution to interactive graphics programming problems. It presents a programming model based on a 3D scene database that dramatically simplifies graphics programming. It includes a rich set of objects such as cubes, polygons, text, materials, cameras, lights, trackballs, handle boxes, 3D viewers, and editors that speed up your programming time and extend your 3D programming capabilities. This toolkit has been used to develop the Virtual Reality Modeling Language (VRML).

2.2.3.b Alambik Software Suite

Alambik technology is simple and coherent, fast, secure, open, modular, and scaled to evolve. It gives you the opportunity to distribute your audiovisual productions to multiple platforms and over different kinds of networks, without the need to rewrite or port your project [Alam02b].


Alambik Basic Core - FREE Editor - C:\Program Files\Alambik\Alambik Basic Core - FREE Editor\BasicSamples\3dScene\3dScene.tv
File Project Edit Script Tools Plug-ins Configuration Windows Help
My Computer
My Documents
My Network Plac
3dScene.tv
Sample1.tv
Sample2.tv
Sample3.tv
Sample4.tv
Sample5.tv
Browser
Script
Ready
L:10 C:1
NUM
INS
12:51 AM

```

        screen.resolution.set (352,288)

        @camera=camera.create (400,150,-200,0,-30,25)
        @viewport=viewport.create ()
        viewport.camera.set (@viewport,@camera)
        @scene=scene.create ()
        scene.viewport.set (@scene,@viewport)

        mesh.color.stack (COLOR_RED)
        @mesh1=mesh.cube.create (100)
        mesh.position.set (@mesh1,-150,0,0)
        scene.mesh.assign (@scene,@mesh1)

        mesh.color.stack (COLOR_GREEN)
        @mesh2=mesh.cylinder.create (100,0,100,4)
        mesh.position.set (@mesh2,0,0,0)
        scene.mesh.assign (@scene,@mesh2)

        mesh.color.stack (COLOR_BLUE)
        @mesh3=mesh.sphere.create (75,8,8)
        mesh.position.set (@mesh3,150,0,0)
        scene.mesh.assign (@scene,@mesh3)

        scene.display (@scene)
        viewport.display (@viewport)

        keyboard.wait (KEY_ESCAPE)

        script.stop ()
```

```
-  RegisterPlugin : C:\Program Files\Alambik\Alambik Viewer\system\stdplugs\TvText.tvo
-  RegisterPlugin : C:\Program Files\Alambik\Alambik Viewer\system\stdplugs\TvWater.tvo
-  RegisterPlugin : C:\Program Files\Alambik\Alambik Viewer\system\stdplugs\TvWindows.tvo
-  RegisterPlugin : C:\Program Files\Alambik\Alambik Viewer\system\stdplugs\TvXml.tvo
```

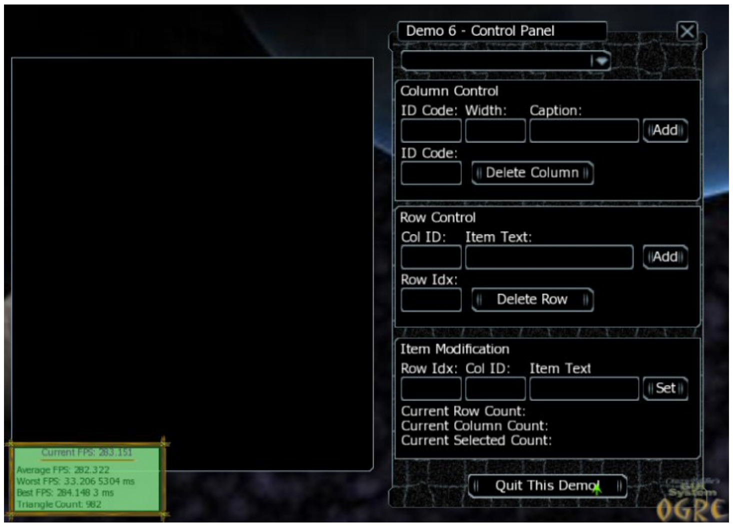
Figure 2-7 Alambik Script Editor

In Figure 2-7 the toolkit ABC from the Alambik Script Editors' family. Work to produce and publishes on the Internet: 2D, 3D, Vectorial, audio, and keyframed animation. Its core is the MNCL syntax [Alam02a], enriched with instructions to publish on Internet. To view the results on the web it is necessary a plug-in for browsers or a viewer for desktop applications.

2.2.3.c Crazy Eddie's GUI System

Crazy Eddie's GUI (CEGUI) System is a free library providing windowing and widgets for graphics APIs / engines where such functionality is not natively available, or severely lacking. The library is object orientated, written in C++, and targeted at games developers who should be spending their time creating great games, not building GUI sub-systems [Craz06].

The GUI toolkit is composed by: button, checkbox, combobox, editBox, Framewindow, listHeader, ListHeaderSegment, multiColumList, MultiLineEditbox, ProgressBar, PushButton, radioButton, ScrollablePane, Scrollbar, ScrolledContainer, Slider, StaticImage, StaticText, ListBox, Thumb, TitleBar, tooltip and window [Craz05]. An example of a game UI generated with Crazy Eddie's toolkit is shown in Figure 2-8. Notice that the UI is presented in 2D instead of 3D.



```
Current FPS: 283.151
Average FPS: 282.322
Worst FPS: 33.206 5304 ms
Best FPS: 284.148 3 ms
Triangle Count: 982
```

Figure 2-8 Game user interface created with Crazy Eddies' toolkit

2.2.3.d mjbWorld

mjbWorld [Bake06a] is an open source 3D toolkit which aim is to build a open and free program to build 3D worlds. Alongside this, it would also be good to work with others to agree free and open standards which would allow similar programs to interchange data. [Bake06a] I would like the program to be able to: Interactively and graphically build 3D models, Display them, Store them and read them in standard formats (VRML, X3D, Java3D), Simulate Newtonian physics so that objects move realistically and collide and interact properly.

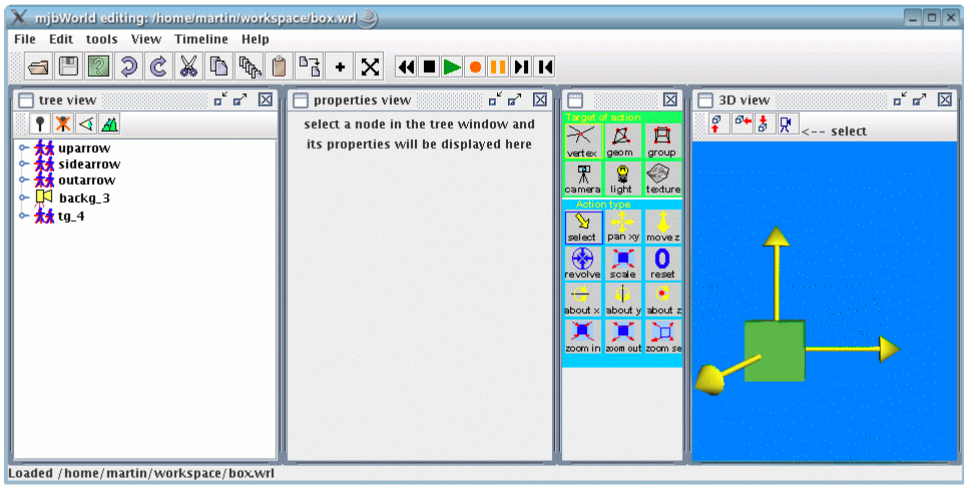


Figure 2-9 mjbworld Environment

With this toolkit three of the most used languages for web 3D programming, i.e. VRML, X3D and Java 3D are used. It allows to model dynamically the content of the worlds and then to export/import them to the standard languages. Summarized in the Table 2-2:

file type	extension	import	export
X3D	.x3d or .xml	Yes	Yes
VRML 2	.wrl	Yes	Yes
Wavefront	.obj	Yes	Yes
Java3d Source	.java	No	Yes
C Source	.c	No	Yes

Table 2-2 mjbWorld characteristics

2.2.3.e Autodesk Family®

Autodesk® family of software tools for 3D development includes several toolkits for 3d development. This family of software is powerful for designing: Manufacturing solutions, Infrastructure (Civil Engineering & Construction, Electric Utilities, Emergency Response, Mapping & GIS, Public Works, and

Transportation) and Building [Auto06a]. The advantage of using this software is that with the plug-ins they provide, it is possible to import / export files from most commercial web 3D languages, such as Java3D, X3D, VRML.

In Autodesk family of software, among the variety of tools, Autodesk inventor® helps designers to pass 2D designs to 3D models, this software is very powerful and provides the possibility to create dynamic animations. The software is more dedicated to expand AutoCAD® users transitioning to 3D [Auto06a]. The two more oriented tools to design Web 3D are: Maya ® and 3D Studio Max ®, both are considered the world's most powerfully integrated 3D modeling, animation, effects and rendering solutions [Auto06b].

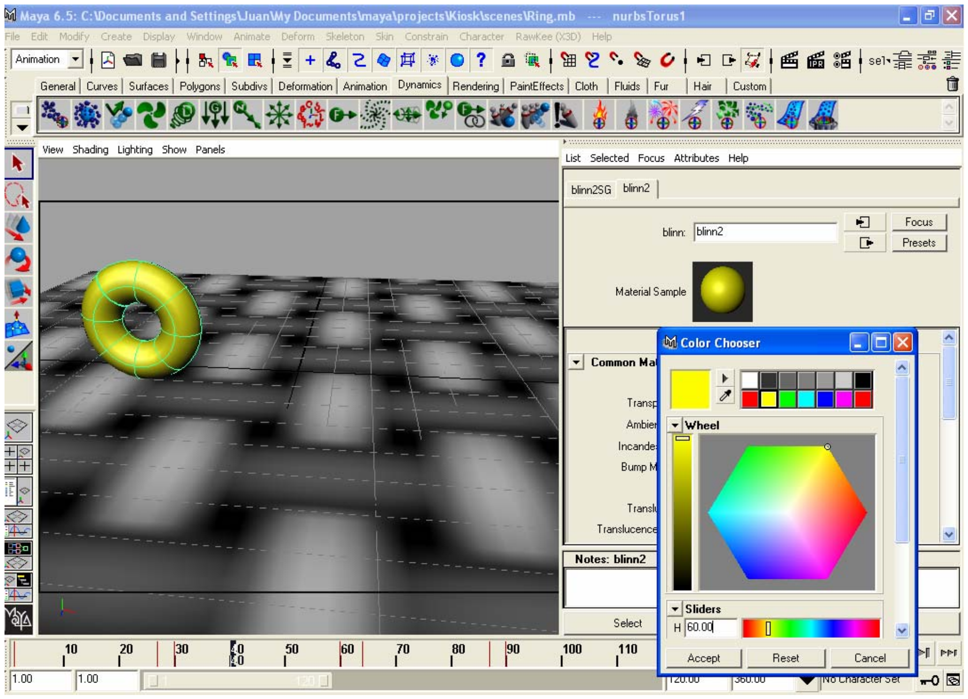

Maya 6.5: C:\Documents and Settings\Juan\My Documents\maya\projects\Kiosk\scenes\Ring.mb --- nurbsTorus1
File Edit Modify Create Display Window Animate Deform Skeleton Skin Constrain Character RawKee (X3D) Help
Animation
General Curves Surfaces Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects Cloth Fluids Fur Hair Custom
View Shading Lighting Show Panels
List Selected Focus Attributes Help
blinn2SG blinn2
blinn: blinn2
Focus
Presets
Material Sample
Common Ma
Notes: blinn2
Select
Color Chooser
Wheel
Sliders
H 60.00
Accept Reset Cancel

Figure 2-10 Maya 6.5 environment

Maya, see Figure 2-10, originally created by Alias, recently merged to Autodesk, Figure 2-11, typically has been used widely to create movies, while Max to design games. Both have a lot of open source work around them, in plug-ins that make these software compatibles with many formats, Table 2-3 summarize their capabilities for open source formats.

		Max		Maya	
file type	**Extension**	**Import**	**Export**	**Import**	**Export**
x3d	.x3d or .xml	Yes	Yes	Yes	Yes
VRML 2	.wrl	Yes	Yes	Yes	Yes
Wavefront	.obj			Yes	Yes
Java3d Source	.java	Yes	Yes	No	No
C Source	.c	Yes	Yes	No	No
Alambik	.snc	Yes	Yes	No	No
Blitz3D	.b3d	No	Yes	No	No
Anark		Yes	Yes	No	No

Table 2-3 Max & Maya Supporting types

In [Bake06b] there is a complete list of the plug-ins that Max provides, not just for open source software but also for commercial ones, such as Alambik, Anark, etc. The disadvantage of using this software is its cost which could be a restriction for some users. Secondly, it seems that in the near future Maya will no longer exist, as Autodesk could merge it in Max 3D.

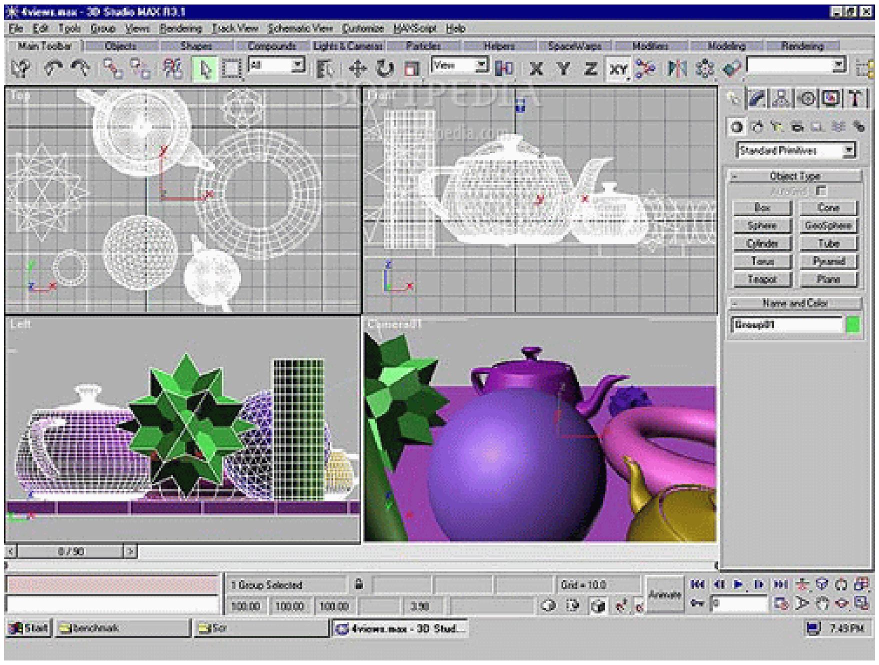


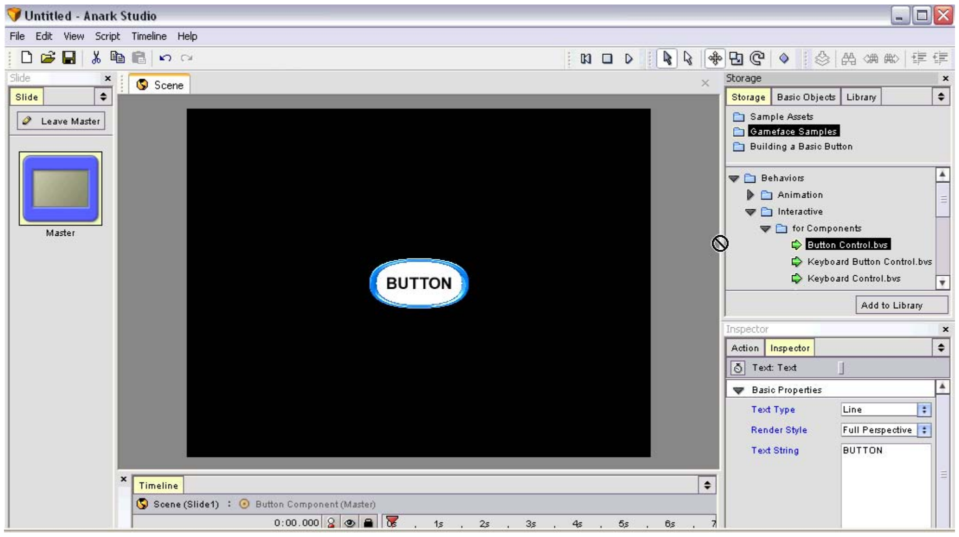
Figure 2-11 Max 3D environment

2.2.3.f Anark Studio

Anark Studio is the application authoring tool for creating interactive 3D content. The product is oriented to design, training and marketing when interactive 3D is needed.

In [Anar06a] the key features of this software are summarized, among others, the key aspects that are interested are: it run 3D presentations on over 90% of existing computers, author drag-and-drop custom widgets such as menus, buttons, and more, Text Object provides crisp, professional and full-featured text that integrates seamlessly with 3D scenes, even text display text data from external sources like XML.

The 3DUI behavior it is possible to add to the animations. The system has a set of actions and events that allows to program interactivity without scripting, as is necessary in VRML or X3D, see in Figue 2-12 a predefined behavior is selected to be added to the button. Scripting with any script language is possible in Anark studio, theoretically, as any script language can be configured to be used. However, just Lua (www.lua.org) scripting is used.



Text: Text
Text Type: Line
Render Style: Full Perspective
Text String: BUTTON

Figue 2-12 Anark Studio adding behavior to a Button

A dynamic UI - by its very nature - cannot be fully and explicitly designed by the artist, programming skills should be added [Anar06b].

The XML support is not just on Text components but also it is possible to load arbitrary XML data from an external file and then reference that data using the Anark scripting engine. The content could be integrated into Web-based systems; also, it provides the possibility to dynamically generate complex Anark Web sites.

Finally, reusability is possible from existing tools, with the import options for 3D models, images, sounds, video and more, so as any existing data in Anark presentations. Integrate Anark content into your existing authoring tools, such as 3D Studio Max.

		Anark	
file type	**Extension**	**Import**	**Export**
3ds Max	.3ds	Yes	Yes
Maya	.mb	No	Yes
Lightwave	.LW	No	Yes
Cinema 4D	.C4D	No	Yes

Table 2-4 Anark Supporting types

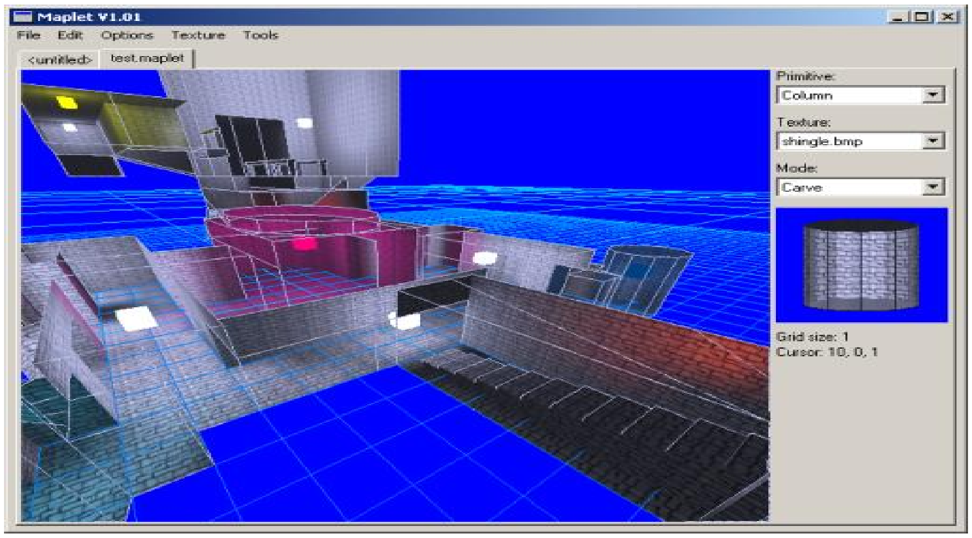
2.2.3.g Maplet

Maplet is a 'Constructive Solid Geometry' based modeler, this means, that instead of modeling with polygons, vertices and so on, all modeling is performed with solid 'primitives' [Blit06d]. Using operations such as 'carve' and 'fill', solid primitives are combined together to create models.

Maplet features a WYSIWYG (what you see is what you get) editor that guarantees the production of geometrically 'correct' models which is very important for precise collision detection and optimal rendering [Blit06d]. InFigure 2-13, Maplet environment is shown. All editing is performed in full 3D, using a 'reference plane' which can easily be moved up or down.

file type	extension	import	export
	.B3D	No	Yes
	.x	No	Yes

Table 2-5 Maplet Supporting types



Primitive: Column
Texture: shingle.bmp
Mode: Carve
Grid size: 1
Cursor: 10, 0, 1

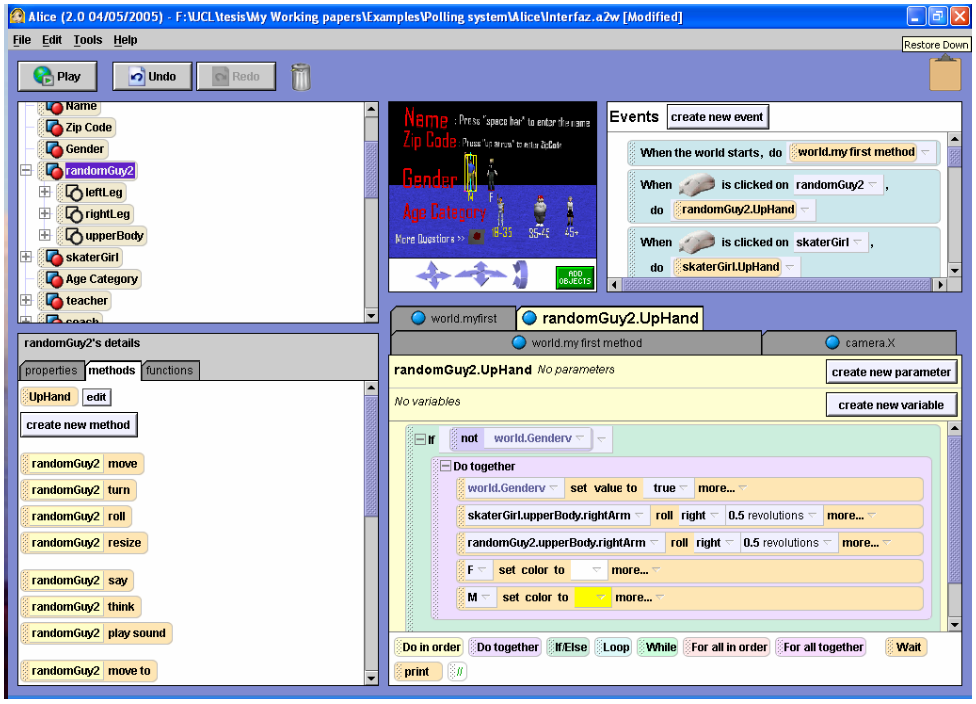
Figure 2-13 Maplet Environment

2.2.3.h Alice

Alice [Carn06] is an open source toolkit for defining 3D content based on predefined sets of objects. Is a funny way to learn about object oriented programming so as to handle with VR. Alice is based in Java 3D, and has and exported to generate content in Java 3D, which can be seeing in a browser with the java 3D virtual machine installed.

In Figure 2-14 the Alice environment, which is divided in four main sections. The scene tree in the top left, in which all the objects attached to the virtual world. Behind, the left bottom, three tabs to display the properties, methods and functions related to the object selected in the world tree. In the top center the preview of the world. On the top right the event handler editor and on the bottom right the method and function editor. Both editors work with a drag and drop of properties and a predefined set of events (such as onclick, when world starts) and of control sentences (such as loops, conditionals).

All the predefined set of behaviors that provide Alice let the developers design the content and animates it through easy drag and drop operations. However when a specific behavior is desired and was not considered in Alice it is possible to define it in their editor. For instance, the method editor (right bottom) in Figure 2-14 shows the code corresponding to select the gender, whether Masculine M or Feminine F. The method shown here is for the click on the Male option, as the object man is selected in the world tree. Using a state variable, called world.Genderv, we ask if is not already picked, as the event will be trigger each time that the mouse click on the man character. Then, using a predefined method called all together that is useful for animations that you want to occur at the same time, the five next lines corresponds to assign the selected value to the state variable world.Genderv, the following second and third lines corresponds to rise the hand for the man and down the hand for the woman, finishing with the modification of the text objects F and M, changing their color to white and yellow respectively.



```
When the world starts, do   world.my first method
When [mouse] is clicked on randomGuy2 ,
  do   randomGuy2.UpHand
When [mouse] is clicked on skaterGirl ,
  do   skaterGirl.UpHand
```

```
randomGuy2.UpHand  No parameters
No variables
If  not  world.Genderv
  Do together
    world.Genderv  set value to  true  more...
    skaterGirl.upperBody.rightArm  roll  right  0.5 revolutions  more...
    randomGuy2.upperBody.rightArm  roll  right  0.5 revolutions  more...
    F  set color to     more...
    M  set color to     more...
```

Figure 2-14 Alice Environment

Alice is an easy to use toolkit very useful for academic purposes, its main goal is to teach how to program object oriented applications. Also, covering VR spectrum apparently it looks that it catch the attention of major industries, as the creators of Sim City, the famous real life simulation game, will donate their characters, which look so realistic to enhance the presentation of Alice. Apparently there will be more news about Alice in the near future.

2.2.4 Rendering engines

Rendering engines are the solution that many of the projects described before in this section provide to render stand-alone 3D applications.

2.2.4.a Alambik® Viewer

Alambik technology is simple and coherent, fast, secure, open, modular, and scaled to evolve. It gives you the opportunity to distribute your audiovisual productions to multiple platforms and over different kinds of networks, without the need to rewrite or port your project [Alam02b]. Alambik® Viewer developed from the porting of Alambik® Script 4.0. Alambik viewer could work under the Windows operating systems and is compatible with Netscape 4+, Internet Explorer 4+, Opera 6 and Mozilla [Alam03]. The Alambik viewer is shown in Figure 2-15.

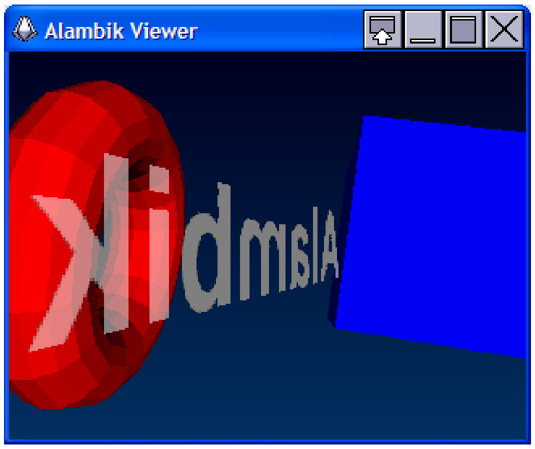


Figure 2-15 Alambik Viewer

2.2.4.b Object-Oriented Graphics Rendering Engine (OGRE)

OGRE (Object-Oriented Graphics Rendering Engine) is a scene-oriented, 3D engine written in C++. The class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes [Ogre05].
OGRE just provides a world-class graphics solution; for other features like sound, networking, AI, collision, physics etc, other libraries are required. Many

experienced game developers have expressed their approval of this approach, because there are no inbuilt constraints, [Ogre05].

The Ogre source is made available under the GNU Lesser General Public License (LGPL). In Figure 2-16, an example of the rendering made with OGRE, notice that the components regarding the User Interface remains in 2D.

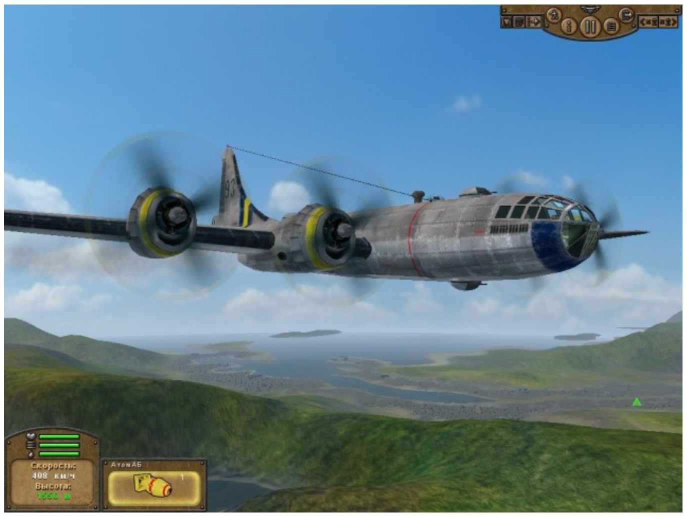


Figure 2-16 OGRE rendering

2.2.5 3D Desktop Systems

Several 3D desktop replacements for Microsoft Windows XP exist such as Microsoft Task Gallery (http://research.microsoft.com/adapt/TaskGallery/), the Infinite3D Cube (http://www.infinite-3d.com/), SphereXP (http://www.ha mar.sk/sphere/) which is taking the known concept of three-dimensional desktops to its own level. Similarly, SUN has initiated the Looking Glass Project (http:// wwws.sun.com/software/looking_glass/index.html) as a 3D desktop environment for Linux workstations. These environments are very powerful for their manipulation of windows in 3D, but they are not intended to render 2D UIs with 3D effects.

2.2.5.a Clara

Clara (http://www.spatialknowledge.com/projects/clara/) is a 3D web browser that lets walk, fly, or jump through a virtual world where all the objects are usable, interactive web-page. In Clara it is possible to read the pages as traditionally interacting in 2D, i.e. scrolling, clicking, or hearing is possible. The screenshot of Figure 2-17 shows the presentation of Clara rendering multiple pages at once. Clara is implemented as a Windows program in versions for 3D boards running OpenGL or DirectX. Clara realy offers not just a fancy new way of interacting when browsing web sites but also is based on the spatial knowledge, the intuitive knowledge of humans about their three-dimensional environment [Wiki06], in others worlds, the picture about the organization of our surrounding. Clara is a work still under construction at the time this thesis is written.

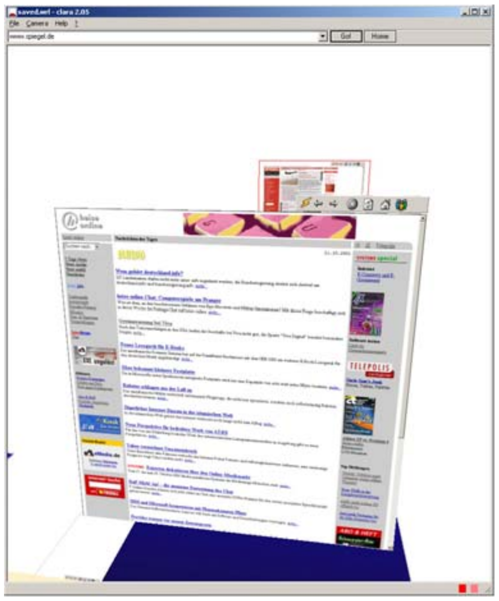


Figure 2-17 Clara Web Browser

2.2.5.b SphereXP

SphereXP (http://www.hamar.sk/sphere/) which is taking the known concept of three-dimensional desktops to its own level. It offers a new way to organize objects on the desktop such as icons and applications. SphereXP, like other similar environments, are usually limited to presenting existing interactive applications and their UIs in a flat 2D way, even if they are working in a 3D world (see Figure 2-18)

The Sphere is in theory a 3D workspace represented by a sphere. The user is exactly in the middle of it. All objects are situated around them. They can easily turn around and manipulate everything. Objects can be moved around the sphere according to some rules. It is possible to bring objects closer to the view port or send them back. The theory can be applied to almost any known program, starting with the desktop interface. Notice that they assume this potential of the sphere. However the truth is that they do not have control on all Windows applications even for the desktop. More than this, another problem that they will find always is that as new windows applications emerge there will always be a necessity to adapt then to use them in the sphere.

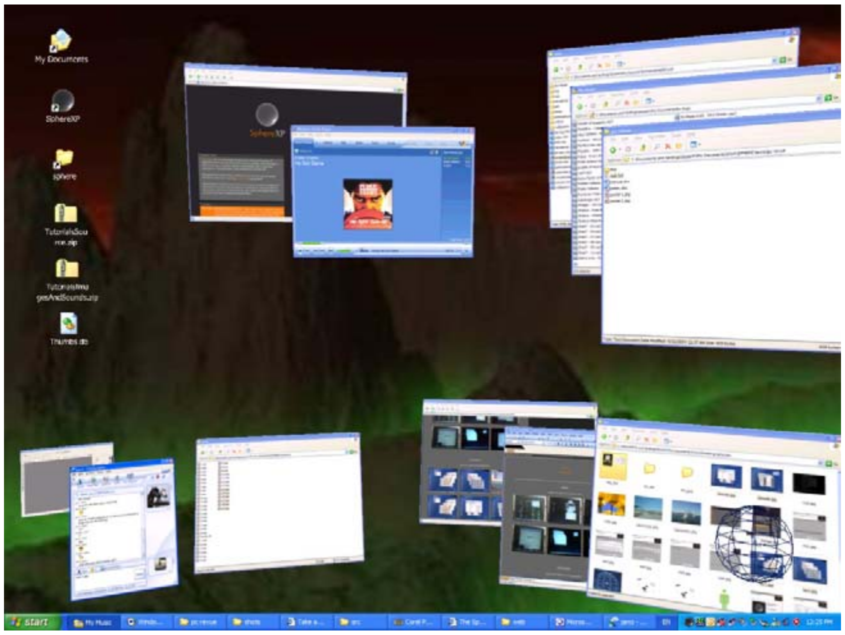


Figure 2-18 SphereXP three dimensional desktop

2.2.5.c 3DNA

The 3DNA Desktop also intends to improve the way we work with Windows and the Web. This effort offer different world views called, add-ons that the user can interchange. 3DNA provide an easier ways to organize files, folders, and applications-direct access, in different objects such as the wall or drawers. Also provides unique capabilities that are not possible in a 2D Windows interface such as the ability to speed-surf dozens of websites at a time. The favorites section has a screenshot of the web site, which we can zoom in/out, this offers the user more means to remember a booked page.

The User experience is enhanced with unlimited customization possibilities, integrated 3D games, and an ever growing collection of great-looking Add-on Worlds. Since the 3DNA Desktop does not replace Windows, the Start Menu, or the Task Bar, users can easily use the new 3D interface immediately. The movement and navigation within a 3DNA Desktop is identical to a 3D video game and is also fully customizable by the user (short cuts can be easily specified to a specific place in the world).

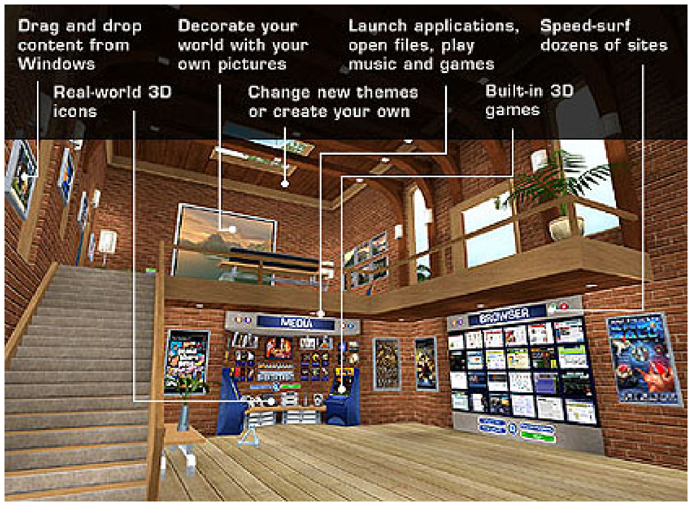
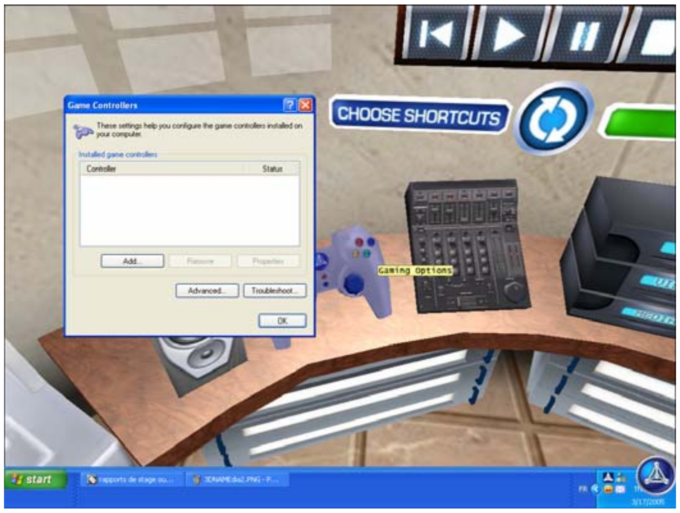

Drag and drop content from Windows
Decorate your world with your own pictures
Launch applications, open files, play music and games
Speed-surf dozens of sites
Real-world 3D icons
Change new themes or create your own
Built-in 3D games
MEDIA
BROWSER

Figure 2-19 3DNA Desktop, source http://www.3dna.net/products/desktop.htm

3DNA just intend to replace the desktop as shown in Figure 2-19, not to change the way windows UI looks. In screenshot of Figure 2-20 we show the 2D dialog shown when clicking on the 3D game control. So there is no possibility to interact directly with the objects in the world.

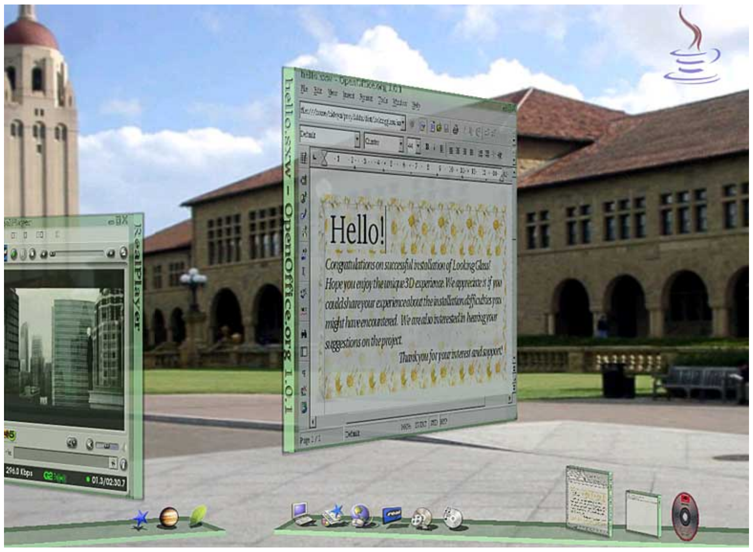

Game Controllers

These settings help you configure the game controllers installed on your computer.

Figure 2-20 3DNA Desktop, source http://www.3dna.net/products/desktop.htm

2.2.5.d Looking Glass

Similarly to the described efforts, SUN has initiated the Looking Glass Project (http://wwws.sun.com/software/looking_glass/index.html) as a 3D desktop environment for Linux workstations. It is an open source development project that supports running unmodified existing applications in a 3D space, as well as APIs for 3D window manager and application development. At the moment, existing application integration is supported for Solaris x86 and Linux platforms. The library for 3D application development is available for Linux, Solaris and Windows.



Congratulations on successful installation of Looking Glass! Hope you enjoy the unique 3D experience. We appreciate it if you could share your experience about the installation difficulties you might have encountered. We are also interested in hearing your suggestions on the project.
Thank you for your interest and support!

Figure 2-21 Looking glass Desktop

In the screenshot of Figure 2-21, rotate the windows to shows information on the back of screens is one of the options provided by the tool making the user experience more intuitive. Similarly the screenshot shows two different desktops rendered in the same screen, depicted with the windows and the task bars.

2.2.6 Model-Based 3D User Interface Approaches

In software engineering, specification-based (or model-driven) approach relies in the power of models to construct and reason about software systems. This approach is based on models. To generate them we need to identify the main properties of real life objects. To do so some kind of judgment is required. In [Limb04c] we identify the goal of model-based approach:

"*The goal of specification-based, or model-based approach, for user interface development is to propose a set of abstractions, development processes and tools enabling a engineering approach of user interface development. The characteristics of an engineering approach are its systematic (development based of rational principles), its reproducibility, its orientation towards quality criteria*"

Analogous to section 2.1.3, our approach considers various levels of abstraction and types of concerns for the development of 3D User interfaces. These abstractions levels are: Task & Concept*s,* Abstract UI, Concrete UI, Final UI, and the Context of use. More detail on the contents and descriptions will be provided in next chapter.

A great effort has been conducted in 2D Model-based user Interface development [Limb04c]. However such kind of effort has not been done for 3D UI. Today, the development life cycle of 3D user interfaces (UIs) mostly remains an art more than a principled-based approach. Several methods [Bowm04, Cele01, Fenc01, Geig01, Neal01] have been introduced to decompose this life cycle into steps and sub-steps, but these methods rarely provide the design knowledge that should be typically used for achieving each step. In addition, the development life cycle is more focusing directly on the programming issues than on the design and analysis phases. This is sometimes reinforced by the fact that available tools for 3D UIs are toolkits, interface builders, rendering engines, etc.

In this section we present some work related to model-based development for 3D UIs. We rely on the models described at the beginning of this chapter to analyze the methodologies, which are: Task, Domain, Abstract, Concrete, and Final User Interface models. A unified iconographic representation is used to present development processes (Figure 2-22). Each symbol represents a type of models. Note that the difference between "interactor-type independent" and "toolkit independent" representation is stressed by two different icons associated with abstract UI. A dialog model is also represented as a separated entity when the dialog model is substantially important in the development process. Solid arrows represent the derivation of one model from one or several other ones. Dashed arrows represent a significant knowledge adjunction in the design process (e.g., a designer manually determines a layout; a template is chosen to drive the derivation).

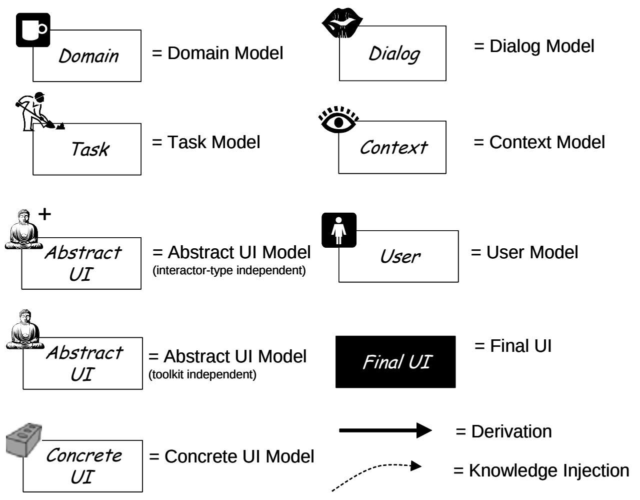


Figure 2-22 Symbols used for the state of the art on CADUI tools

2.2.6.a Web & Information System Engineering Lab

The virtual reality Web & Information System Engineering (VR-Wise) [Pell05b] project looks for conceptual modeling of Virtual environments. Starting from web design methods and design methods for Virtual Reality. WISE Lab has three main project aimed at designing and specifying Virtual (Web) Environments in a systematic way and that can be supported by design methodologies and tools.

They have developed models for virtual objects in what they called OntoWeb tool [Pell04a], part of the *OntoBasis* project. They called their models ontologies and with them they look for Foundations, Construction, Services and Applications. Their approach is base don the idea that as a Virtual Environment is composed of objects it may be possible to extract properties of these objects and their relationships in the Virtual Environment from available ontologies covering the domain under consideration.

From the ontologies they build virtual correspondences mapping the ontologies concepts one-to-one to virtual properties, which is a good first step to design Virtual worlds. Considering our methodological development path we place this approach at the Content description for UI, in which objects/content are described in models that correspond to a specific platform. They generate a direct mapping from the concepts described in the domain model to virtual objects VRML or X3D.

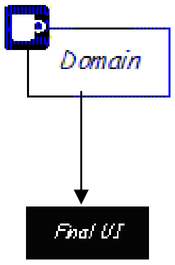


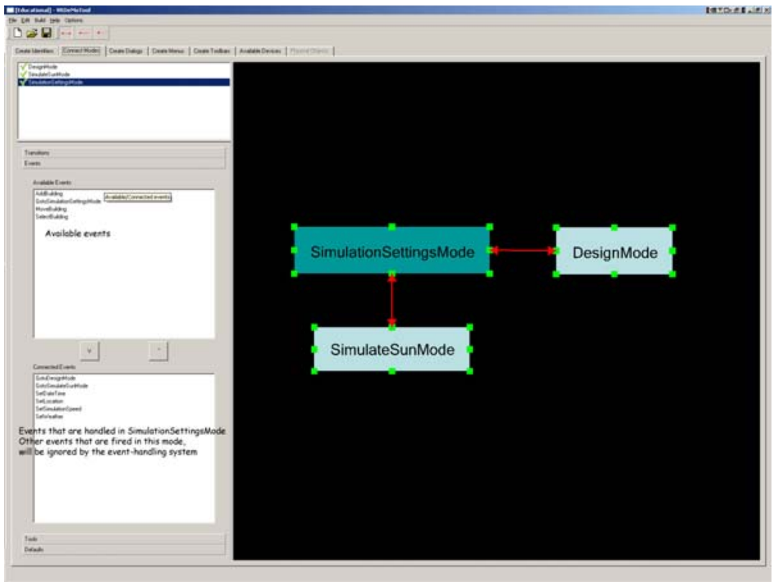
Figure 2-23 OntoBasis development steps

2.2.6.b CoGenIVE

The code generation for interactive virtual environments (CoGenIVE) [Cupp05] method is a way to developed, as its name state, interactive virtual environment following a model based approach. This work is linked with the VRIXML language [Cupp04], already reviewed in this chapter, section 2.2.2.c. The VR-Wise approach and CoGenIVE are linked in the IWT SBO VR-DeMo project

As stated before, VRIXML support several input/output devices. Apart from the devices the framework contains 2D/3D hybrid UI widgets, and example is shown in Figure 2-6.

Apart from this concrete level of description, CoGenIVE considers task and dialog models [Cupp05]. The dialog is visualized as a state chart, so the designer can specify the different states of the user interaction, see Figure 2-24, so as the certain task and events are predefined.



Events that are handled in SimulationSettingsMode
Other events that are fired in this mode,
will be ignored by the event-handling system

Figure 2-24 CoGenIVE StateChart

An interesting feature of this tool is that considers the reverse engineering process and is actually implemented. Any change that is done in the FUI by hand is tracked the way back to its abstract representation and vice versa, any change made in the models respects the changes made manually. In Figure 2-25 the methodology is described. The arrow depict that the process of abstraction is possible when the models

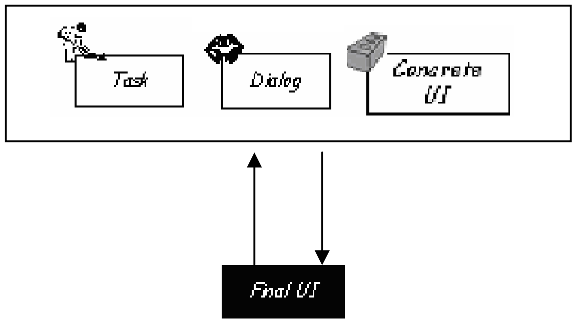


Figure 2-25 CoGenIVE development process

As stated in [Cupp05] task, dialog and objects are specified and then UI code is generated with its functionality. The concrete level, depicts the UI models.

2.2.6.c Interaction Techniques Markup Language

Previously reviewed in this chapter as XML-based language for developing 3D UI, the Interaction Techniques Markup Language (InTML) is more than just a way of representing the UI.

Related to the methodology proposed by [Figu04] that is aimed to design VR applications d in VR applications. It enables the designer to concentrate on the architecture of the application, without dealing with too many details. The ontology proposed is shown Figure 2-26.

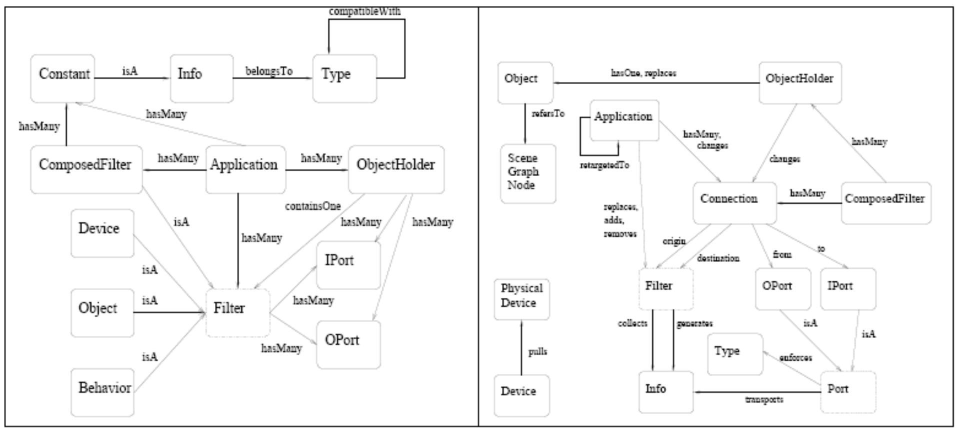


Figure 2-26 Entities and Relationships of InTML. Source [Figu04]

In the center of the left square is the filter entity, the abstract building block of InTML that can be any device, interaction technique, behavior, or content in a VR application. Its interface is defined in terms of input and output ports (IPort and OPort), 28 events are handled or defined for them.

The behavior description correspond to the dialog specification, objects and devices models are part of a concrete model, because it is independent of implementation, at first view, assuming that there is no direct relation between those components and the implementation. The mechanism of InTML start with the specification of the goal of the projects, with the definition of the main tasks, everything is documented in InTML documents, which are refined though the process [Figu04]. However, this task in done manually without following nor a standard, neither with a set of concepts. For this reason task is something that injects knowledge but not derivate, curve arrow in Figure 2-27.

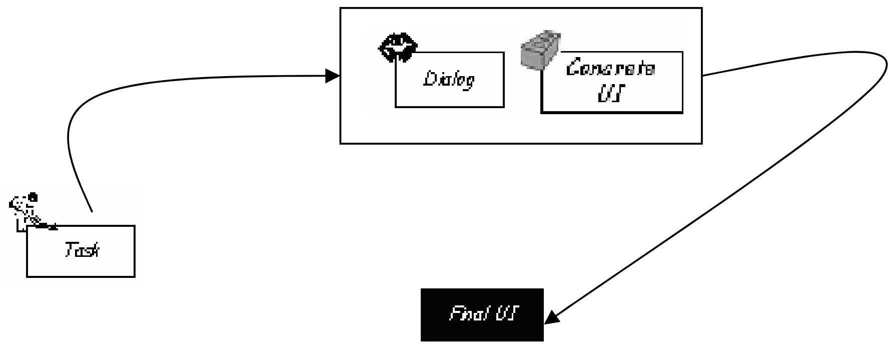


Figure 2-27 InTML. Development Process

Similarly as tasks, the code generation for the FUI is straight forward of InTML methodology.

2.2.6.d Contigra

COmponent-orieNted Three-dimensional Interactive GRaphical Applications (CONTIGRA) project CONTIGRA [Dach02] is a XML application on top of X3D that give users components for interaction techniques and control widgets. The project has models for behavior, Behavior3D [Dach03]. Below an extract of Behavior 3D, that is a XML-based, in fact correlated to X3D format.

```
<TouchSensor DEF="LCD_Sensor"/>
<TouchSensor DEF="Keyboard_Sensor"/>
<StateMachine stateCount="3" transitions="1 2 LCD_Sensor.touchTime OpenLaptop.startTime,
                                          2 1 LCD_Sensor.touchTime CloseLaptop.startTime,
                                          2 3 Keyboard_Sensor.touchTime OpenKeyboard.startTime,
                                          3 2 Keyboard_Sensor.touchTime CloseKeyboard.startTime"/>

<AnimateRotation key="0 1" to="1 0 0 0, 1 0 0 -1.7"  cycleInterval="2" DEF="Openlaptop "/>
<AnimateRotation key="0 1" to="1 0 0 -1.7, 1 0 0 0"  cycleInterval="2" DEF="CloseLaptop"/>
<Sequential DEF="OpenKeyboard">
        <AnimateTranslation key="0 1" to="0 0 0, 0 0.05 0" cycleInterval="1" />
        <AnimateRotation key="0 1" to="1 0 0 0, 1 0 0 -1.5" cycleInterval="1" />
</Sequential>

<Sequential DEF="CloseKeyboard">
        <AnimateRotation key="0 1" to="1 0 0 -1.5, 1 0 0 0" cycleInterval="1" />
        <AnimateTranslation key="0 1" to="0 0.05 0, 0 0 0" cycleInterval="1" />
```

</Sequential>

Two X3D touch sensors trigger the animation of a laptop. A state machine node in Behavior3D defines the possible transitions between states one to three. Sensor connections are established also for the animation, defined for opening and closing the keyboard.

The core concepts in Behavior3D are animations, sequences of actions, and state machines. On top of them, other behaviors can be defined. Behavior3D targets 3D applications running over the web, in standard PCs, due the intrinsic characteristics of X3D and its interaction model, [Figu04].

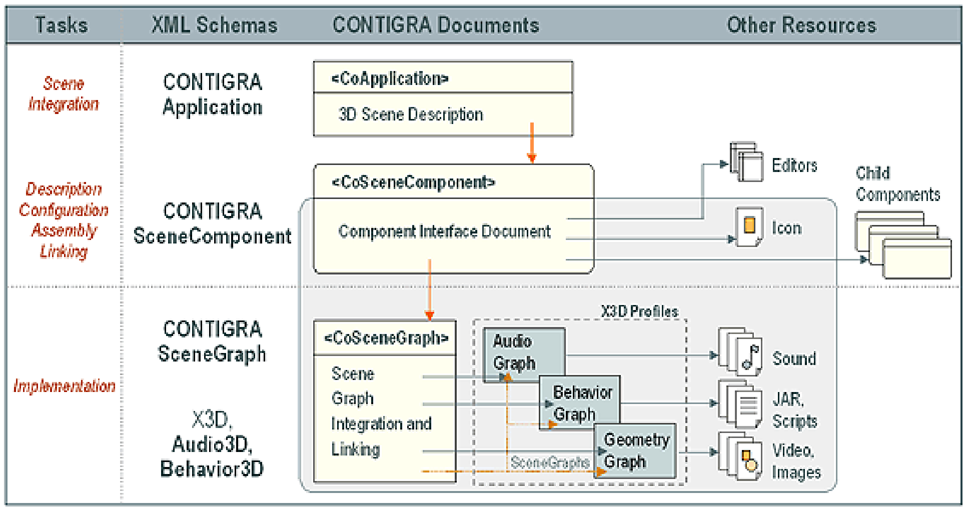


Figure 2-28 Contigra Architecture. Source Contigra Web site

Part of the Contigra project, see Figure 2-28, Audio3D [Hoff03] is a solution of offered to describe 3D applications with audio that allows the description of complex acoustic environments but is still suitable for efficient real time sound rendering with 3D sound APIs. Similar to X3D a hierarchical, acyclic scene graph is used in Audio3D to organize nodes in groups and subgroups.

Contigra concepts are mapped directly to a FUI, as they are described using X3D profiles. So Contigra development process is composed of dialog (Behavior 3D), Concrete UI (Models involved for defining the UI) and Final User Interface is generated automatically.

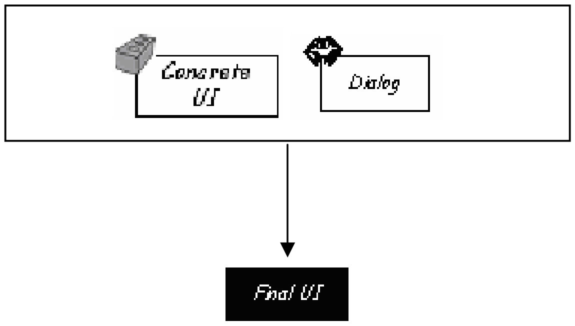


Figure 2-29 Contigra Development Process

2.2.7 Comparison on Developments

The present section gives a synthetic overview of: i) Model-based surveyed in this chapter and ii) for the software tools. For this purpose, two families of properties are segregated: properties regarding conceptual content of methodologies, and properties regarding the model transformations underlying these methodologies.

2.2.7.a Comparison on Model-based Developments

Table 2-6 sum up this comparative analysis of model-based approaches reviewed. The properties analyzed in the comparison are coherent to those used in [Limb04c], which are:

- Models manipulated by the methodology.
- Inter-mode linking. Three object were chosen to depict the different relationships:
    - **(A, …, B)** indicates A, …, B are grouped models that are done at the same level.
    - **A↔B** indicates A derivates B and B reengineered A
    - **A→B** indicates A derivates B
    - **A≈ ↔B** indicates that model A concepts could be manually linked to model concepts B and B can be manually reengineered to A.
    - **A≈→B** indicates that model A concepts could be manually linked to model concepts B. This means that the rules and the models exist but not a tool to support the automatic transformation.
- Target Languages designate the languages of the UI to produce. the CUI model to FUI (code). But the theory and the rules required so as the

corresponding code exists.When Manually

- Availability of models (Avl. Mod.) refers to the possibility for an external tool to process the manipulated models. Possible values:
    - ✗: models are stored in an internal format not made explicit e.g., models are tightly coupled with the tools.
    - √: means that an external format for models exists e.g. models are available under a machine understandable format. A typical form is an XML language.
- Extensibility of models definition (Ext.Mod.): refers to the possibility of extending definitions of models with new elements.
    - Orig.: means that models were intended to be extensible, but only by the originator of the methodology. This guarantees some interoperability of tools around a language.
    - Design: means that models are extensible and the designer (e.g., the tool user) is responsible for this extension.
    - ✗ : means that no mechanism supports model extension e.g., the system is bundled with a particular set of model definitions.

2.2.7.b Comparison on Toolkits

Table 2-7 sum up a comparative analysis of Toolkits reviewed. The properties analyzed in the comparison are:

- Models manipulated by the toolkit, if any.
- Inter-mode linking. Three object were chosen to depict the different relationships:
    - **(A, …, B)** indicates A, …, B are grouped models that are done at the same level.
    - **A↔B** indicates A derivates B and B reengineered A
    - **A→B** indicates A derivates B
    - **A≈ ↔B** indicates that model A concepts could be manually linked to model concepts B and B can be manually reengineered to A.
    - **A≈→B** indicates that model A concepts could be manually linked to model concepts B.
- Target Languages designate the languages of the UI to produce.
- Availability of toolkits (Avl. Mod.) Refers to the possibility for an external tool to process the manipulated the software. Possible values:
    - ✗: not available.
    - √: open source or available at certain level.

	Models t= task, Do = Domain Di = dialog AUI=abstract presentation CUI=concrete user interface U = user, C = context.	**Inter Model Transformation** ↔Bidirectional derivation → Derivation link ≈→ Manual Derivation ≈ ↔Manual Bidirectional der. FUI = Final User Interface	**FUI target languages**	**Avl Mod.**	**Ext. Mod**
VR-Wise	CUI	CUI → FUI	VRML, X3D	√	Orig.
CoGenIVE	T, Di, CUI	(T, Di, CUI) ↔ FUI	C++	√	Orig.
InTML	Di, CUI	T ≈→ (Di, CUI), (Di, CUI) ≈→ FUI		√	Orig.
Contigra	CUI, Di	(CUI, Di) → FUI	X3D, Behavior3D, Audio3D	√	Orig.
Our methodology	T, Do, C, AUI, CUI	T ↔ Do, T ↔ AUI, AUI ↔ CUI, CUI ↔ AUI, T ↔ CUI, (T, Do,AUI,CUI) ↔ C, CUI≈→ FUI	Java, XHTML, Flash, HTML, Voice XML, Java 3D, X3D, VRML.	√	Orig.

Table 2-6 Model-based methodologies comparison

	Models **C = concrete** **A = Abstract**	**Inter Model Transformation**	**Platforms**	**FUI target languages**	**Avl Mod.**
BlitzMax	C	C → FUI	Mac, Windows, Linux	openGL, Basic, DirectX	x
Carina	C	C → FUI	Mac, Windows, Linux	xVRML	√
Alambik	C	C → FUI	Windows	MNCL	x
Crazy Eddie	CUI	CUI → FUI	Windows	C++, OGRE	√
mjbWorld	C	C → FUI	Windows	X3D, VRML, Java 3D, c, Wave Front	√
Maya	C, A	C→ FUI, A→ FUI	Mac, Windows, Linux	X3D, VRML, Wave Front	x
Max 3D	C, A	C→ FUI, A→ FUI	Mac, Windows, Linux	X3D, VRML, Wave Front, C, Alambik, Blitz3D, Anark	x
Anark	C, A, Di	(C, Di)→ FUI (A, Di)→ FUI	Windows	Lua script, anark, Maya, Max 3D, Cinema 4D, Light Wave	x
Maplet	C	C → FUI	Windows	B3d	√
Alice	C, A, Di	(C, Di)→ FUI (A, Di)→ FUI	Mac, Windows	Java 3D, HTML	√
VR-Wise	CUI	CUI → FUI		VRML, X3D	x
CoGenIVE	T, Di, CUI	(T, Di, CUI) ↔ FUI		C++	x
Contigra	CUI, Di	(CUI, Di) → FUI		X3D, Behavior3D, Audio3D	x
Our methodology	T, Do, C, AUI, CUI	T ↔ Do, T ↔ AUI, AUI ↔ CUI, CUI ↔ AUI, T ↔ CUI, (T, Do,AUI,CUI) ↔ C		Java 3D, X3D, VRML.	√

Table 2-7 Toolkits comparison

2.3 Conclusion

We review in this state of the art, briefly the development of 2D UI, passing directly in deep to the 3D UI development approaches. Four 3D UI development approaches were considered in this state of the art:

A *programming approach* allows a straightforward implementation of a final interface. In terms of quality criteria, these approaches vary depending on the degree of portability, the resource consumption (expressed in time units, monetary units, lines of code, etc), and the ease of use (which depends on provided tool support, intuitiveness of the concepts, legibility of the code, etc). The languages reviewed include XML-based, which are extremely linked with a methodology and represent concepts. However, this phase of development alone does not assure the quality of the result. Programming (and maintaining) a 3D UI without any method can be a chaotic activity. It gives no guarantee for regularity. There are no evaluation criteria to consider. "rush to code" without any structure favors a "trial and error" method. The result of such a work will highly depend on contingency factors such as the developer's experience or the development context, [Limb04c]. Finally, the communication channel between developers and the Final users does not exit. So, the programming approach should be taken as it is. Programming an interface is not engineering it [Limb04c].

A *toolkit approach,* similarly, allows a straightforward implementation of a final interface. In this case using predefined set of objects that help developers in their programming task. The toolkits reviewed here in some cases, Contigra, CoGenIVE, correspond not just to a toolkit to help developers in their programming task but also considers the methodological process proposed in the projects. In some cases a toolkit can be part of the engineering process, this depend on the context.

Rendering engines allow 3D programming languages to be visualized. In some of the reviewed tools there is a need for a specific render engine as they do not use a standard programming language. These render engines in some case has their own API to render sophisticated graphics. Render engines linked to a programming languages or a toolkit can produce fancy 3DUI. A render engine is just a required complement but is not part of the engineering process.

A *specification-based* approach provides us with means to specify relevant properties of a 3D UI at various levels of abstraction. This approach has many benefits notably of being reproducible and allowing high level reasoning.

To conclude on these approaches, similarly as [Limb04c], we operate a three step analysis:

First, a set of selected *observations* is provided. An observation is a synthetic and descriptive assessment (as opposed to a normative assessment) that is made regarding properties of surveyed transformational methodologies.

Second, *shortcomings* are outlined from observations. A shortcoming is a normative assessment that is made regarding a property of surveyed transformational methodologies. A shortcoming is normative in the sense that it positions the state of the art with respect to ideal properties identified in the software engineering literature.

Third, a set of *requirements* for a solution to overcome the above mentioned shortcomings is identified. The internal validity of the solution proposed in this dissertation will be assessed with respect to this set of requirements.

2.3.1 Observations

Observation 1: Methodological diversity. Surveyed tools can be categorized into different categories depending on their main goal. While some are interested in Interaction techniques, InTML, other just in the 3D UI description and its behavior, CoGenIVE, Contigra, others in world content, VR-Wise. A method that covers almost the whole path, starting with goal-oriented task models, followed by exploration and navigation descriptions, finishing with interaction in response to system initiative. Each stage of the model is associated with generic design properties. Even that this method is tool independent, it is sometimes difficult to map the recommendations to physical actions in the tool.

Observation 2: Inter-method conceptual similarities. The Concrete UI (VR-Wise and Contigra) and the behavior (Contigra, CoGenIVE) are to of the most common models shared by the applications.

Observation 3: Inter-method conceptual dissimilarities. Of course as there are similarities, of course there dissimilarities between methods. Each method follows its own perspective and it is difficult to have a single one. Users-tasks are not considered. There is a common intention in including this model, for instance [Cupp05]. The rest considers interaction techniques, InTML, other just in the 3D UI description and its behavior, CoGenIVE, Contigra, others in world content, VR-Wise.

Observation 4: Conceptual openness. Method seems to have a particular concern for extension possibilities of their underlying ontology.

Observation 5: A focus on not just on the graphical modality. All environments deal at the first place with graphical modality but also include models of inputs devices; CoGenIVE emphasizes the haptic modality but is not limited to it; InTML interaction techniques also consider the input/output channels. In 3D applications this is very useful.

Observation 6: Standards do not deal with multimodal interaction. VRML and its predecessor X3D, so as Java 3D, are widely used in Web applications but Input modalities different from the mouse and keyboard are still straightforward. Solutions that deals with haptic interaction are build in C, CoGenIVE.

Observation 7: Transformations are not first class citizens. Transformations are in most methods hidden to the designer (i.e., built-in), untraceable and, not modifiable. In some environments, though, rules can be parameterized by dialog wizards (CoGenIVE). In no method a designer is provided with a stand-alone language allowing her to define custom transformation rules.

Observation 8: Single entry point, single exit point. Methods define their development process with one single entry point (i.e., the development process starts from an imposed artifact) and one single exit point (i.e., the artifact resulting from the development cycle is fixed by the method).

Observation 9: No complete Life-cycle development methodologies. The development life cycle of 3D user interfaces (UIs) mostly remains an art more than a principled-based approach. The methods reviewed rarely provide the design knowledge that should be typically used for achieving each step. InTML describe requirements in informal document with out any standard format, user task is still not considered. CoGenIVE has considered task models but without tool support. The VR-Wise and Contigra projects do not consider task models. In addition, the development life cycle is more focusing directly on the programming issues than on the design and analysis phases, examples of this are still CoGenIVE and Contigra. This is sometimes reinforced by the fact that available tools for 3D UIs are toolkits, interface builders, rendering engines, etc, all the toolkits reviewed have that purpose.

Observation 10: No user task formalization. The toolkits followed a content-centric approach, instead of a user-centered approach; hence, involvement of users in the requirements analysis and evaluation are not formally part of the methodology.

Observation 11: Lack of reusability from one language to other. The reviewed methods are usually restricted to only one programming or markup language and do not allow easy porting of code from one platform to another. For instance VRIWML is a Markup language for CoGenIVE, VR-Wise has their own language, Contigra extends X3D for their Behavior3D language.

2.3.2 Shortcomings

From these observations, we can conclude by presenting several shortcomings. All of them accordingly to the methodological aspect, these shortcomings concern the way existing approaches concretize transformational development with the definition of methodological stages, steps (i.e., transitions between stages), and transformation catalogs to perform these steps [Limb04c]. These shortcomings lead us to conclude that transformational development of user interfaces can be improved along several dimensions. The list of shortcoming is as follows:

Shortcoming 1: Lack of a methodology for developing 3D UI – Methodologies to cover the life—cycle development process is still needed. The novel scenario of using 3D UI is putting the effort more on the content rather than on models or methods. (Observation 1, 2, 3, 10).

Shortcoming 2: Lack of a User task models – The user task is not formally represented in any model as a starting point for the 3D UI development. When considered, the user tasks are described in documents, InTML

Shortcoming 3: Lack of Abstract models – There is no abstract level of specification, apart from the idea proposed in InTML of an abstract object that is independent of modality, called filter. Any definition is clearly linked with its final representation. So, any task can not be easily such as user task is not formally represented in any model as a starting point for the 3D UI development. When considered, the user tasks are described in documents, InTML

Shortcoming 4: Lack of transformational flexibility – Almost all methods are correlated to their final presentation, as their input is mapped one to one to the output.

Shortcoming 5: Lack of 3D UI toolkits – The toolkits are intended to help developers in Virtual reality content and the User Interface is not considered is this efforts. In some cases it is possible to write a script, as in Maya, Alambik. Then the file is exported in a format that the script handles, such as VRML, X3D. However there are no 3D UI models that can be reused to control the UI, such as buttons, menus, etc. There is always a need to start from scratch.

Shortcoming 6: Lack of Standardization – Apart from VRML, X3D and Java 3D, there area several languages that that intend to represent VR applications but they are following their own description and can not be interoperable.

Shortcoming 7: Lack of genuine 3D UI – When they exist, models of 3D UI are similarly to their counter part in 2D. In the worse case, java applets are used to interact with virtual world, this disrupting the Virtual navigation. When controls are rendered in the virtual space, even that they are rendered in the virtual space they mimic their counter part in 2D UI, there is no fancy representation taking advantage of the 3D space.

2.3.3 Ontological Requirements

We provide a list of requirement we seek to address with this dissertation. Some of these requirements are motivated by the above observations and shortcomings, some are desirable properties found in the literature that apply on any methodology. We want to introduce a 3D UI specification language which:
Requirement 1: Expressivity – means that a conceptual frameworkshould provide enough details to address problems that motivated the elicitation of its constituent concepts. In our context models should, at least, provide enough details to allow an implementation of the system it describes. This essential requirement is not fulfilled by many formal methods, for instance those focusing

on verifying state properties of the system that is being built (Motivation: general principle in software engineering, Obs. 6).

Requirement 2: Machine processable – states that the proposed ontology should be legible by a machine. To allow the transformational approach.

Requirement 3: Human readable – means that the provided ontology should be proposed in a format that enables its legibility by a human agent. Such efforts are done in InTML, CoGenIVE, VR-WISE and Contigra. Although the main concern is on machine processable.

Requirement 4: Standards – states that the expression means used to represent our ontology should rely on well accepted standards in the software engineering community, maybe using X3D as target language.

2.3.4 Methodological Requirements

Requirement 5: Methodological explicitness – states that the constituent steps of our methodology should be defined in a way that facilitates the comprehension of its internal logic and its application.

Requirement 6: Methodological flexibility – refers to the ability to initiate the development from any development stage (i.e., multiple entry points) and to terminate it at any development stage (i.e., multiple exit points).

Requirement 7: Executability – states that development steps should be expressed in such a level of accuracy that it is possible to execute them by an automaton.

Requirement 8: Methodological separation of concern. – refers to a partitioning of methodological steps according to the process types they realize (general principle in software engineering).

Requirement 9: Methodological extendibility – refers to the ability left to the designer to extend the development steps proposed in a methodology.

Requirement 10: Methodological Homogeneity – refers to the property of methodological steps of being defined using a common syntax**.** All transformation steps should be described in a single formalism that facilitates their understanding and processing.

Requirement 11: Methodological reuse – refers to the possibility in a methodology to capitalize on the knowledge defined by designers to perform development steps and re-using this knowledge for other developments.

Chapter 3 Three Dimensional User Interfaces Taxonomy

3.1 Introduction

Computational models are used in a wide range of application in exact sciences, so as in social and economical sciences [Robe83]. Two tasks are required to construct models. Firstly, the conceptual aspect that involves: problem definition, system conceptualization and representation of the model. Second, the technical aspect that involves: the behavior of the model, the evaluation and the policy of analysis and use of the model. To generate 3DUI models we have to evaluate and analyze 3DUI in the literature to extract the relevant characteristics of present developments.

For this purpose, taxonomy the applications will be necessary. This will be useful to identify the context of use of such applications. Also, will help us to understand the task in a more detailed level and how certain techniques address the task [Bowm00]. With taxonomies and implemented tools, hybrid models could be created or tested. New possibilities could be explored.

Several taxonomies have been proposed for 3D UI based on different characteristics, such as: interaction techniques [Poup98], metaphors [Bowm00], and 3D widgets [Dach02]. In our research we are proposing a new taxonomy that extends Milgram and Kishino continuum of mixed reality, [Milg94]. To introduce and define a wider spectrum of such interfaces while offering different basic or advanced mechanisms and techniques for virtualizing a user interface, we have extended this continuum by adding a more continuous range of UIs in the virtual part since it can be a 2D UI, a 3D rendering of a 2D UI, a genuine 3D UI manipulating 3D objects, and so forth. Moreover, a second dimension has been added to represent the degree of immersion that is allowed at each step: low and high immersion.

3.2 Interaction Techniques Taxonomy

Four are the main 3D interaction tasks that could be found in most complex 3D applications. In [Bowm01a] they are described as:

1. *Navigation* the most common VE task and is subdivided in two tasks, which are:
    a. *Travel* refers to the physical movement from place to place.
    b. *Wayfinding* the cognitive or decision-making component of navigation, and it asks the questions, "where am I?", "where do I
2. *Selection* is the picking of an object or a set of objects for some purpose.
3. *Manipulation* refers to the specification of object properties, such as: *position* and *Orientation.*
4. *System Control* is the task of changing the system state or the mode of interaction. Examples in 2D that do this are menus or command-line interfaces.
    a. *Implicit*
    b. *Explicit*

3.2.1 Metaphors Taxonomy

In a task model tree view [Pate97] we show the metaphors taxonomy for the manipulation task, inspired in [Poup98] metaphor taxonomy. The user has several means to achieve its manipulation task.

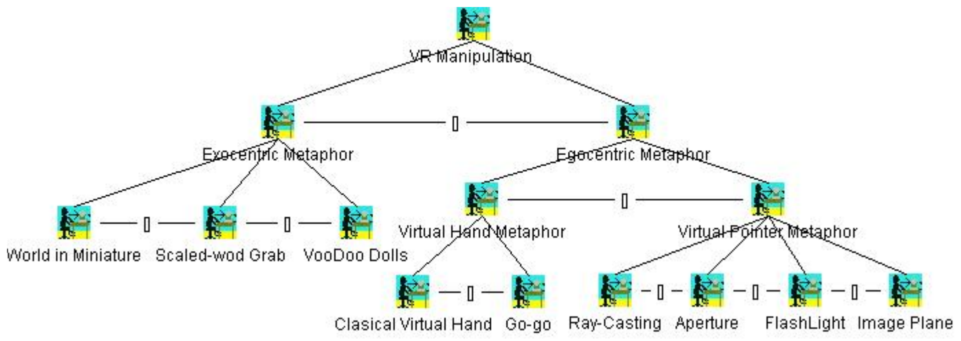


Figure 3-1 Taxonomy of virtual metaphors.

A metaphor in a VE is not an "atomic" but rather a "composed" interaction technique, which makes it harder to model. Leafs of the task model, presented in Figure 3-1, correspond to an Interaction technique itself.

3.2.2 3D Widgets taxonomy

Another important Taxonomy to consider is Contigra [Dach02], whose objectives not only include the standardization of a repertoire of 3D widgets, but also metaphors and interaction techniques, everything structured in the form a hierarchy. Among those widgets, the repertoire includes some of the elements that are used in almost every 2D and 3D application, such as the button and the toggle button, but also other widgets that are only specific to 3D environments, such as the ring menu. The disadvantage is that many of the widgets of this hierarchy have not been developed or are not publicly available yet, which limits its adoption as a standard in the field.

3.2.3 Mixed reality continuum Taxonomy

Milgram and Kishino continuum of mixed reality [Milg94] defines a continuum of real-to-virtual environments between Mixed Reality (MR). Their objective was the concept of having both "virtual space" on the one hand and "reality" on the other available within the same visual display environment.

The concept of a "virtuality continuum" relates to the mixture of classes of objects presented in any particular display situation, as illustrated in Figure 3-2, where real environments, are shown at one end of the continuum, and virtual environments, at the opposite extremum.

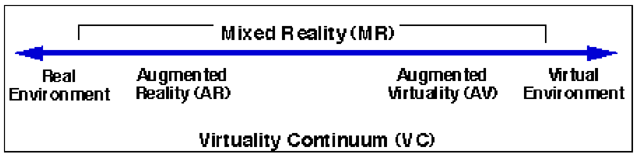


Figure 3-2 Simplified representation of a "virtuality continuum"

Milgram and Kishino differentiate their continuum based on the devices that render the projection of the virtual world. The diversification of devices produce the different sensation of immersion, six displays were described: (1) monitor based (non-immersive) video displays, (2) the same video displays but using a head mounted display (HMD) rather than monitors, (3) HMD equipped with see through capability, (4) The same as 3 but using a video display of the real world, (5) completely graphic display environments, completely immersive, partially immersive or otherwise, to which video reality is added, and finally, (6) completely graphic but partially immersive environments.

Their taxonomy helps to distinguish among the various technological requirements necessary for realizing and researching MR displays, with no restrictions on whether the environment is supposedly immersive (HMD based) or not. To do so, they propose three questions that should be solved, See Figure 3-3.
How much do we know about the world being displayed?. The answer is what they called Extent of World Knowledge. On the left extreme no information is known about the world to be displayed, on the other extreme, complete information is known. In the middle of the two poles there are three scenarios, the system will identify where are the objects but not what are they, second the system knows what are the objects but no where they are, and the last one, the system knows both where the objects are and what are they.

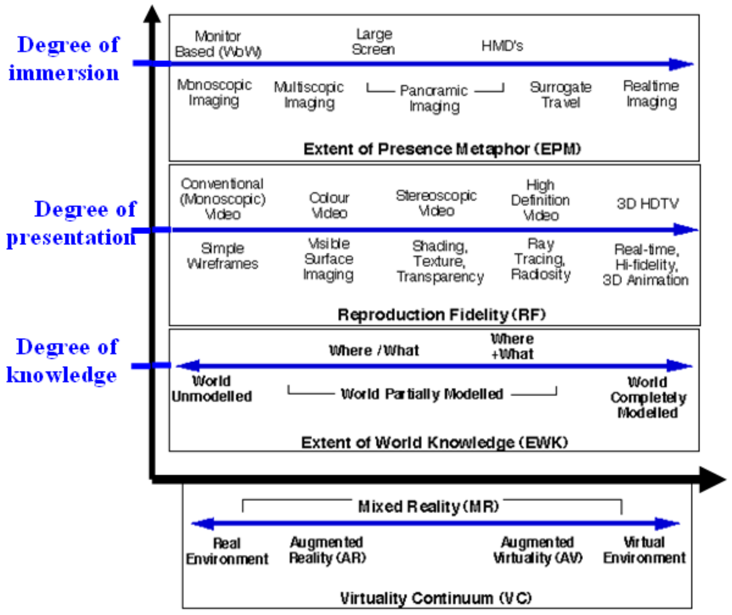


Figure 3-3 Extended representation of a "virtuality continuum"

How realistically are we able to display the world? the Reproduction Fidelity. This dimension deals with realism of the projection on the display, in terms of image quality and in terms of immersion, or presence, within the display. This differentiation is based on the device that is used to render the images.
What is the extent of the illusion that the observer is present within that world? They called this the Presence Metaphor. This element refers to the degree of immersion, which range from non-immersive (exocentric environment) to the high-level of immersion (egocentric environments).

In the above description of the continuum, summarized in Figure 3-3, the x axis denote the different level of mixed reality and the y axis denote all the elements the application should accomplished. For instance, if we consider a virtual environment, the there values for it are: 1) the maximum for degree of knowledge (full information of the model to be rendered), the highest quality for the presentation (i.e. Real-time hi fidelity, 3D animation, etc.), and finally (3) the highest possible degree of immersion (real time imaging).

3.3 A Taxonomy for 3D User Interfaces: Extension of the virtual continuum

Milgram and Kishino continuum of mixed reality [Milg94] just point to MR applications but actually the diversification of virtual worlds has more varieties than just MR. Our goal is to explore how we can expand, refine these specification to reach a wider spectrum of 3D user interfaces (UIs) rather than just MR applications. This offers a wide set of options for new developers to identify the type of application that they want to develop.

To introduce and define a wider spectrum of such interfaces while offering different basic or advanced mechanisms and techniques for virtualizing a user interface, we have extended the mixed reality continuum by adding a more continuous range of UIs in the virtual part (Figure 3-4) since it can be a 2D UI, a 3D rendering of a 2D UI (whether genuine as in our work or simulated), a genuine 3D UI manipulating 3D objects, and so forth. Moreover, we keep the idea of having dimensions to represent the degree of immersion. We propose just two levels: Desktop immersion (exocentric worlds) when the user is only looking at the screen (desktop virtual UI) or high when the user is really immersed in the system CAVE, HMD in a physical space (egocentric worlds) , similarly to [Poup98] for their interaction techniques taxonomy.

From the left to the right of Figure 3-4, we have respectively; a detailed description of each level of the taxonomy is presented bellow.

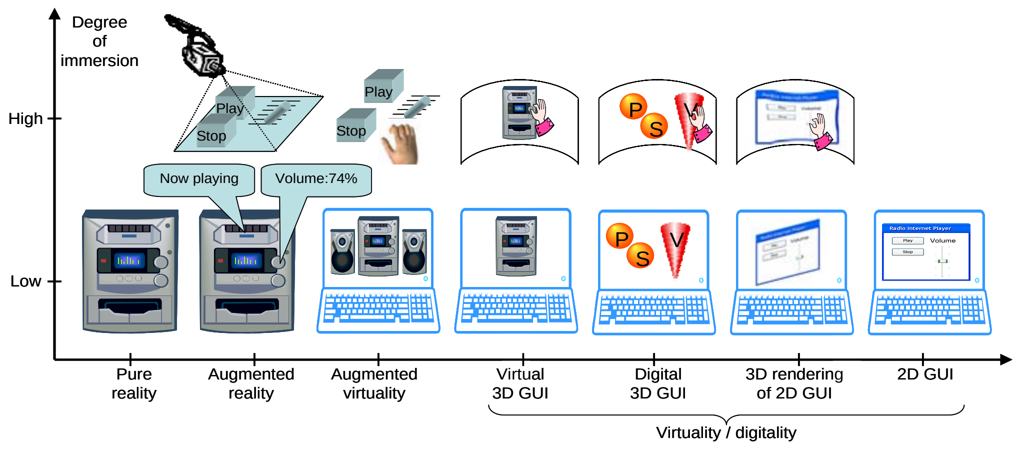


Figure 3-4 An extended continuum of user interfaces

3.3.1 Pure Reality

All the kind of displays listed above clearly share the common feature of juxtaposing "real" entities together with "virtual" ones. To differentiate between the two terms they proposed the following description, the "real" object is one that has an actual objective existence, an image that looks real, even could be generated by a computer (non-direct viewing of the object) and is presented in a display or is shown through a HMD, also, a real object has luminosity in its location. On the contrary, a "virtual" object exist in essence or effect but not formally or actually, it can not be sampled directly and thus it can only be synthesized, this means that an object even that could look real it is not, to clarify this the luminosity of objects do not exist, so the presentation is similar to holograms and mirror luminosity images. More clearly, a "virtual" image of an object is one which appears transparent, that is, does not occlude other objects located behind it.

Pure reality refers to real-world objects of hardware with which we interact, for instance, the stereo system in Figure 3-4. In pure reality objects we identify the source of motivation for the development of software interfaces, the innovation of applications, the born of new metaphors. As in 3D we have less restrictions of space as in traditional 2D desktop applications it is possible to create sophisticated applications that normally are inspired in a pure-reality objects. Several examples have been developed in this sense; see in Figure 3-5 the rotational menu presented in [Wang05].

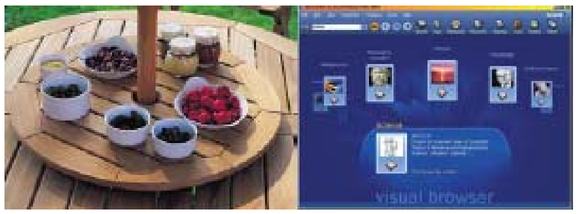


Figure 3-5 3D Carousels in daily life and computer interfaces

3.3.2 Augmented Reality

[Milg94] define augmented reality (AR) as "any case in which an otherwise real environment is augmented by means of virtual". AR interfaces allow users to effectively interact with augmented virtual objects as well as share them with each other in a simple and efficient manner, just as we do it with everyday physical objects. AR is quite appropriate for describing the essence of computer graphic enhancement of video images of real scenes.

In AR applications there are some kinds of Ubiquitous Computing, which change the way in which users interact with computers by providing (virtually) ubiquitous access to services and applications through a large number of cooperating devices. Also, the use of HMD applications are good examples of augmented reality when the see through capability is enabled. Figure 3-6 shows an example of an augmented non-immersive application. The real world reminds but through a HMD certain information appears in this case referent to the description of journals that are in a bookcase.

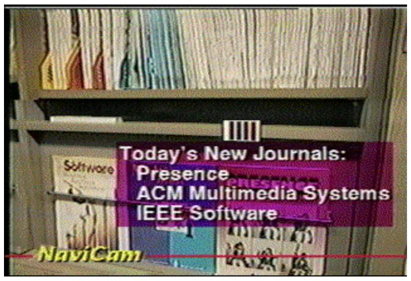


Figure 3-6 Rekimoto's NaviCam system and augmented Interaction 1995

On the immersive-degree of immersion case, the situation changes in the felling of the immersion but the user still interact with physical objects, in this category, we could classify the graspable interfaces, term introduced by Fitzmaurice 1995, [Fitz95]. This kind of applications refers to the projection of user interfaces that use of physical artifacts to control, organize and manipulate digital information. [Reki99] work (Figure 3-7), is an example of augmented reality with a high degree of immersion. Images are projected on a table and there is a tracking on the markers so the interaction of the world is tangible, physical interfaces + augmented reality interaction with computing devices.

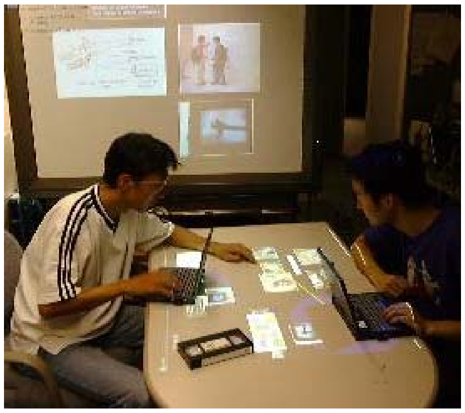


Figure 3-7 Rekimoto's Augmented Surfaces

3.3.3 Augmented Virtuality

A virtual word is one that is generated primarily by computer; those represent the augmented virtuality (AV) unlike virtual reality that replaces the physical world, AV enhances the physical reality by integrating virtual objects into the physical world which become in a sense an equal part of our natural environment. The low degree scenario of augmented virtuality is a virtual representation of a user interface that interacts with a physical device. We could use the interface displayed on a screen, on a wall, etc. and using an input device the user is capable to directly operate on the device through the interface as a remote control.

An example of augmented virtuality with high degree of immersion is the work of [Kiyo00] (Figure 3-8). The user is surrounded by a virtual representation that is projected through the HMD devices, but they also could see and interact with the real-world, as the see through capability is enabled.

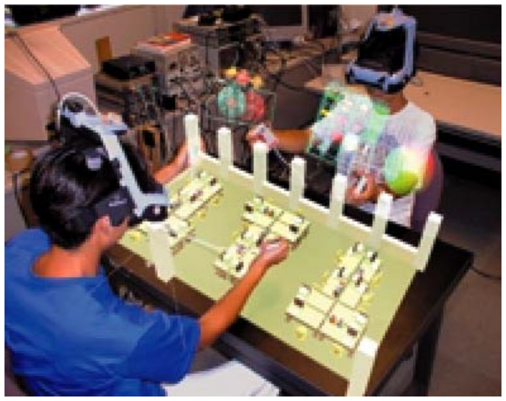


Figure 3-8 Kiyokawa et al. 2000

3.3.4 Virtual 3D GUI

A graphical user interface (GUI) is a program interface that takes advantage of the computer's graphics capabilities to make the program easier to use, generally are related to control programs on the computer. This is one of the extensions that we propose to the original continuum. We think that with the development of computer graphics, virtual desktop applications are more frequently used. This is what we called virtual 3D GUI. Notice that all what the user see is generated by the computer and there is no interaction with real world things.

A purely virtual 3D GUI, i.e. with low degree of immersion, consists of a world where virtual objects mimic their real-world counterparts, but displayed on the user‘s screen. All objects of the UI are virtual and directly operated by direct manipulation of them. The magic mirror of [Gros99] is an example of the use of a mirror metaphor to help the user to see what is occulted in the virtual world.

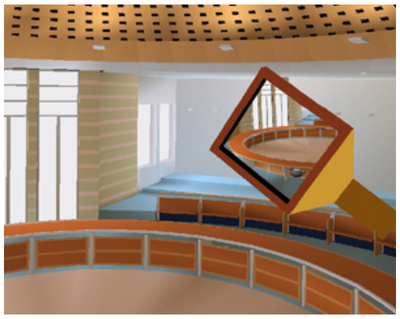


Figure 3-9 Magic Mirror

The high degree of immersion virtual 3D GUI is a graphical representation on a different display, not mandatory, than the traditional screen, a Cave environment could be an option. The interaction with the objects projected in the display is with the user's hands, but they could wear a HMD that recreates their hand. The image plane of [Pier97] (Figure 3-10) is an example of this type of applications. The user selects objects projected in a wall with his hand.

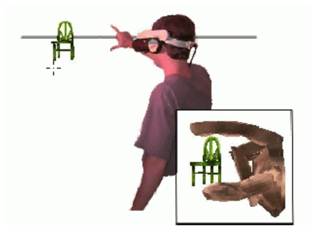


Figure 3-10 Image Plane

3.3.5 Digital 3D GUI

The digital 3D GUI is a 3D UI where 3D objects correspond to the tasks (e.g., a sphere to trigger the "Play" function) and the elements (e.g., a cone representing the current volume). The 3D objects used, not necessarily correspond to known or traditional metaphors. The differentiation with the previous definition is that objects are spatial, this means, they are not necessarily attached to a wall, a table or anything in the virtual world. To differentiate the degree of immersion, the only difference is the type of display used to render de virtual world, this means, low degree is the presentation on a screen, and high degree is the presentation on a Cave display.

In this category we also identify innovation, such as that done by [Dach01] with their Collapsible Cylindrical Trees (Figure 3-11), an innovative way to present menus in 3D.

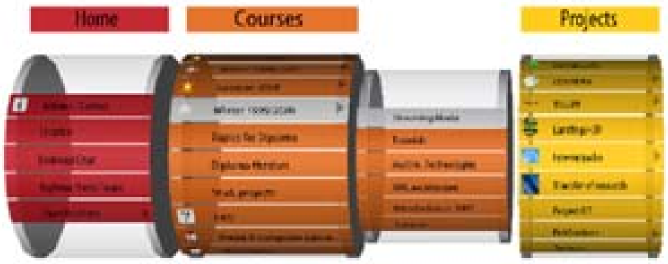
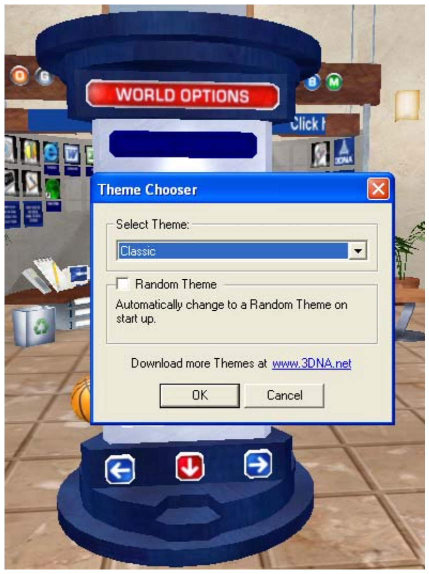


Figure 3-11 Collapsible Cylindrical Trees

3.3.6 3D rendering of 2D GUI

A 3D rendering of a 2D GUI in a virtual environment is typically a 3D desktop such as Task Gallery, Windows 3DNA (www.3dna.net), Figure 3-12, where traditional windows are manipulated in a virtual environment. To differentiate the degree of immersion, the only difference is the type of display use to render de virtual world, this means, low degree is the presentation on a screen, and high degree is the presentation on a CAVE-based UI where the classical 2D UIs are projected and manipulated by glove and hand recognition techniques.

[Moli05] proposed the VUIToolkit, a set of widget prototypes implemented in both VRML97 and X3D versions that makes possible to map interface elements described at the CUI (Concrete User Interface) level of UsiXML and those that have been included in the toolkit to allow the generation of a FUI (Final User Interface) in a VRML97 or X3D-based 3D environment. This toolkit quickly produces 2D GUIs rendered in a 3D environment, based on UsiXML language (www.usixml.org). The final result of their applications is shown below (Figure 3-13), and is in X3D and VRML format. One of the remarkable characteristics of this toolkit is that it transforms the standard plain 2D widgets into a truly 3D representation, not as in 3DNA, see Figure 3-12, in which dialog boxes are exactly the same as 2D GUI.



Theme Chooser

Select Theme: Classic

Random Theme

Automatically change to a Random Theme on start up.

Download more Themes at www.3DNA.net

OK Cancel

Figure 3-12 3DNA desktop application

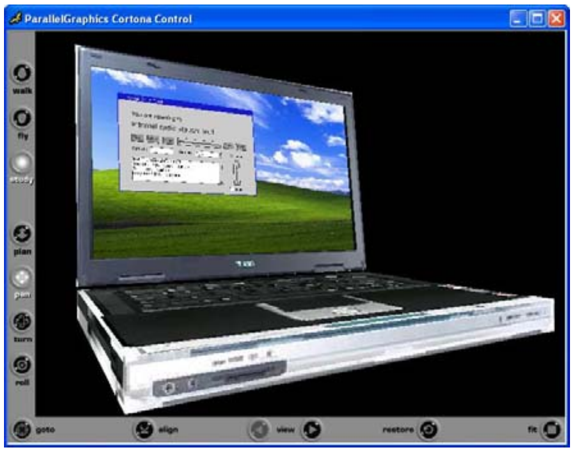


Figure 3-13 The virtual laptop with the rendered in VRML97

3.3.7 2D GUI

A traditional GUI is projected in 2D, there a lot of examples, such as: Windows, Microsoft Office, Internet Explorer, etc. Also, there a lot of programs, such as visual studio (Figure 3-14), Delphi, JBuilder, that are 2D GUI based that serve to develop new 2D GUI Applications.

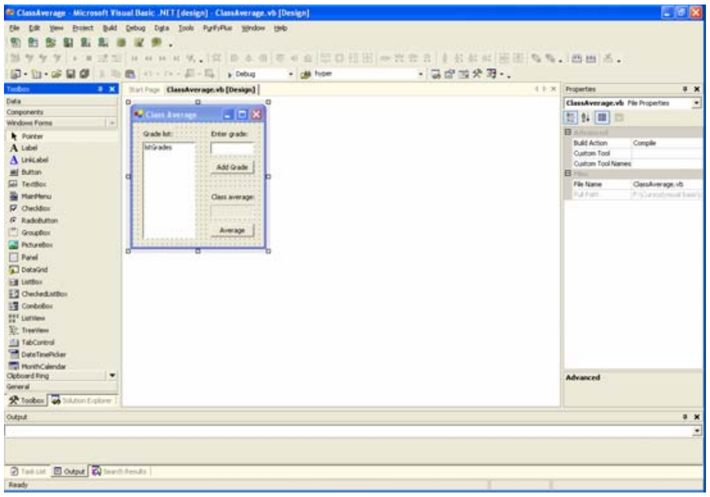


Figure 3-14 Visual Studio software tool to develop 2D GUI applications

3.4 The development of a internet radio player

In section 3.3 a Taxonomy were introduced. Now following this approach we would like to introduce an example to show the presentation of the same case study and how the different blocks could be covered. The Internet radio player, introduced in [Moli05], is an application to reproduce music.

3.4.1 Pure Reality

The true radio player as a physical device is shown in Figure 3-15.

Figure 3-15 A stereo system that reproduce music

3.4.2 Augmented Reality

A tangible UI, similar to the depicted in Figure 3-7, where physical objects are attached to the player functions (e.g., a physical cube for the "Play" function and a series of graduations for representing the volume of the loudspeakers). A camera captures the movements of the user and interprets them in the same way as is performing in another virtual UI, except that all operations are performed in the real world, with an effect in the virtual world, see Figure 3-16, which sketches this view. On the top Ceil of this virtual room the user movements on the physical devices on the table are tracked, the buttons on the table could be manipulated by different users. The virtual view of the radio player could be seen either on the table or on the back wall, and on the left the real object is affected by this manipulation. The real player could be also manipulated affecting the visual presentation.

This draw is just even that is in 3D is just imitating the real environment, so it shows the low level of immersion. However, if we imaging that the user is wearing a HMD an the user is looking at the same physical place, everything could change an looks as in the Figure 3-16, that is the augmented reality with a high level of immersion. Another feature that could be provided at this level of immersion is to provide information to the user when looking at the radio player, such as the volume, the song data, etc.

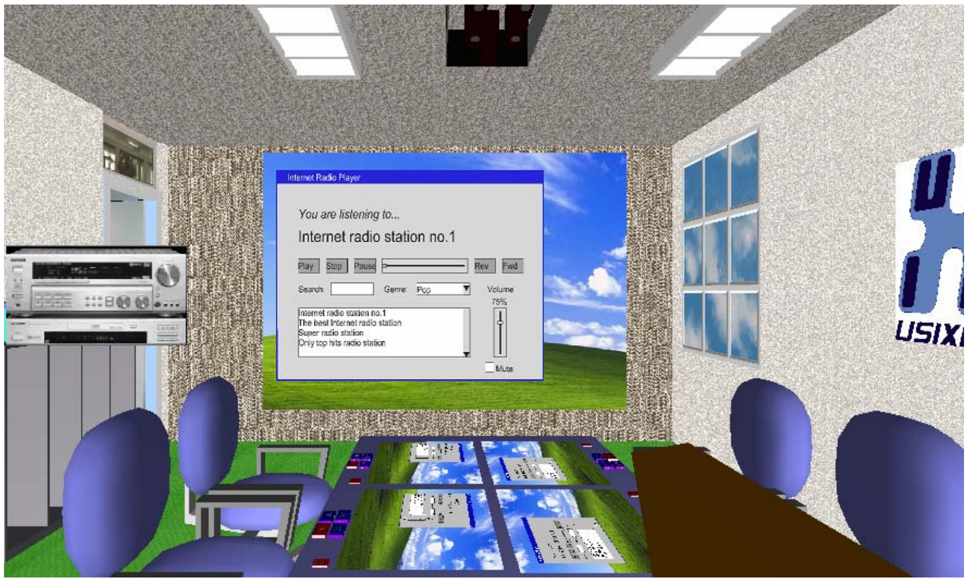


Figure 3-16 An Augmented Reality stereo system that reproduce music simulation

3.4.3 Augmented Virtuality

In the augmented virtuality there is no direct manipulation of physical object but still we could affect the real objects. Following the same example of the radio player, a virtual representation of the radio player, but with the incorporation of the true physical loudspeakers to directly operate on them through the interface as a remote control. Figure 3-17 also depicts an example of such an augmented reality UI. Virtually we will see a complete player but in reality there will be just the speakers connected to the computer. The view of the Figure 3-17 corresponds to the high level of immersion, as we could see everything as a whole in virtual reality. Also we could imagine not just the use of a HMD to display and see the player but also the user could be immersed in a CAVE and a user equipped with HMD and a glove can directly interact with these objects.

The opposite case as the presented above, low level of immersion could be as shown in Figure 3-4, in which a laptop renders the virtual projection of the radio player and is in there where we manipulate the speakers by manipulating the volume of the virtual player.

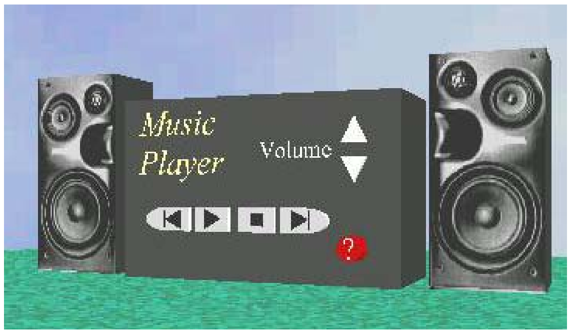


Figure 3-17 An example of augmented virtuality for the case study

3.4.4 Virtual 3D GUI

A purely virtual 3D GUI consisting of a world where virtual objects mimic their real counterparts, but displayed on the user's screen. All objects of the UI are virtual and directly operated by direct manipulation of them. Similar as the previous example, Figure 3-17, the difference relies on the speakers are also recreated virtually. In 3DNA (http://www.3dna.net/) desktop application they have a low level of immersion example of the radio player, Figure 3-18, scattered in the UI. The speakers are controlled either by double clicking either the virtual representation of them or the sound console next to the virtual representation of the player, which is behind the help label. The controls of the player are on top of the help label, and corresponds to the traditionally used for operation such as start or play, pause, etc. A CAVE-based UI, where the radio player is directly represented by its virtual scene. A glove similarly manipulates the user's hand to mimic the real world's operations.

One could say that the use of mouse and keyboard are physical inputs and as a consequence the augmented reality / virtuality is the same than the virtual 3D. Notice that in this case, the operations are performed in the virtual world as opposed in the real world in the tangible UI, presented in previous section. The main difference between them is the absence or presence and manipulation of physical objects to interact with the virtual world, so as the feedback that we can verify in the view over the physical device after an operation.

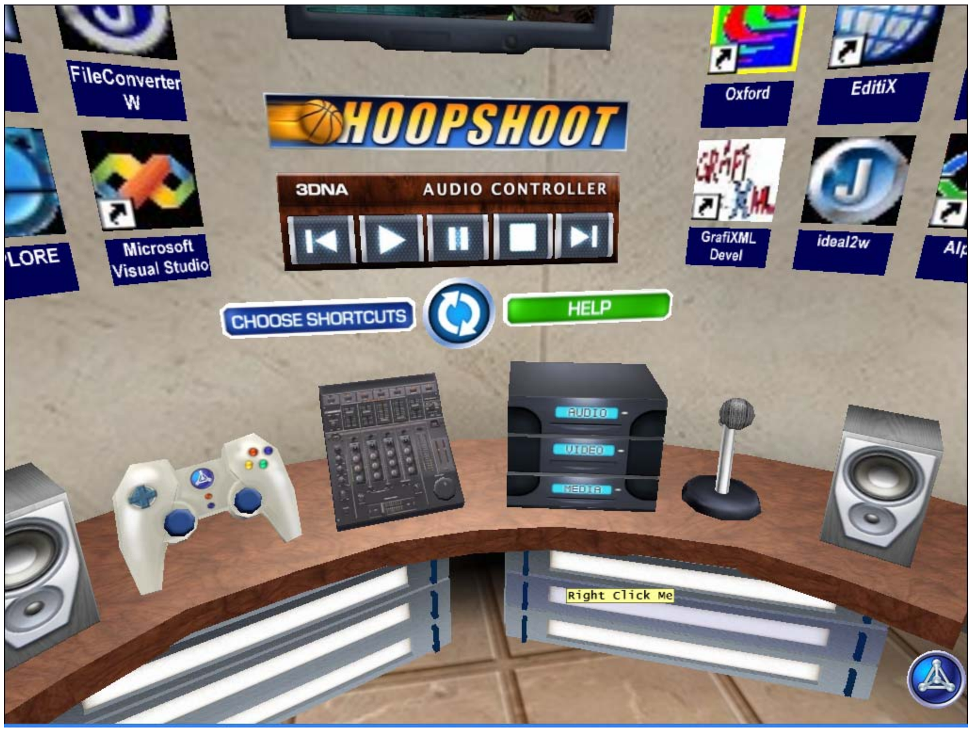


Figure 3-18 An example of augmented virtuality for the case study

3.4.5 Digital 3D GUI

A 3D UI where 3D objects correspond to the tasks (e.g., a sphere to trigger the "Play" function) and the elements (e.g., a cone representing the current volume). Figure 3-19 also depicts an example of such a 3D UI where all objects are really spatial. The rendering, whether using the screen or a CAVE-based display a HMD, etc., as in previous sections defines the level of immersion for this representation.

This approach promises and offers a wide option to designers as we could imagine an infinite range of option to represent in different ways what traditionally has been represented either in 2D UI or by imitating as "it is" the real world. We identify that at this level the potential of development has not been quite explored and bring us the goal of producing, what we called "genuine virtual representations" of the UI. We argue that the most different, special and attractive we made the 3D UI the most the UI will enjoy it. Of course this is a merely supposition that we will prove not in this dissertation but will be part of the future work.

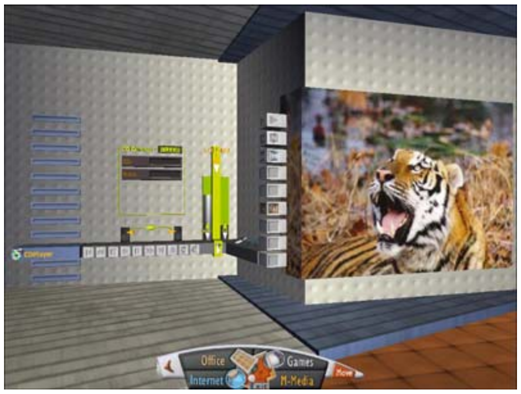
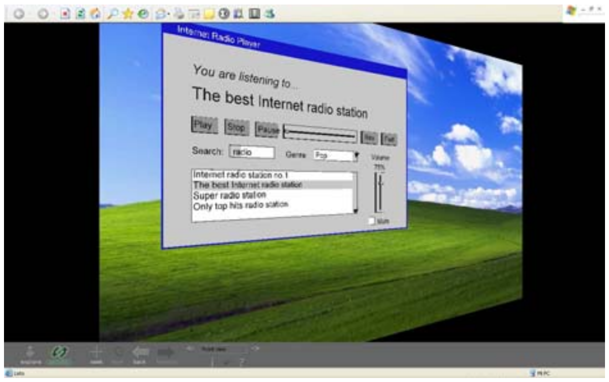


Figure 3-19 An example of a genuine 3D UI for the case study

3.4.6 3D rendering of 2D GUI

The VUIToolkit, introduced in [Moli05], is a rendering engine for 3D UIs specified in UsiXML in VRML97/X3D. In the screenshot of the Figure 3-20, we show the result of using the Toolkit that generates the 3D rendering of the 2D internet radio player. In this toolkit the 2D components have been enriched with volumes. As 3D widgets are rendered in a way that remains similar to the "Look & Feel" of 2D widgets, except that the "Feel" is a genuine 3D behavior, [Moli05] called this FUI "3D rendering of 2D UIs". This approach provides an option to the use of Java applets UIs to manipulate virtual applications in the Web, instead, the use of the VUIToolkit would not disrupt the 3D "look" as Applets does for web applications.



Search: radio
Genre: Pop

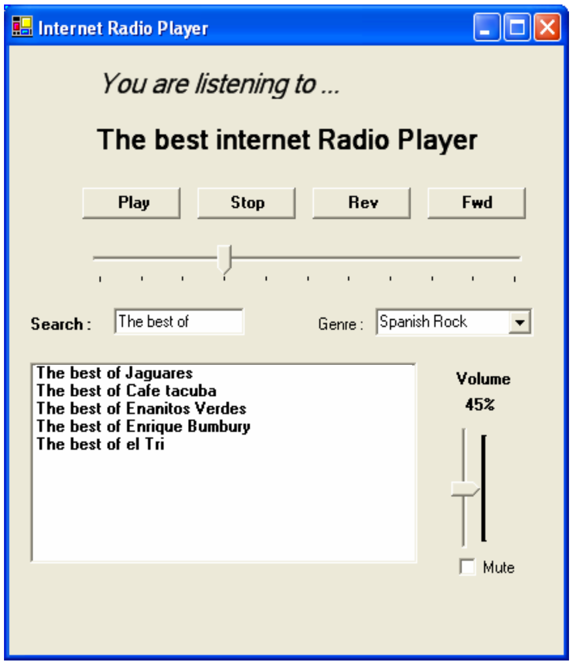
Internet radio station no.1
The best Internet radio station
Super radio station
Only top hits radio station

Figure 3-20 An example of the 3D rendering of a 2D GUI

As in the previous section the high level of immersion can be achieved with a CAVE-based UI.

3.4.7 2D GUI

A classical 2D form-based GUI made in Visual Basic for the radio player is shown in Figure 3-21.



Search : The best of
Genre : Spanish Rock

The best of Jaguares
The best of Cafe tacuba
The best of Enanitos Verdes
The best of Enrique Bumbury
The best of el Tri

Figure 3-21 A 2D GUI of the radio player case study

3.5 Conclusion

In this chapter we propose an extension of [Milg94] that corresponds not just to more than mixed reality applications but also to a new reality of Web-based virtual applications.

We identify that referring to the digital 3D GUI has not been well exploited as the relevant issues for designers are always on VR content rather than on the controls. We will work in the future of this dissertation to provide a solution to this lack of digital 3D GUI.

The taxonomy proposed tries to cover not just the types of VR applications but also two of the most important sources of inspiration. One of the goals of this taxonomy is to provide design ideas when a certain kind of control want to be developed in a specific type of VR, then designers could see how have been done, if there is a solution, or, if not, how at other levels of the taxonomy authors solved the problem. Ideally a software tool with a repository of solutions that can be reused can be the best but the work to do that is really considerable. First, because of the hardware required to manipulate the VR application, second because of the programming language used, and third because of the information sharing, most solutions described in papers are not open source so there is no way to have access to their code. As a consequence, there is a need to start from scratch.

As taxonomies are the results of the impression of their authors, researchers agree that there is no better or bad taxonomy, this depends on what they study, [Bowm00]. Taxonomies could be used to generate a particular classification, as we do. Also, is useful to understand the task in a more detailed level; understand how certain techniques address the task [Bowm00]. In our case the generation of the taxonomies allows us to identify the basic components to create a model–based approach for the 3D user interfaces, as is a way to capture the technical aspects.

Chapter 4 Model-Based Development of Three Dimensional User Interfaces

4.1 Introduction

This chapter addresses the ontological shortcomings and requirements of Chapter 2 and 3 by defining an original ontology aimed at describing various concepts relevant to 3DUI development.

The word "ontology" seems to generate a lot of controversy. It has a long history in philosophy, in which it refers to the subject of existence. It is also often confused with epistemology, which is about knowledge and knowing. In the context of information sciences, an ontology is a formal specification of a conceptualization [Grub93].

"*A conceptualization is a simplified representation of the world produced for some purpose. An ontology is, thus, a set of descriptions of the concepts and relationships within a field of knowledge used among a community of agents (humans or computers). Ontologies constrain the interpretation of concepts within a domain*" [Limb04c].

Notice that to create the ontology we could have two views of the world whether is objective or subjective. When we refer to concepts that are shared by a community of agents we refer to a subjective view of the world, as the concepts could differ from one research group to another. The advantage of such a view is that there is no wrong or correct ontology but a suitable one for the purpose that the research is looking for. With the time and its use ontologies could become a standard but then again this doesn't mean that is would be the only way to see this reality.

Several ontologies have been developed defining concepts related to 3DUI, for interaction techniques [Figu02], behavior in 3d [Pell05a] and mixed reality applications [Figu06]. Our ontology is inspired in the ontology specified in [Limb04c]. While defining ontologies we need to take care on *how* and *what* names are selected to define concepts. The most basic (abstract) foundational concepts and distinctions must be defined and specified. Again subjective views emerge at this step, as we can not assure that we already cover all the possibilities of concepts in a problem. This work requires several iterations which implies the reviewing of literature in order to enrich as much as possible the ontology, finishing when no changes could be done to it while reviewing more literature. One last step when defining ontologies is the definition of the presentation of the ontology, i.e., the meaning in real world terms of it.

As we said we rely in the framework proposed by [Limb04c]. In Figure 4-1 three essential components introduces any ontology: a conceptual content (i.e., abstract concepts), the formal foundations used to represent the ontology (i.e., abstract syntax), and the definition of the appearance of the ontology (i.e., concrete syntax). The structure of this chapter reflects these three aspects.

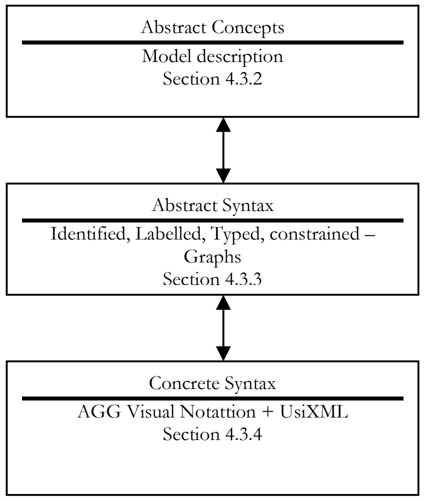


Figure 4-1 Limbourg language structure. Source [Limb04c]

In section 4.2 UML class diagrams for the ontology description along with definitions in natural language, the mathematical structures underlying this ontology, i.e., its abstract syntax described in [Limb04c], can be found complete in Annex A for the graphs definitions and Annex B for transformational rules. The notion of "directed, identified, labeled typed graph" is introduced, motivated and exposed, finishing with the concrete syntax of the language, which are: the visual (i.e., graphical) and a textual one (i.e., an XML language called UsiXML).

Section 4-3 presents the conceptual content of the extension to this language, which has been created considering just a general perspective of 3DUI, in UML class diagrams. The three layers will be completed in this section to clearly separate what it was done.

4.2 Ontology for Three Dimensional User Interface Specification

Every person in charge of the development of 3D Computer-Human Interfaces (3DCHI) will confront the necessity to find the set of elements that compose them. As we show in chapter 3 the variety of 3DUI options can be divided in five categories with two levels of immersion. However in each level several solutions have been proposed. This chapter is designed to show all the abstract concepts of such solutions in ontology that extends the Limbourg [Limb04c] ontology that do not consider 3DUI in deep, as is considering in detail 2D and multimodal UI.

4.2.1 Action types for Task Model

A task model describes the various tasks to be carried out by a user in interaction with an interactive system. The task model used in this methodology is similar as the proposed in [USIX06], which is an extended version of ConcurTaskTree (CTT) [Pate97], selected as it represents user's tasks along with their logical and temporal ordering, see Appendix D for more detail.

One of the attributes which is relevant for future transformation are the user actions. Several approaches have been used to user actions, for input device [Bles90], for 2D UI [Cons03], or for web searching engines [Jans06]. For abstract tasks that are independent of modality [Limb04c] rely on [Cons03], to define a user action as a tuple formed with: *Action type* and *action item*. This expression qualify a UI in terms of abstract actions it supports, a verb describes the type of activity at hand; an expression designates the type of object on which the action is operated. By combining these two dimensions a derivation of interaction objects supposed to support a task becomes possible.

However the set of interactive functions provided by [Cons03] did not express neither the set of tasks that actually users can not perform with information systems, nor the abstraction level required for task models that should be independent of modality and platform. For instance the view task assumes a graphical modality. The actual version, extracted from [USIX06], of action types for tasks is shown in Table 4-1.

actionType	Definition
start/go	Specifies that the AIO triggers an action
stop/exit	Specifies that the AIO puts an end to an action
select	Specifies that the AIO allows a selection action over multiple options
create	Specifies that the AIO is creating an item
delete	The AIO is dedicated to the deletion of items
modify	The AIO is dedicated to the modification of items
Move	The AIO allows the movement of an item
Duplicate	The AIO allows the creation of copies of an item
Toggle	The AIO specifies the existence of two different states of an item
View	The AIO allows to display an item using the graphical modality

Table 4-1 Action Types

Similarly as proposed by [Cons02] user action is composed of both elements of the action the type and the item required in the action. The set of items is listed in Table 4-2.

actionItem	Definition
Element	Specifies that the item has a single characteristic
Container	Specifies that the item is an aggregation of elements
Operation	Specifies that the item is a function
collection of elements	Specifies that the item is composed of a list of elements
collection of containers	Specifies that an item is composed of a list of containers

Table 4-2 Action Items

We would like to extend this description of action types by separating, first, task that are user type and secondly system type. Task models do not just model user interaction with the system but also action that the system performs. This separation of concepts will help designers in their task. Secondly, keeping the abstract specification as a basic description will be useful for the future transformation into an Abstract User Interface, but allowing the designer describing the task as "it is" and not in an abstract way. The task themselves belongs to an abstract category, so designers will not take care on understanding this abstract levels, also maybe they are not interested in developing a multiplatform – multimodal UI. However our action types categories will allow them to define a task that could be dependent of modality, does not limit the capability of the task model to be transferred to a different platform or modality of use. With the abstract types, it would be possible to switch between platforms and modalities. For instance, the task rotate object, clearly imply manipulation of object in 3D but if we just refer to and abstract task called "manipulate" or "move", in the virtual space it is quite ambiguous, as it can means a translation or a rotation, so we lost information with the use of an abstraction.

Our work, see Figure 4-2, is inspired in the user task taxonomies proposed by [Leno84], [Fole84], input devices [Gree88], haptic interaction [Hutc89], [Calh84], [Bles90] and [USIX06].

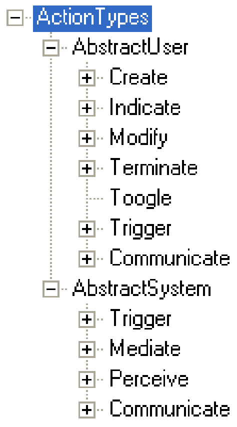


Figure 4-2 A Taxonomy of Action Types

The property *type* of the task normally is enough to infer to what type of user we refer. However, we prefer to continue with this distinction as the communication action of view (user action) is different to show (system action). So if just say that we have the communicate action and that inside this action we can view or show something, normally a document, if we are on the graphic modality, or say or describe in the vocal modality. The user can Create, Indicate, Modify, Move, Terminate, Toggle, trigger or communicate to the system and the system can perform some actions that are transparent to the user, as stated by [Leno84], the system mediate, perceive and communicate user actions, these actions are transparent for the user, for instance in a ATM transaction the user provides their code, and the system process it, check it and provide the user what they want but all these task are hidden to him. The provide code task generates system task, Figure 4-2. The description of the abstract action types is in **Error! Reference source not found.**.

actionType	Definition
Trigger	Specifies the triggers of an action
Terminate	Specifies the end to an action
Indicate	Specifies some sort of indication
Create	Specifies the creation an item
Modify	An action of modifying an item
Toggle	The existence of two different states of an item
Communicate	The action to communicate the user to the system or vice versa
Mediate	The action of mediate user actions
Perceive	The action of identifying any user action

Table 4-3 Action types

In **Error! Reference source not found.** we show the sublevels of abstraction for the **Create** user action type. That is composed of Associate that involves Name and Group items that intend to create associations such as radio groups.

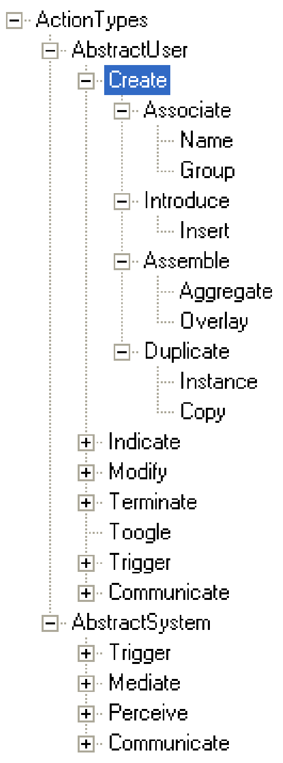


Figure 4-3 The Create Action Types Taxonomy

The action types describe until here are still independent of modality and platform. This change in the **Indicate** taxonomy, **Error! Reference source not found.**, because there are actions that are dependent on the input devices used and on the modality. The user can indicate abstract actions such as: position (even talking), reference, selection of any item, focus on any item, reach any item (even a position), locate an item, quantify. However, when referring to indicate by means of pointing, *sketching* to indicate some item, indicate a *scale* value, to indicate a *trace*, to indicate with the speech. One can say that, as input modalities, our senses are not actions but means to achieve those actions; we agree with that. However, touch any item to indicate some sort of select path, orientation, route, etc. For instance, if a text box can be filled: typing, speaking, or sketching the name. A further work for this proposal can be to identify the modality or platform that involves the action types. The different modalities (haptic, vocal, 2D and 3D graphic, and all sorts of body tracking: eyes, head, hands, arms, feet, even brain waves) can be the solution for grouping the actions. The context model, section 4.3.4, considers the input/output hardware channels used.

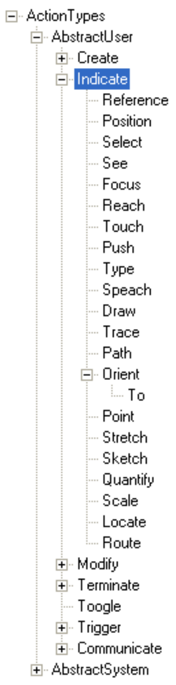
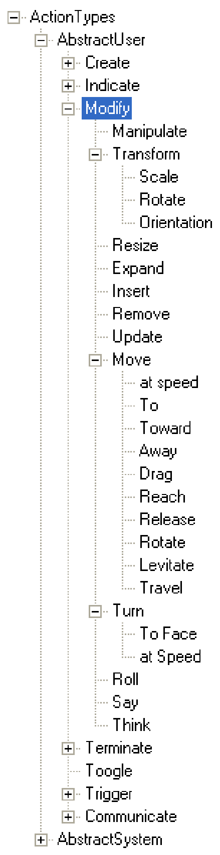


Figure 4-4 The indicate Action Types Taxonomy.

Through the transformational approach that we follow, attributes, such as the action type, consolidate final instances of the models when reifying (concretize an abstract model) them. Patterns of design and patterns of UIs can be used to identify and generate the FUI. The **Modify** action type, **Error! Reference source not found.**, refers to task related to change in some way the current state of an item.



```
ActionTypes
  AbstractUser
    Create
    Indicate
    Modify
      Manipulate
      Transform
        Scale
        Rotate
        Orientation
      Resize
      Expand
      Insert
      Remove
      Update
      Move
        at speed
        To
        Toward
        Away
        Drag
        Reach
        Release
        Rotate
        Levitate
        Travel
      Turn
        To Face
        at Speed
      Roll
      Say
      Think
    Terminate
    Toogle
    Trigger
    Communicate
  AbstractSystem
```

Figure 4-5 The Modify Action Types Taxonomy

The sub-actions considered again in some cases are abstract enough to be independent of modality, such as resize, expand, insert, remove, update. However the rotate operation does not have sense at least in 2D UIs.

The remain action types are in Figure 4-6. The **Terminate** action involves all sort of action that refers to end, finish, cancel, close, complete, etc. the **Toggle** action remains alone, just representing a two state item and the user action that manipulates this item by toggling it. The **Trigger** action refers to user action that launch or as its name says "triggers", traditionally system functionalities, it can be play an audio/video file, perform any action, function or method. The user to visualize information requires the **communication** action.

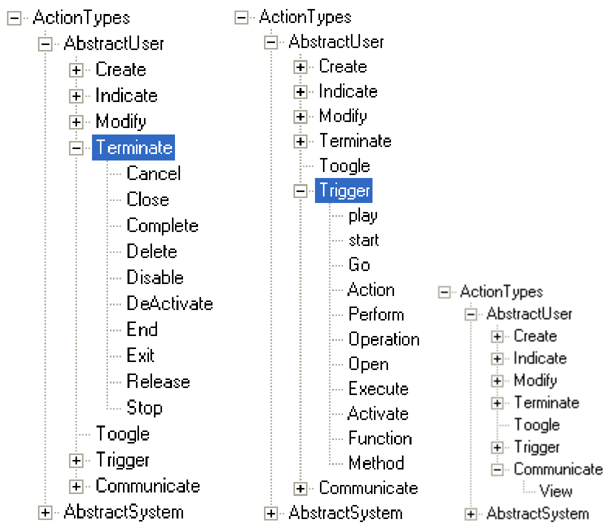
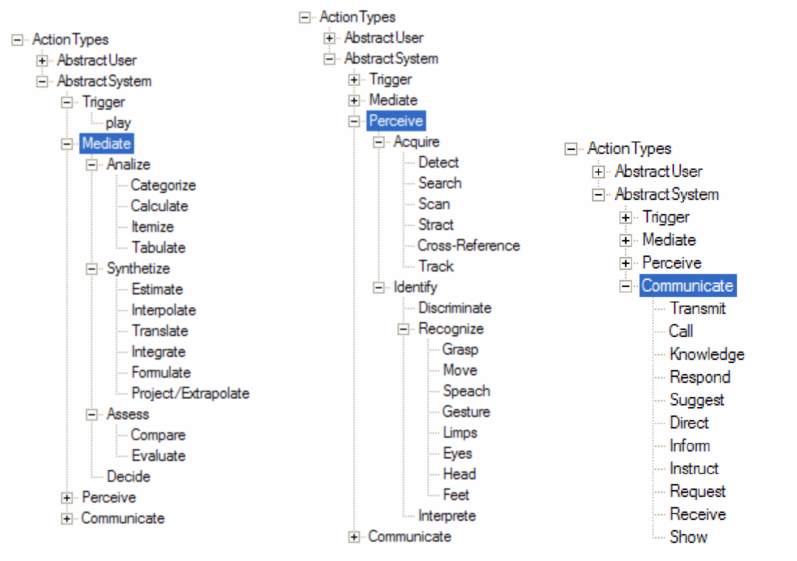


Figure 4-6 The Move, Terminate Toggle and Communicate Action Types Taxonomy

System action types, Figure 4-7, are normally triggered by the system by a direct or indirect user interaction. With direct user interaction, the user triggers an action that generates a system action. On the other hand the precondition attribute of the temporal relationships is used to specify a task that the system triggers with out any user interaction, for instance: backup the data base each 2 hours. In any case the system actions are transparent to the user [Leno84].



- Action Types
  - Abstract User
  - Abstract System
    - Trigger
      - play
    - Mediate
      - Analize
        - Categorize
        - Calculate
        - Itemize
        - Tabulate
      - Synthetize
        - Estimate
        - Interpolate
        - Translate
        - Integrate
        - Formulate
        - Project/Extrapolate
      - Assess
        - Compare
        - Evaluate
      - Decide
    - Perceive
    - Communicate

- Action Types
  - Abstract User
  - Abstract System
    - Trigger
    - Mediate
    - Perceive
      - Acquire
        - Detect
        - Search
        - Scan
        - Stract
        - Cross-Reference
        - Track
      - Identify
        - Discriminate
        - Recognize
          - Grasp
          - Move
          - Speach
          - Gesture
          - Limps
          - Eyes
          - Head
          - Feet
        - Interprete
    - Communicate

- Action Types
  - Abstract User
  - Abstract System
    - Trigger
    - Mediate
    - Perceive
    - Communicate
      - Transmit
      - Call
      - Knowledge
      - Respond
      - Suggest
      - Direct
      - Inform
      - Instruct
      - Request
      - Receive
      - Show

Figure 4-7 System Action Types Taxonomy

We keep the abstract categories proposed by [Leno84], so as some of its categories, which are convenient, the system needs to analyze, synthesize, assess and decide when the system **Mediate**. Actions inside each category are still independent of modality, which is logic as they normally deal with data not with the UI, even that a feedback of such actions could be provided to the user. This task is done by the system action **Communicate.** Finally, the **Perceive** action refers to the event handling, so as input devices recognition, which in its sublevel is clearly modality dependant.

Finally, the abstract types of item that the abstract task can manipulate are described in table 4-4.

actionItem	Definition
Element	Specifies that the item has a single characteristic
Container	Specifies that the item is an aggregation of elements or collection of elements
Operation	Specifies that the item is a function
Data	Specifies that the item is any data value
collection of elements	Specifies that the item is composed of a list of elements
collection of containers	Specifies that an item is composed of a list of containers

Table 4-4 Action Items

4.2.2 Three Dimensional Task Patterns

In software engineering, a design pattern is a general repeatable solution to a commonly-occurring problem in software design [Gamm95]. A design pattern isn't a finished design that can be transformed directly into code; it is a description or template for how to solve a problem that can be used in many different situations. Imagine a task that involves classical data base operations, insert, delete, modify, locate, we can imagine a predefined UI to handle this in different modalities, even code can be automatically generated so as the data base, using the domain model. However the current state of our methodology does not consider task patterns. In this section we refer to task patterns identified in the literature for common 3D User task, such as: navigation, selection, control, manipulation. These tasks not also have different tasks to achieve them but also interaction techniques. These patterns will help designers in their task when designating an application in general. As this idea of task patterns have several contexts such as: the organizational task patterns (ongoing work), data base task patterns, audio/video handling. Ideally those patterns should be used in a software tool, such as IdealXML [Mont05], that provide designers.

Anther advantage of such patterns is not just at the task design level, with the reification between models, it can be possible to derive a UI following a well know pattern. *GraphiXML,* a graphical tool to draw 2D user interfaces, uses a set of predefined menus that corresponds to common user tasks, such as editing, file, view, format, etc. Such kind of help when designing would reduce the time designated to this task.

From [Bowm01a] we summarize the Universal 3D interaction tasks: **Navigation, Selection, Manipulation** refers to the specification of object properties (most often position and orientation, but also other attributes)., **System Control** is the task of changing the system state or the mode of interaction. We introduce some task patterns in the next sections identified, these a preliminary work that will be complemented in the future. Details on the implementation can be found in Siggraph (http://people.cs.vt.edu/~bowman/3dui.org/course_notes.html) courses.

4.2.2.a Navigation pattern

In [Bowm01a] navigation is defined as the composition of two concurrent tasks: travel and Wayfinding. Similarly [Tan01] define three tasks which are: knowledge (wayfinding), search (travel) and inspect (system control). We consider the proposal of [Bowm01a], as system control in our opinion is not part of the navigation task itself. The user can navigate thought controls that can be defined in the travel task.

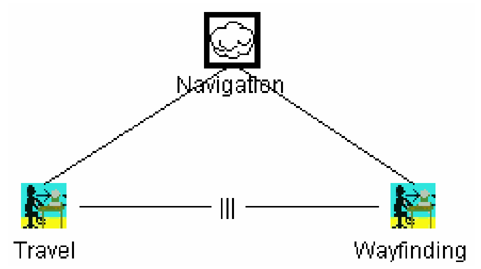


Figure 4-8 Navigation pattern

4.2.2.b Travel pattern

Travel is the motor component of navigation and just refers to the physical movement from place to place. The described Interaction techniques in [Bowm01a] for traveling are:

- Gaze-directed steering using head tracking [Mine95]
    - Indicate direction
        - User move head toward the desired direction
        - System track direction
    - Translate to view point selected
- Pointing using hand tracking [Mine95], [Bowm97b]
    - Indicate direction
        - User move arm toward the desired direction
        - System track direction
    - Translate to view point selected
- Map-based [Bowm98]
    - Indicate direction
        - Select icon
        - Drag icon
        - Release icon
    - Translate to view point selected
- Grabbing the air [Mape95].
    - Indicate direction
        - Select position (pinch or click on it)
    - Translate to view point selected
    - Stop selection
        - Release button or stop pitching

The pattern for traveling is depicted in Figure 4-9. As explained above, several interaction techniques allows the travel task but all of them considers indicate position and to translate the view.

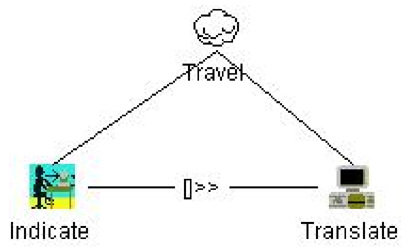


Figure 4-9 Travel Pattern

4.2.2.c Wayfinding

Wayfinding is one of the two pillars when navigating [Krui00], represent the cognitive or the decision-making component for user to define their path through the virtual world. [Krui00] identify different Wayfinding tasks:

- Extract Information
    - User position
- Build up spatial knowledge
    - World structure
- The user uses the spatial knowledge to make a decision

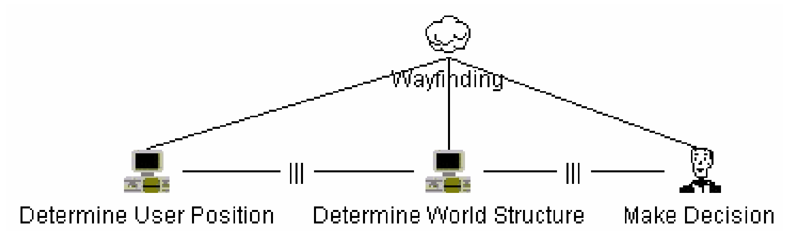


Figure 4-10 Wayfinding Pattern

4.2.2.d Select Pattern

The *Select* task is simply the specification of an object or a set of objects for some purpose. In [Bowm01a] four interaction techniques are defined to do this task in 3D, they are:

- Virtual Hand. The most common technique is the virtual hand metaphor that is the representation of the hand to touch objects as we do in the real world. There are two varieties exist, with augmented virtuality (without haptic feedback) or augmented reality (with haptic feedback).
    - *Indicate* position
        - Move hand
    - *Identify* Intersect/Collision
        - Hand position with objects
    - If any intersection then *Select*
- Ray-Casting is another common technique that uses the metaphor of a laser pointer, an infinite ray extending from the virtual hand [Mine95].
    - *Indicate* position
        - User Move hand
        - System Determine Ray Direction
    - *Identify* Intersect/Collision
    - If any intersection then *Select*
- Sticky finger [Pier97] is a technique that considers the occluded objects, with the virtual hand, that users want to reach.
    - *Indicate* position
        - User Move head
        - User Move hand
        - System Determine Ray position by subtracting hand position
    - *Identify* Intersect/Collision
    - If any intersection then *Select*
- Go-go. The go-go interaction technique [Poup96], inspired in inspector gadget ability to extend its arm to reach objects, introduces a non-linear mapping between arm extension and virtual hand position.
    - *Indicate* position
        - User Move hand
        - System Determine hand position based on its extension
    - *Identify* Intersect/Collision
    - If any intersection then *Select*

The select pattern is shown in Figure 4-11. Notice that the pattern consider the abstract task of indicate, several means can be used to do so, as pointed in this section by means of virtual hands for immersive VR applications, but is general enough to let open the possibility that a 2D based menu can be use to indicate the object. Information passes from this task to the system that has to identify if there is an object that collides with the indication made by the user, if so it will, mark the object.

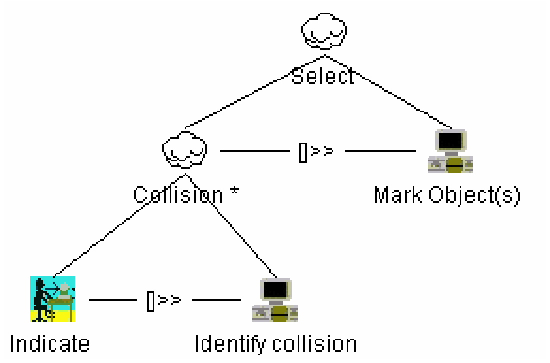


Figure 4-11 Select Pattern

4.2.2.e Manipulation pattern

There is a string link between the manipulation task and the selection. An object can not be manipulated if in some way has not been previously indicated. The manipulation task is, in some cases, linked with the selection technique. Special care must be considered when an object is released, what is going to be its position, when using a technique, such as go-go. In [Bowm01a] four techniques are described, which are:

- The virtual hand, used in all the family of hand techniques (go-go, ray-casting, any arm extension).
    - *Select* Object
    - Attached object to the hand
    - *Transform* object
    - *Release* object, depending on the technique the position of the object will be calculated.
- The Hand-centered Object Manipulation Extending Ray-Casting (Homer) technique [Bowm97a].
    - *Select* using ray-casting
    - *Translate* the hand to the selected object
    - *Transform* object
    - *Release* object.
        - *Translate* the hand back to its normal position
- Scaled-world grab is a technique that is related to the occlusion techniques for selecting.
    - *Select* using occlusion technique
    - *Translate* the user or world to the selected object
    - *Transform* object
    - *Release* object.

        - *Translate* the user or world to their previous position
- World-in-miniature (WIM) technique [Stoa95; Paus95] uses a small "doll house" version of the world that allows the user to do indirect manipulation.
    - *Select* Object using any virtual hand technique
        - Attached mini-object to the hand
    - *Transform* object
    - *Release* object, depending on the technique the position of the object will be calculated.

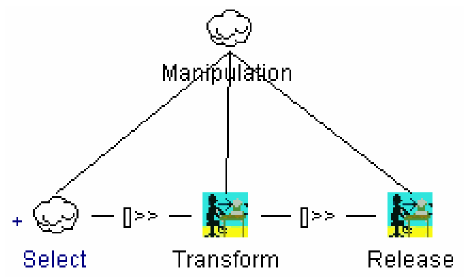


Figure 4-12 Manipulation Pattern

In Figure 4-12 we show the manipulation pattern.

4.2.2.f System control pattern

This pattern is quite complex as there are a wide range of operation that deals with system control. Deals with menus, buttons, speech, tracking, input devices known and new ones. Normally these techniques involve a sort of selection technique [Bown01a], but contrary to the select objects when a system control is selected some kind of feed back must be provided to the user. In Figure 4-13 we show a simplified system control pattern. Further investigation will be conducted.

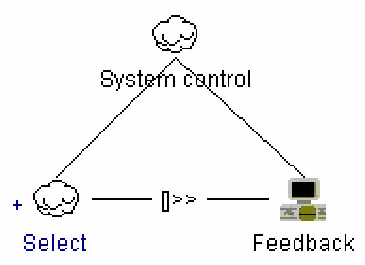


Figure 4-13 System Control Pattern

4.3 Three Dimensional Extension to the Concrete User Interface Model

The actual CUI do not consider components in the virtual space. As presented in our taxonomy (Chapter 3), we identify seven different categories of 3D user interfaces. We are neither interested in the physical UIs, nor in the 2D GUIs, as these are already described in the CUI meta-model. The remaining five categories (augmented reality/virtuality, virtual/digital 3D GUI, and 3D rendering of 2D UI) can be grouped, as for augmented reality and virtual reality the main difference between them are the input devices used to manipulate the UIs. For the 3D rendering of 2D GUI [Moli05] proposed the inheritance of the actual specification of 2D UIs (Figure D-6 and D-7 in appendix D) by adding new attributes (Figure 4-14) for the components described in the toolkit. Notice that this specialization of the model.

The main characteristics added to the components are references to position: top and left and to its size: width and height. As the 3d rendering of 2D GUIs are just imitating how 2D GUIs are, there is no need to include new attributes in general but just the correspondent to its size and position, similar approach is follow in [Cupp04] with the VRIXML that has its attribute position.

Another important feature of this approach is that labels for buttons, windows, checkboxes, combo box items are not anymore attributes of the components but components. The outputText component is specialized for such presentation. Even that this approach is inspired in the VUIToolkit [USIX06] the components are still abstract an independent of implementation; one can consider a transformation from this CUI model to different implementations such as: VRIXML, Contigra.

A second approach to extend the CUI model is to separate the graphical individual components in two groups: 2D based and 3D based (Figure 4-15). We assume that even that they have similarities, sometimes in behavior, presentation and characteristics; there are components and attributes in 2D that do not have sense in 3D and vice versa. In our proposal, similarly as in the 2D description, we separate in two groups our 3D Graphical Concrete Interactive Objects (3DGCIO), *3DContainers* (3DC) and *3DGraphicalIndividualComponents* (3DGIC). The 3DGCIOs are more specialized than their counter part in 2D this is depicted in Figure 4-16. As an infinite variety of graphical presentations can be proposed for both, 3DCs and 3DGICs a meta-object is proposed for generating 3DGCIOs.

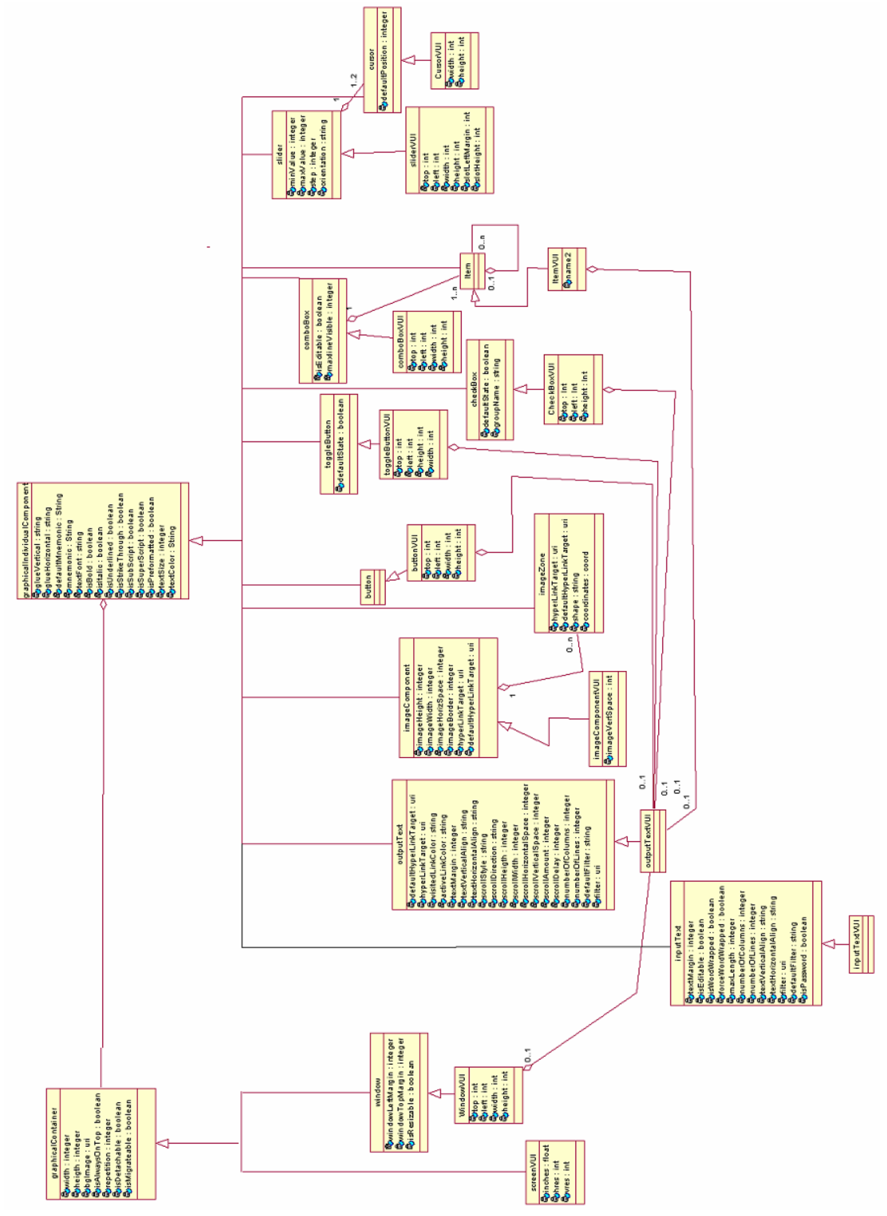


Figure 4-14 Concrete User Interface model extension to 3D rendering of 2D UIs

The attributes inherited from the CIO class, are relevant and have different representations. For instance labels on objects are depicted with the defaultContent attribute. An icon can be translated into a character or any object referred in the hyperlink.

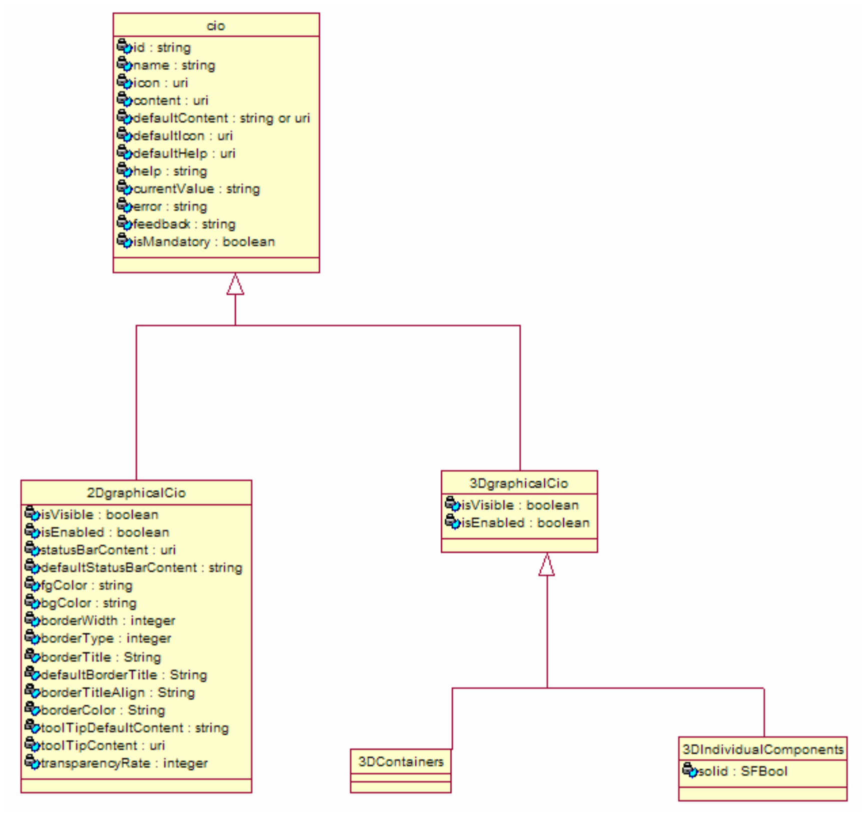


Figure 4-15 Three Dimensional CUI base classes

This CUI extension is based on the characteristics of: VRML, X3D languages and Maya, Blender and RawKee, software tools for modeling 3D, and Studierstube [Stud] and their set of 3D widgets [Maqu04]. Our meta-object 3DGCIO can be composed of a series of elements (sensors, appearance, grouping components and an abstract object SFNode). The model is depicted in Figure 4-16, which is coherent with [Web304a] standard abstract definition for the language X3D. The X3D abstract specification focus on abstract specifications of 3D content. The simplified model is shown in Figure 4-16, there are more grouping models, so as sensors, here we just show those used in our case studies, further diagramming will be provide as implementations an examples are developed following this standard proposal.

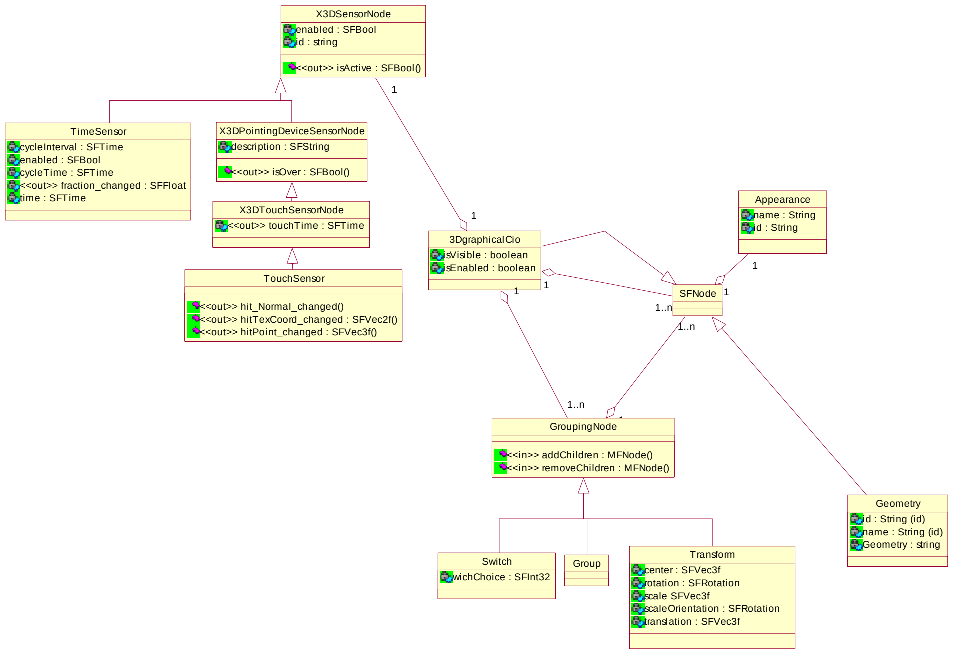


Figure 4-16 3DGCIO meta-aggregation relationships

The 3DGCIO can be composed of several SFNodes, which can be: predefined 3DGCIOs or any geometry, this class depicts any shape (cube, sphere, and polygons). With these capabilities, we can design any imaginable object for containers and individual components, apart from a set of predefined ones. The rest of the characteristics are described bellow.

Grouping nodes, which contain children nodes, are the basis for all aggregation. Several types of groping elements are defined for VRML and X3D. Among other groups we show just the implemented ones.

- Group. A Group node contains children nodes without introducing a new transformation. It is equivalent to a Transform node containing an identity transform.

- Transform. The Transform node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. The translation, rotation, scale, scaleOrientation and center fields define a geometric 3D transformation consisting of (in order):a (possibly) non-uniform scale about an arbitrary point; a rotation about an arbitrary point and axis; a translation.
- Switch. The Switch grouping node traverses zero or one of the nodes specified in the choice field.

X3DSensorNode. This abstract node type is the base type for all sensors. For instance the touch sensor (for the click on the triggers) and the time sensor (for the rotations).

Appearance. The Appearance node specifies the visual properties of geometry. The value for each of the fields in this node may be NULL. However, if the field is non-NULL, it shall contain one node of the appropriate type. The Appearance element as defined in Figure 4-17 is composed by five elements (Material, Texture, texture transform, fill Properties and line properties).

Material properties are related to light and color, whose attributes defined in [Web304a] are:

- Transparency. Defines how clear is the object, with a range of values [0.0(completely opaque), 1.0(completely transparent)].
- DiffusseColor. The color reflected by the object, this means, the color of the object.
- SpecularColor. The shiny color, this means, the shiny spots on the apple.
- AmbientIntensity. A double value that indicates the reflection of light from the object.
- EmissiveColor. This is useful for shining objects.
- Shininess. Combined with the specular color produce the shine effects, a low value produce soft glows, while higher results in sharper

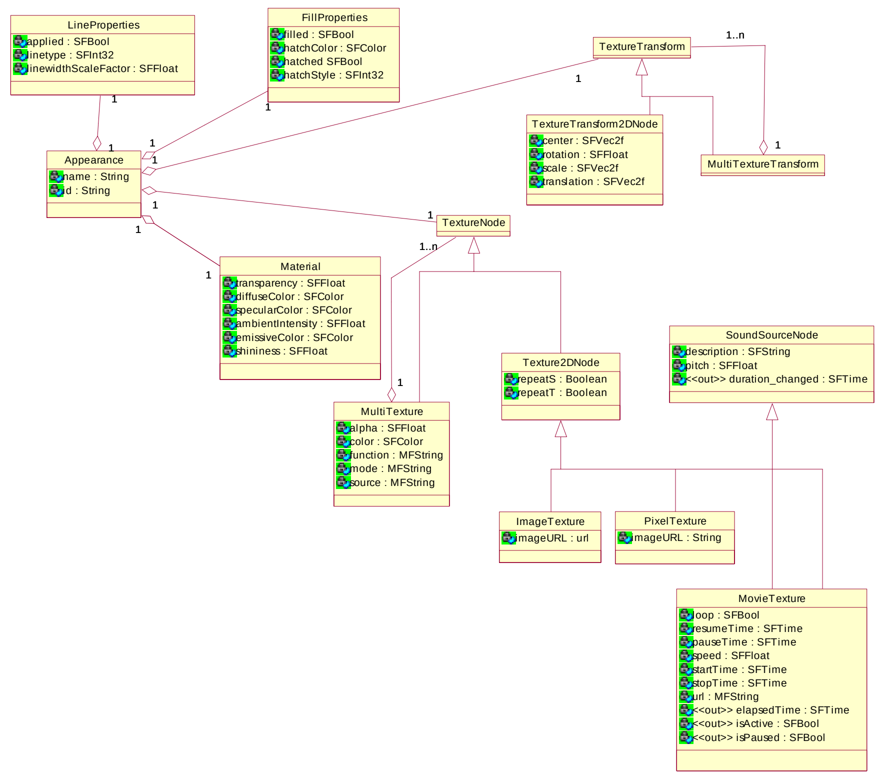


Figure 4-17 Appearance meta-model

The **LineProperties** node specifies additional properties to be applied to all line geometry. The *linetype* and *linewidth* shall only be applied when the applied field has value TRUE. When the value of the applied field is FALSE, a solid line of nominal width shall be produced. The color of the line is specified by the associated Material node. The **FillProperties** node specifies additional properties to be applied to all polygonal areas on top of whatever appearance is specified by the other fields of the respective Appearance node. Thus, hatches are applied on top of the already rendered appearance of the node. Hatches are not affected by lighting.

The **Texture** abstract node type is the base type for all node types which specify sources for texture images. Two are the types of textures that can be applied to objects, multi texture and 2d texture. The 2D texture defines ImageTexture, MovieTexture or PixelTexture, for each of them new attributes are defined. Further information about their properties can be found in [Web304a]. The MultiTexture enables the application of several individual textures to a 3D object to achieve a more complex visual effect. The **texture transform** specifies how the texture is applied to the object, as two type of texture exists; two texture transformations are required for 2D and for multi texture.

The meta-model for 3D **containers** and **GCIOs** is shown in Figure 4-18. More objects than those shown in the UML diagram has been analyzed but there is no implementation related.

We consider as much literature as we can covered to see the different possible presentations of these two GCIOs. As we pointed in chapter 3, by using our taxonomy we cover not just the potential implementations but also, if we did not find any, we offer to designers some clues of how the object has been developed, so they can imitate similar implementations.

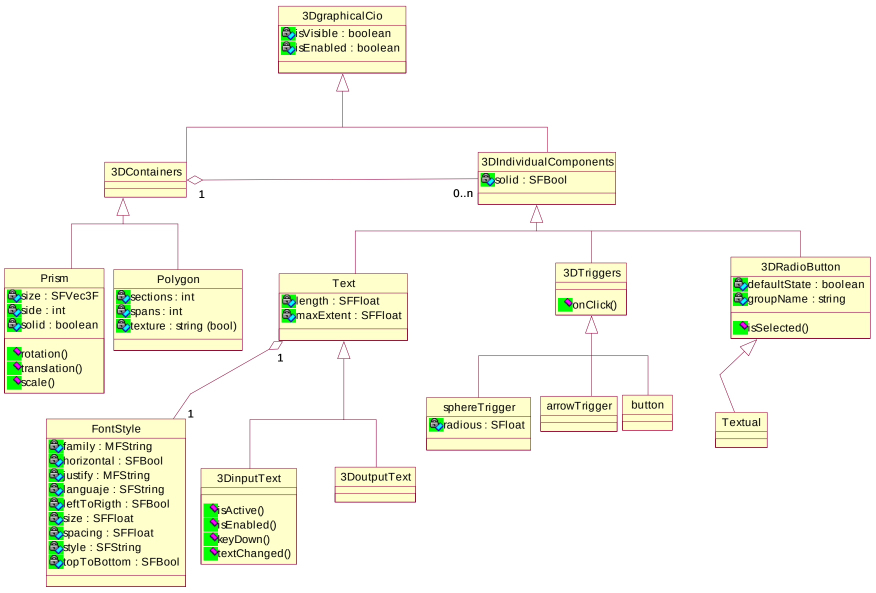


Figure 4-18 3D Concrete User Interface Meta-model

Ideally in the virtual space we could imagine an infinity set of objects that could be used as **containers**. The virtual space itself is the basic container for all the concrete interface objects (CIO), i.e., entities that users can perceive and/or manipulate. So we could have 2D renders such as *Polygons*, irregular or regular, n-sized; 3D renders such as: polyhedrons, which involves prisms, parallelepipeds, pyramids, cones, spheres; also we consider the fact that any combination of shapes can be used, as shapes could be created and function as a container, even, as shown in used in several examples shown in the state of the art, 2D container, such as windows, menu bars, are also considered as containers.

Two sets of object are described in this section, 3D Containers and 3D Concrete interactive objects. The set of Containers identified in 3D applications for the Mapping are:

Scene: This is the counter part to the window in 2D. The Scene is the virtual environment, so any object could be placed in the scene by just specify the x, y, z position.

4.3.1 2D renders

Polygons, regular polygons, n-sized:

Plane: the plane is a rectangle with height and width, text, graphics and many other objects could be attached to a plane; also different colors could be assigned to the plane to offer some syntactical related information. See Figure 4-19 below, in which several planes are use to render the windows in a 3D Sphere [Sphe05].

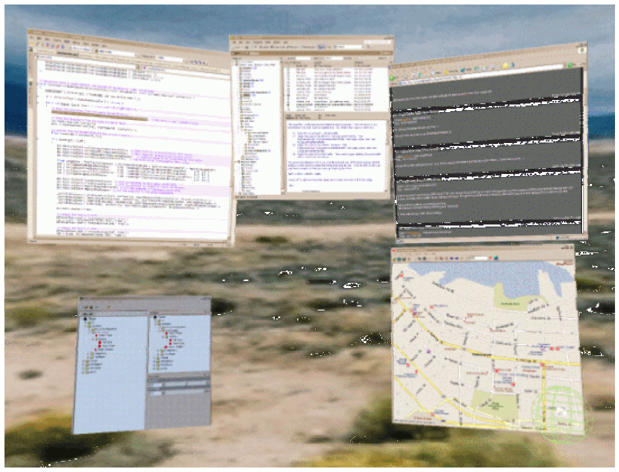


Figure 4-19 Scene with several planes that contains the windows of a Windows' system

4.3.2 3D Containers

Polyhedron is a three-dimensional shape that is made up of a finite number of polygonal faces [Wiki05]:

Figure 4-20 Polyhedron

There are different categories of polyhedrons that could be used as 3D Containers that are:
Prism is a *n*-sided polyhedron made of an n-sided polygonal base, a translated copy, and n faces joining corresponding sides [Wiki05].

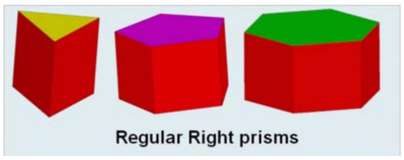


Figure 4-21 Regular Prism

A *Wall* is a rectangular prism, from a rectangle. The wall offers a more realistic 3D container compared to the plane, see Figure 4-19. The values of width, height and deep offer the possibility to create different objects, such as cubes. Also different effects could be possible to add to each side of the wall, such as color, transparency, see Figure 4-22 [Park98].

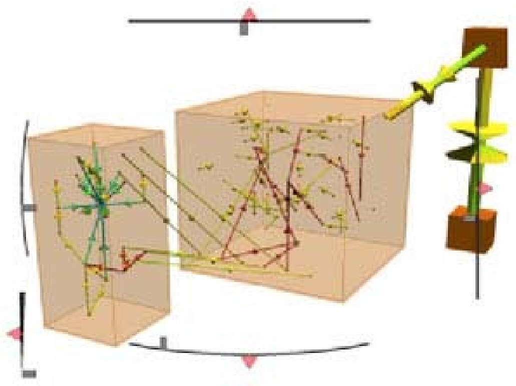


Figure 4-22 Wall Container with transparency

Considering that the prism use a regular polygon as its base even that the size could be n, notice in Table 1 that the increment in the number of sizes produce a figure close to a circle. I think that the 10-Sized polygon decagon, could be the as much the size.

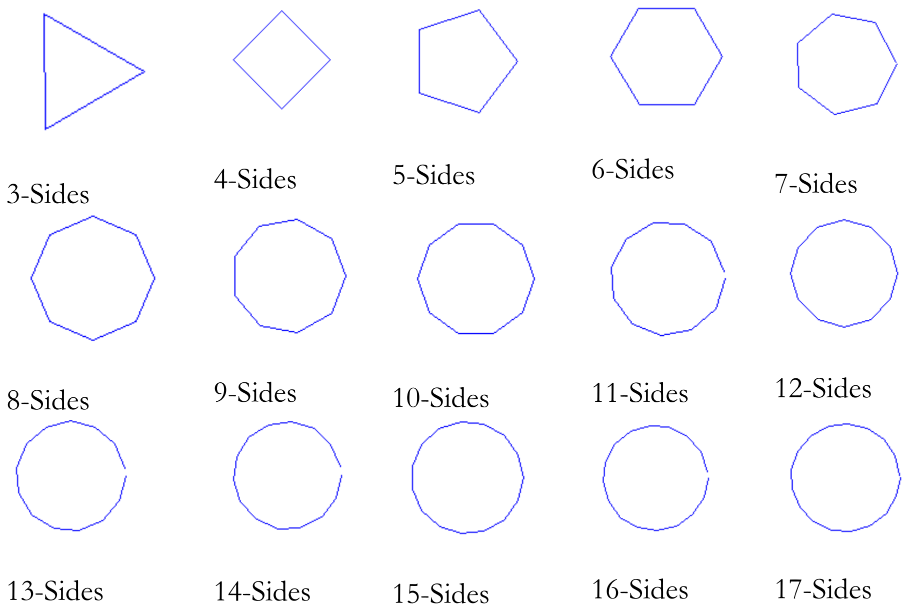


Table 4-5 Polygons approximation to a circle

The relevant aspect of considering the number of sides of the polygon is that it is expected that on each side of the polygon it could be possible to attached objects, so as the number of sides is incremented, the area to attach objects is reduced, there is a direct relation on this.

Parallelepiped or *parallelopipedon* is a three-dimensional figure like a cube, except that its faces are not squares but parallelograms [Wiki05].

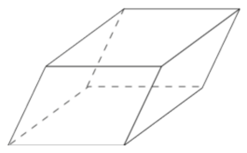


Figure 4-23 Parallelepiped

Pyramid is a polyhedron formed by connecting an *n*-sided polygonal base and a point [Wiki05].

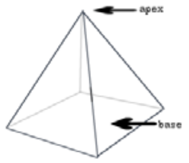


Figure 4-24 Pyramid

Cone is a solid object obtained by rotating a right triangle around one of its two short sides [Wiki05].

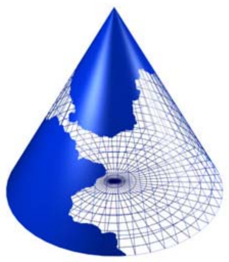


Figure 4-25 Pyramid

Sphere: A **sphere** is a perfectly symmetrical geometrical object. In mathematics, the term refers to the surface or boundary of a **ball**, but in non-mathematical usage, the term is used to refer either to a three-dimensional ball or to its surface [Wiki05]. The surface of the sphere could be used to attach objects but also Similar to the wall but instead a sphere could be used as a container, see Figure 4-26.

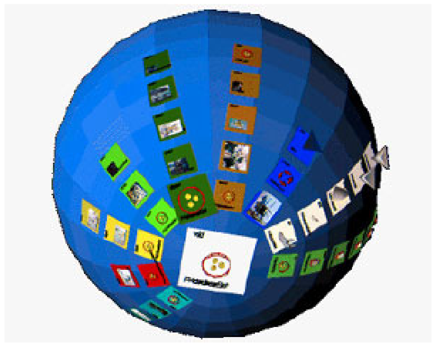


Figure 4-26 Sphere Container [Fair93]

4.3.3 Three Dimensional Graphical Individual Components

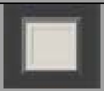
This section is based on examples developed in the Toolkits review in the state of the art, and the examples shown in the case study, so as some shown inside the taxonomy. The use of the taxonomy provides hints to actual solutions for designers. More important is the fact that there can be more than the examples shown in each level, further investigation can fill the gaps or add more examples to the existing levels.

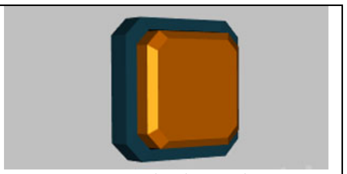
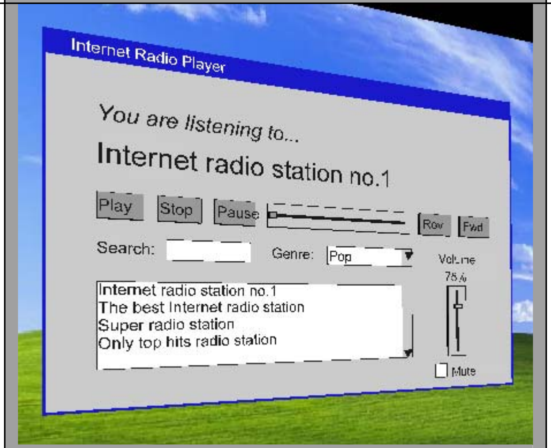
4.3.3.a Button

Definition : is a alternatively called trigger button as its aim is to trigger any kind of action available in the system [USIX06].

Abstract attributes:
defaultContent : the text component string that defines the text displayed on the button.
surroundedColor : the Appearance attribute that define the color and texture of the button around the button, see the black color described in Contigra.
centerColor : the Appearance attribute that define the color and texture of the button in the center of the button.

Abstract events:
click : the user clicks on the button.

Button		
	Immersive	**Non-Immersive**
Pure Reality		Buttons from a remote control
Augmented reality	 Button Studierstube	Bioelectric Control
Augmented Virtuality		

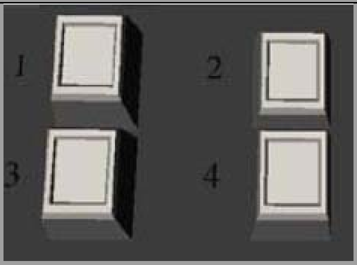
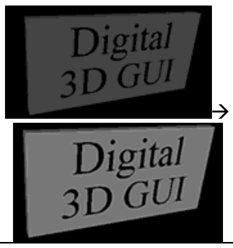
Virtual 3D GUI		 Representation in Contigra
Digital 3D GUI		
3D rendering of 2D GUI	 Internet Radio Player You are listening to... Internet radio station no.1 Play Stop Pause Rev Fwd Search: Genre: Pop Volume 75 % Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station Mute	
2D GUI		

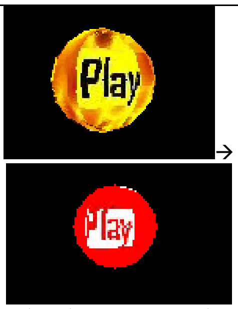
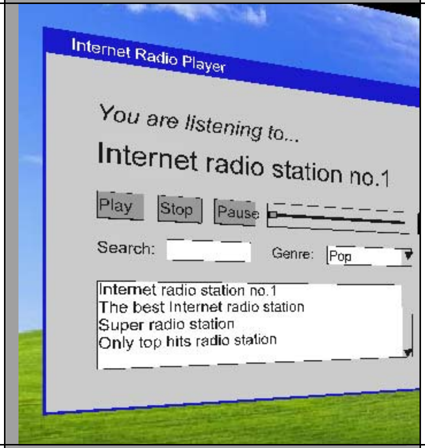
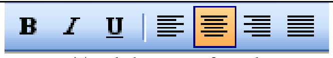
4.3.3.b Toggle Button

Definition : enables a Boolean choice by pushing a multi states button [USIX06]

Abstract attributes:
defaultContent : the text component string that defines the text displayed on the button. **(Label attribute Inherit in Studierstube)**
surroundedColor : the Appearance attribute that define the color and texture of the button around the button, see the black color described in Contigra.
centerColor : the Appearance attribute that define the color and texture of the button in the center of the button.
secundaryColor : the Appearance attribute that define the color and texture of the button when the button change the state.
defaultState : Indicates a default state for a toggleButton := TRUE FALSE. [USIX06]

Abstract events:
EVT_TBT_click : the user clicks on the button.

Toggle Button		
	Immersive	**Non-Immersive**
Pure Reality		Light interrupters
Augmented reality	 Toggle button in studierstube	
Augmented Virtuality		
Virtual 3D GUI		
Digital 3D GUI		

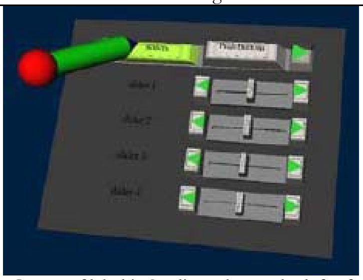
		 Unselected Selected
3D rendering of 2D GUI	 Internet Radio Player You are listening to... Internet radio station no.1 Play Stop Pause Search: Genre: Pop Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station	
2D GUI		 Toggle buttons of word

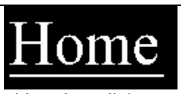
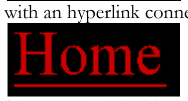
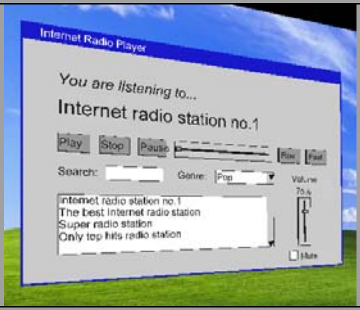
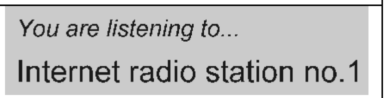
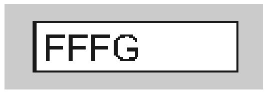
4.3.3.c Text Component

Definition : textual sentence that is placed in different components of 3D words or by itself on the space. It is useful to explain the use of components, to define type or units of measure, to set tittles.

Abstract attributes:
defaultContent : the vector of characters that compose the text of the text component. The text could be divided in different substrings and each one is place in one location of the vector, the purpose of doing this is to replace the use of the return character. [USIX06]
maxLenght : define the size of the space where the text is displayed (the size is vertical or horizontal depending on the orientation of the text) := [0.0 (any length) … ∞). [USIX06] (maxExtent in VRML)
maxLength : a vector that specifies the length of each substring of the text in the local coordinate system := [0.0 (any length) … ∞). [USIX06] (Lengt in VRML)

textFont : define the family of the label (times, serif, …). [USIX06] **(**FontFamily in VRML)
orientation : whether the text advance horizontally or vertically := horizontal, vertical.
advancing : define whether the text is written left to right or vice versa in the case of horizontal orientation, or top to bottom or vice versa in the case of vertical orientation := leftToRight, rightToLeft, bottomUp, topDown.
justification : determine alignment of the text := left, right and center.
style : specifies the style of the text := PLAIN, BOLD, ITALIC, BOLDITALIC. (isBold is Italic in [USIX06])
textSize : specifies the high of the text : = [0.0 (no size) … ∞) [USIX06] (Size in VRML).
spacing : determine the line spacing between adjacent lines of text, this means, when more than one line of text compose the text displayed := [0.0 (no size) …∞).
language : define the value of the language tag that is based on ISO 639:1988 := 'zh' for Chinese, 'jp' Japanese, 'sc' for Swedish.
defaultHyperlinkTarget : define a uri target := uri [USIX06].

Text Component		
	Immersive	**Non-Immersive**
Pure Reality		Text is everywhere in our life in any of its representations, as a string of characters, graphics or as spaces destined to write something
Augmented reality		 Group of label is Studierstube on the left of the column
Augmented Virtuality		

Virtual 3D GUI		 Home Label with an hyperlink connection  Home Feedback of the same label with the mouse on it
Digital 3D GUI		
3D rendering of 2D GUI	 Internet Radio Player You are listening to... Internet radio station no.1 Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station	 You are listening to... Internet radio station no.1 Labels  FFFG Text box
2D GUI		Search : Label in Visual Basic

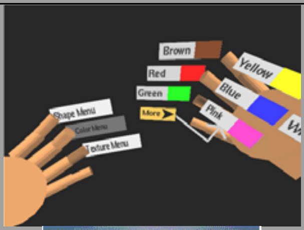
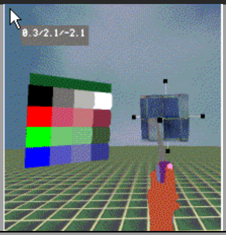
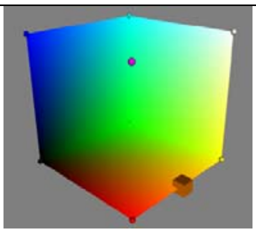
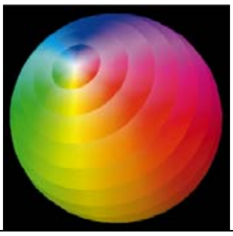
4.3.3.d Color Picker

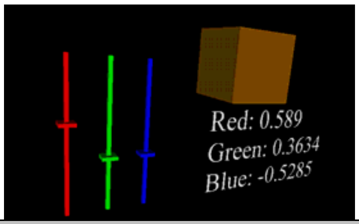
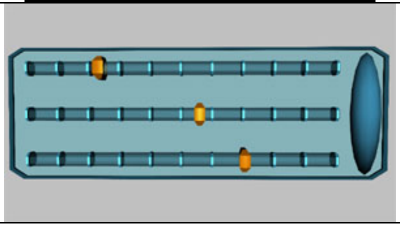
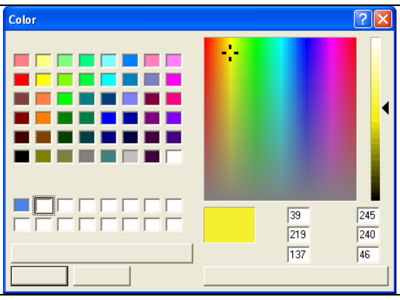
Definition : Enables to choose a color within a palette. [USIX06].

Abstract attributes:

color : a 3D color selected := <red> <green> <blue> the three values are double.

Color Picker		
	Immersive	**Non-Immersive**

Pure Reality		314 Bright Red 668 Jet Black 309 Arctic Silver 203 Morea Green From the '98 3 series coupe brochure Dated 9/97 Color catalog from a paint shop
Augmented reality		
Augmented Virtuality	 Brown Red Green More Yellow Blue Pink Shape Menu Color Menu Texture Menu 	
Virtual 3D GUI		 

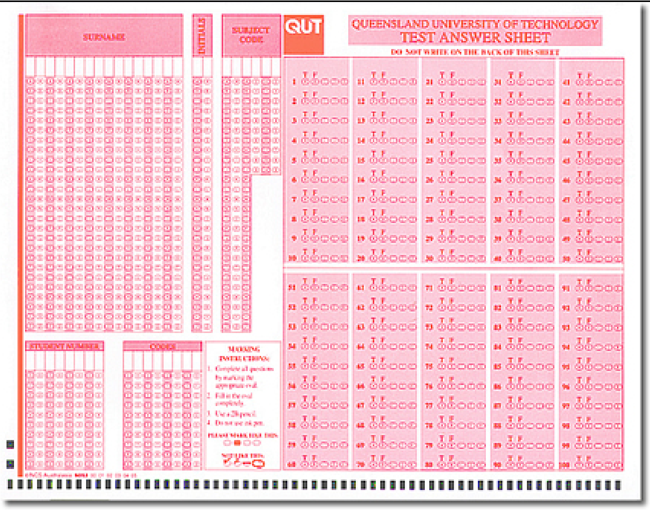
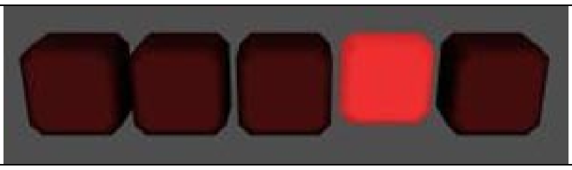
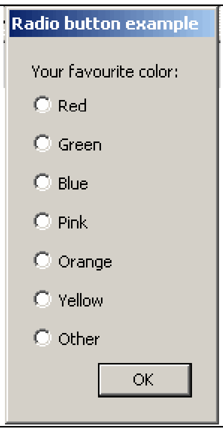
		
Digital 3D GUI		
3D rendering of 2D GUI		
2D GUI		

4.3.3.e Radio Button

Definition : Enables a Boolean choice by checking a circle aside of a label. An optionButton may be differentiated from a checkBox by the fact that when grouped optionButton selection is mutually exclusive while checkBox allows multiple choices. [USIX06]

Abstract attributes:

defaultState : Indicates a default state for a toggleButton := TRUE FALSE [USIX06]
groupName : Is the name of the := String [USIX06]

Radio Button		
	Immersive	Non-Immersive
Pure Reality		 SURNAME INITIALS SUBJECT CODE QUT QUEENSLAND UNIVERSITY OF TECHNOLOGY TEST ANSWER SHEET DO NOT WRITE ON THE BACK OF THIS SHEET STUDENT NUMBER CODES MARKING INSTRUCTIONS Multiple choice exams are a good real representation where we have to chose just one option between different possibilities
Augmented reality		
Augmented Virtuality		
Virtual 3D GUI		
Digital 3D GUI		
3D rendering of 2D GUI		
2D GUI		 Radio button example Your favourite color: Red Green Blue Pink Orange Yellow Other OK

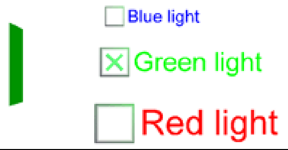
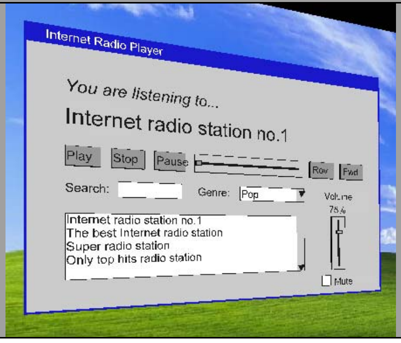
4.3.3.f Check Box

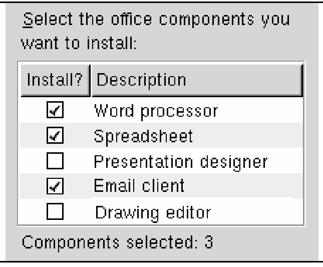
Definition : Enables a boolean choice by checking a square box aside of a label. A checkBox may be differentiated from a radio button (**optionButton)** by the fact that when grouped checkBoxes allow multiple choices while (**optionButton)** selection is mutually exclusive. **[USIX06]**.

Abstract attributes:

defaultState : Indicates a default state for a check box := TRUE FALSE **[USIX06]**

groupName : Is the name of the group := String **[USIX06]**

Check Box		
	Immersive	Non-Immersive
Pure Reality		A checklist in a inventory of a factory
Augmented reality		
Augmented Virtuality		
Virtual 3D GUI		
Digital 3D GUI		
3D rendering of 2D GUI	 Internet Radio Player Play Stop Pause Rev Fwd Search: Genre: Pop Volume 75.4 Mute You are listening to... Internet radio station no.1 Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station	

2D GUI		

Select the office components you want to install:

Install?	Description
☑	Word processor
☑	Spreadsheet
☐	Presentation designer
☑	Email client
☐	Drawing editor

Components selected: 3

4.3.3.g Slider

Definition : Allows a selection of one or two integer value(s) over a set of ordered integers **[USIX06]**

Abstract attributes:

minValue : Is the lower bound of slider values := integer **[USIX06] idem Studierstube**

maxValue : Is the upper bound of slider value := integer **[USIX06] idem Studierstube**

step : is to precise intermediate slider values between minValue and maxValue := integer **[USIX06]**

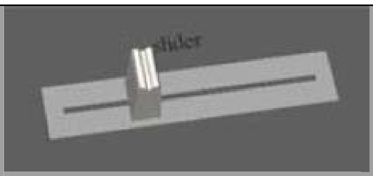
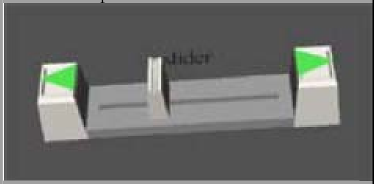
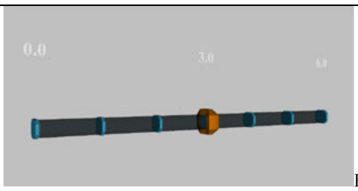
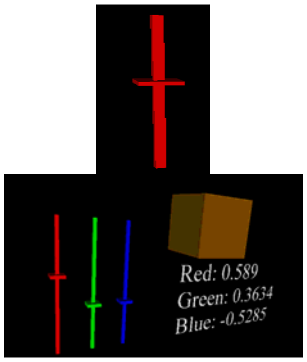
orientation : define the orientation of the slider := string equals vertical or horizontal **[USIX06]**

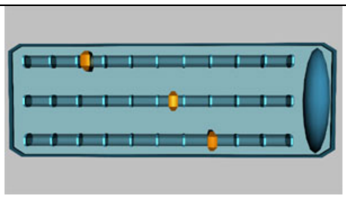
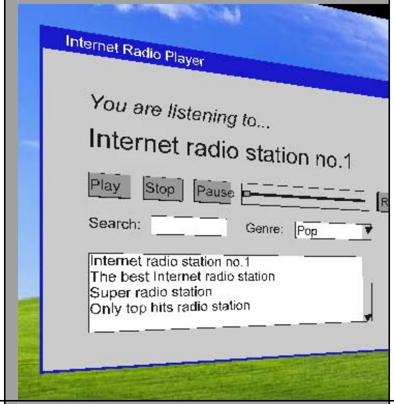
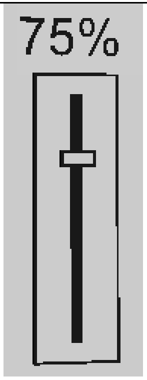
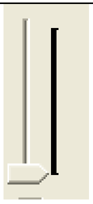
cursorPosition : Indicates a default value for a cursor, that allows to choose a value on a slider := integer [USIX06]

defaultContent : the text component string that defines the text displayed on the slider. **[USIX06]**

increment : define how much the increment or decrement buttons will move the slider := double value (Attribute in Studierstube)

Slider		
	Immersive	**Non-Immersive**
Pure Reality		

Augmented reality	 Simple slider in Studierstube  Incremental slider in Studierstube	
Augmented Virtuality		
Virtual 3D GUI		 Ho rizontal Vertical  Red: 0.589 Green: 0.3634 Blue: -0.5285

		
Digital 3D GUI		
3D rendering of 2D GUI	 Internet Radio Player You are listening to... Internet radio station no.1 Play Stop Pause Search: Genre: Pop Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station	
2D GUI		 Slider in Visual Basic

4.3.3.h ComboBox

Definition : Enables a direct selection over a collection of sequentially **ordered** items. **[USIX06]**

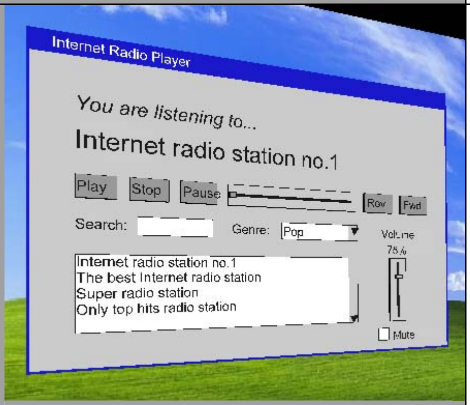
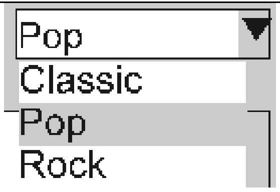
Abstract attributes:

isEditable : Specifies if the content of the textbox (composing a comboBox) is editable or not:= TRUE or FALSE **[USIX06]**

maxLineVisible : Indicates the number of visible lines := integer **[USIX06]**

items : Indicate de content of the combo Box := item object **[USIX06]**

ComboBox		
	Immersive	**Non-Immersive**
Pure Reality		

Augmented reality		
Augmented Virtuality		
Virtual 3D GUI		
Digital 3D GUI		
3D rendering of 2D GUI	 Internet Radio Player You are listening to... Internet radio station no.1 Play Stop Pause Rev Fwd Search: Genre: Pop Volume Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station Mute	
2D GUI		Combo Box

4.3.3.i Item

Definition : Specifies an item populating. [USIX06]

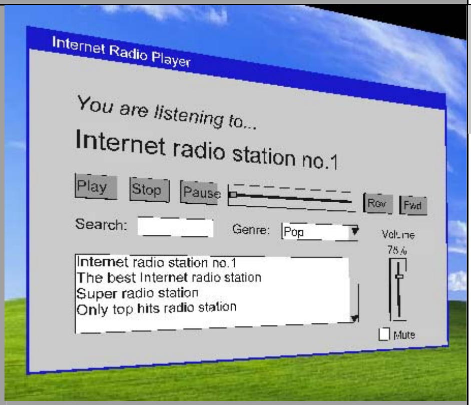
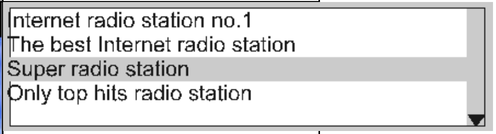
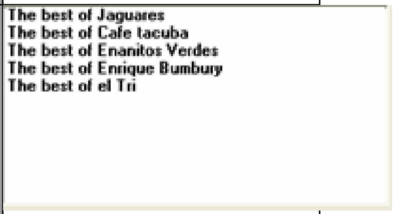
4.3.3.j ListBox

Definition : . [USIX06]

Abstract attributes:

isEditable : Specifies if the content of the listBox is editable or not:= TRUE or FALSE **[USIX06]**
maxLineVisible : Indicates the number of visible lines := integer **[USIX06]**
items : Indicate de content of the list Box := item object **[USIX06]**

List Box		
	Immersive	**Non-Immersive**
Pure Reality		
Augmented reality		

Augmented Virtuality		
Virtual 3D GUI		
Digital 3D GUI		
3D rendering of 2D GUI	 Internet Radio Player Play Stop Pause Rev Fwd Search: Genre: Pop Volume 78 % Mute You are listening to... Internet radio station no.1 Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station	 Internet radio station no.1 The best Internet radio station Super radio station Only top hits radio station ListBox
2D GUI		 The best of Jaguares The best of Cafe tacuba The best of Enanitos Verdes The best of Enrique Bumbury The best of el Tri List box in visual Basic

4.3.4 Context Model Review

A reviewed version of the model corresponding to the definition of the environmental model, which is responsible for describing the world in which any 3D UI could be rendered. In [Moli06] the improvement of the model is presented (Figure 4-27). The physical environment is expanded with surfaces (walls, the table, etc.). The interactive surface is the father of the hardware platform

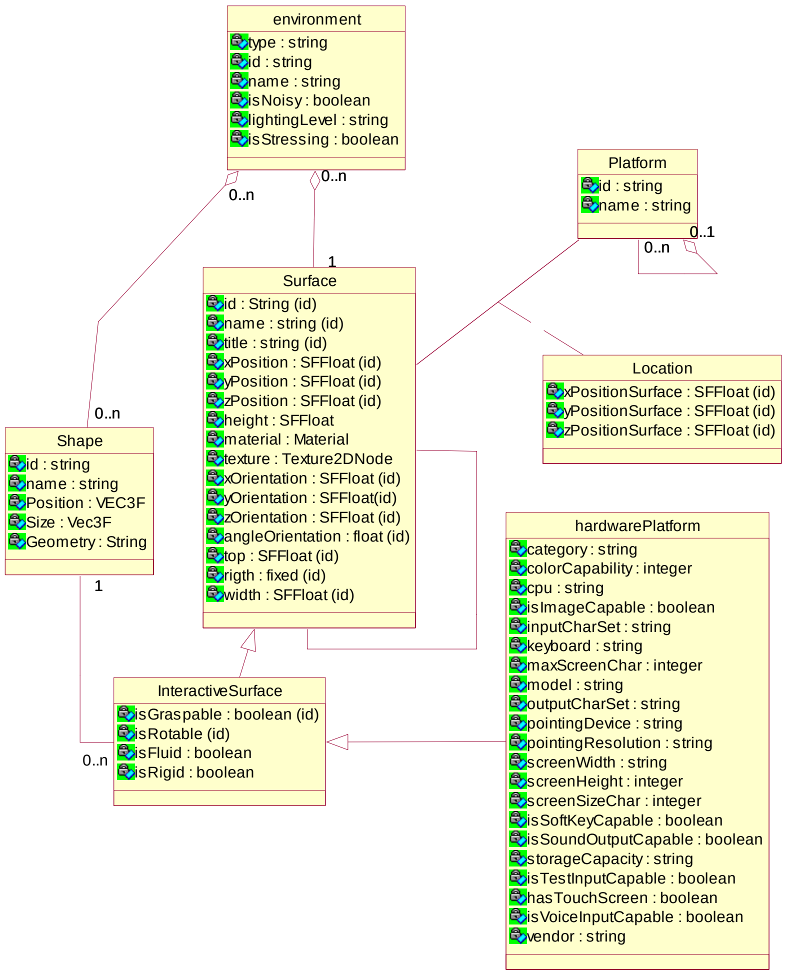


Figure 4-27 Environmental Model

However reviewed meta-model do not consider the separation of physical space and the same representation in the virtual space. It considers that the physical devices used for interacting with virtual reality applications, by using graspable, rotable, fluid or rigid surfaces. In our opinion, a separation of concerns must be used, to differentiate between virtual objects and physical ones, as the virtual environment can be the only visual context for the user when interacting to the UI. The required extension, Figure 4-28, considers a virtual environment that is aggregated by view points for the navigation of users, lights and virtual objects (SFNode). The physical environment corresponds to the described above in Figure 4-27, in which hardware, platforms, surfaces, etc, are part of it.

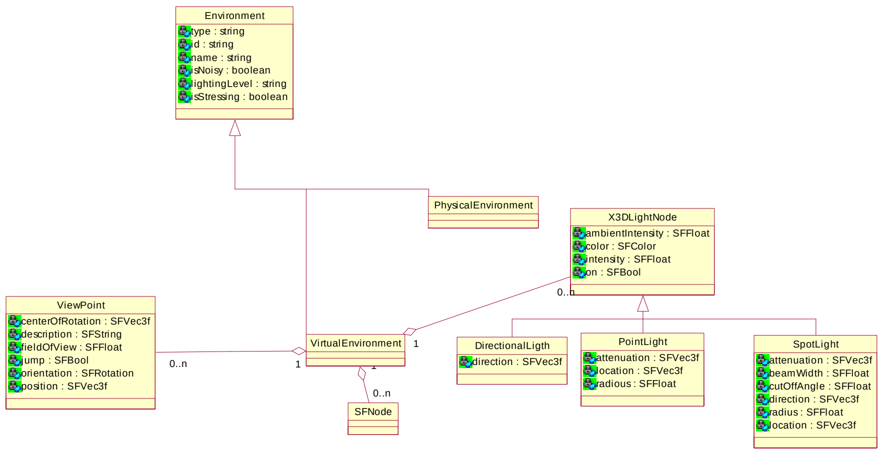


Figure 4-28 Environmental Model extension

4.4 Abstract Syntax: graphs as underlying formalism

The *abstract syntax* is defined as the hidden structure of a language, its mathematical background [Meye90]. Inspired in the mechanism followed by [Limb04c] and that is the basis of the development followed in [USIX06], this research uses as abstract syntax *"enriched" directed graph*. That is to say an identified, labeled, typed, constrained graph. A graph structure naturally describes a set of concepts and their relationships; it is strongly correlated to the concept of ontology [John01]. The graphs will define what to do with our ontology.

Graph structures are appropriate when the number of relationships among the concepts of an ontology become too large to represent them with another mathematical structure (e.g., lists, trees, sets). As argued by [Sowa92] graphs are logically precise, humanly readable (even that not necessarily understood), and computationally tractable. They have been used, for instance, to represent artifacts like code structures, system requirements, expert knowledge, causal systems, probabilistic systems, social structures [Limb04c]. We summarize the graph definitions in appendix B.

4.5 Concrete Syntax: a visual and textual syntax

A concrete syntax is an external appearance. Describing a concrete syntax something consists of describing formally the arrangements to describe it. As proposed in [USIX06] we use two types of syntax for methodology, one visual and another textual.

The **visual syntax** consists of boxes and arrows, a somewhat classic representation for a graphical structure. This visual syntax is mainly used to depict almost all the models defined in the ontology; there is a graphical representation for: the task model, the abstract models, graph declarations and transformation rules. The main diagrammatic characteristic so as the software tools that support them, are shown in table 4-5.

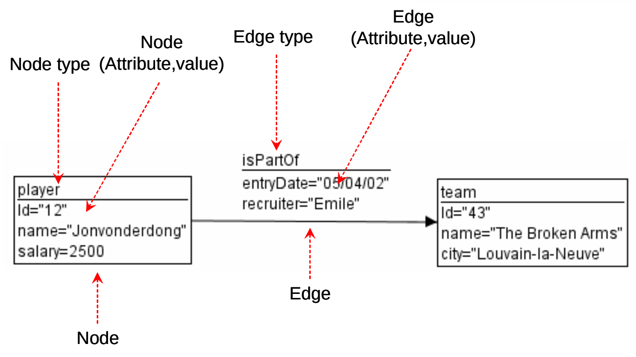
Model	Visual presentation	Tools [USIX06]
Task	**Tasks** Abstract, Interactive, System, User Cooperative **Task operators** [] |=| ||| [[ ]] [> |> >> []>> [T] T* Choice, order independency, concurrent, concurrent with information exchange, disabling, suspend | resume, enabling, enabling with information passing, optional task, iterative task.	IdealXML. Model editor: Task & Domain, AUI, inter-model relationships
Domain	▤ ⇡ ⇧ ↑ ◇— ◆— Class, dependency, generalization, association, aggregation, composition.	
Abstract	Container, component. Components facets input output navigation control	
Graphs	 Node type Node (Attribute,value) Edge type Edge (Attribute,value) player Id="12" name="Jonvonderdong" salary=2500 isPartOf entryDate="03/04/02" recruiter="Emile" team Id="43" name="The Broken Arms" city="Louvain-la-Neuve" Edge Node	AGG Transformation General purpose tool for graph transformation
Transformational Rules	Similar notation as for graphs. The set of rules is in appendix C	
Concrete	The concrete user interface does not have actually a graphical representation.	Maya Editor Alice Editor VRML Java 3D
Final User Interface		

Table 4-6 Tools to support our approach

Similarly, the **textual syntax** is described using an XML-based language, called USer Interface XML (UsiXML). This User Interface Description Language (UIDL) was chosen, among other reasons, because it allows describing the complete development cycle described in this dissertation. Another important reason is the context of such approaches that differs from our needs. From [Guer06], extended with [Cupp04], the following table 4-6 summarizes some common UIDLs.

Criteria / Language	Models	Tool support	Available	Context	Extensible	3D Description Support
XIML	Abstract Concrete Task and domain	Yes	No	Any	Yes	No
UIML	Abstract Concrete Task and domain	Yes	+/-	Any	Yes	No
XUL	Abstract Concrete	Yes	No	Web content Browser	No	No
AUIML	Abstract Concrete		No		Yes	No
UsiXML	Abstract Concrete Task and domain	Yes	Yes	Any	Yes	Partial
VRIXML	Concrete	Yes	No	Haptic interaction	Yes	Yes
InTML	Dialog	No	Yes	Interaction techniques	Yes	Yes

Table 4-7 User Interface Description Languages Comparison

From the table we summarize the main problems identified in the UIDLs. XIML [Puer02] is available via a non-commercial research license. UIML [Abra99], as pointed in [Cupp04] is to large and very abstract, the effort required to extend it to 3D is considerable and not compatible in a XML based complaint, UIML extensions are towards mappings to the FUI and not to the CUI. XUL [Bosw02] is specific for web content. AUIML [Azev00] among other problems it do not cover the entire path that we need, secondly, apart from the traditional UIDLs, do not define the interface from its appearance but from user interactions. VRIXML [Cupp04] and INTML [Figu02] are so specific, to the concrete presentation of the UI and to the interaction techniques, respectively.

Due to the availability of **UsiXML** and all the advantages that it has, we select to be our textual syntax. This UIDL is characterized by the following principles:

- *Expressiveness* of UI: any UI is expressed depending on the context of use thanks to a suite of models that are analyzable, editable, and manipulable by a software agent.
- *Central storage of models*: each model is stored in a model repository where all UI models are expressed similarly.
- *Transformational approach*: each model stored in the model repository may be subject to one or many transformations supporting various development steps. Each transformation is itself specified thanks to UsiXML [USIX06].

Also, UsiXML is able to specify various UIs with the five modalities of interaction. For this purpose, UsiXML is structured according to four basic levels of abstractions defined by the Cameleon reference framework.

4.6 Mappings

So far, we have introduced concepts for the syntactic representation 3D UIs (Ontology) and a way to group them (taxonomy); the stylistics: textual (UsiXML) and graphical (AGG, IdealXML). Also we briefly introduce the mapping from objects from the ontology to directed graphs with direct mapping, i.e. a task is mapped to a task node, and task attributes are mapped to node attributes. However, there is still a lack to define the model-based development for 3D UIs, until now just the models and its representations have been introduced, in this section we describe the transformational process, the semantic level, required to pass from model to model.

As stated in section 4-2, we rely in the Chameleon reference framework introduced by [Calv03]. This framework defines transitions between different models. Theses transitions are called *development steps* (each occurrence of a numbered arrow of Figure 4-29). The transformational process transforms an instance of a source model into another instance of a target model where source and target models types are directly adjacent (this is the desired path but theoretically if appropriate rules are defined transformations between non adjacent models are possible) in the development process.

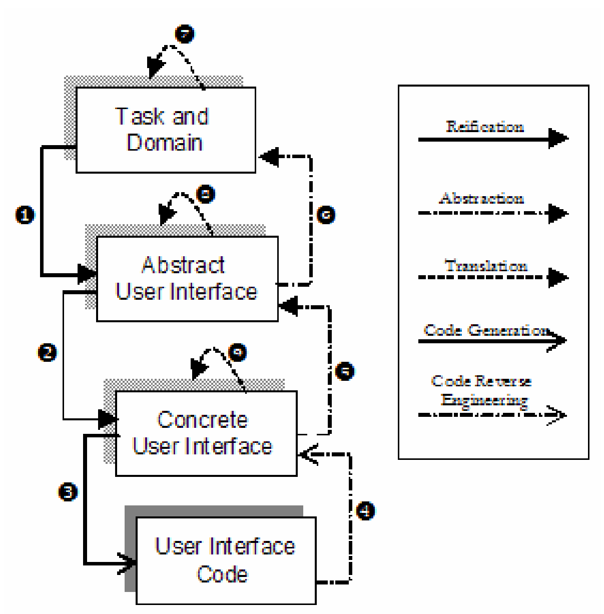


Figure 4-29 Transformation between viewpoints (left, mid.) & chapter reading map (right)

From [Limb04c] the definition of the development steps as follows:

- **Reification** (1,2 in Figure 4-29) is a transformation of a high-level requirement into a form that is appropriate for low-level analysis or design.
- **Abstraction** (5,6 in Figure 4-29) is a transformation of a low level specification into a high-level specification
- **Translation** (7,8,9 in Figure 4-29) is a transformation of a UI specification to adapt this specification to the constraints imposed by a new context of use. The context of use is defined after [Thev01] as a triple of the form *(e, p, u)* where *e* is a possible or actual environments considered for a software system, *p* is a possible or actual target platform, *u* is a user stereotype.
- **Code generation** (3 in Figure 4-29) is a process of transforming a concrete UI model into a compilable or interpretable code.
- **Code reverse engineering** (4 in Figure 4-29) is the inverse process of code generation i.e., it retrieves a concrete UI specification from a coded artifact.

Different types of *development paths* have been identified in [USIX06]:

- **Forward engineering (**or **requirement derivation)** is "the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation" [Chik90, Byrne92]. In this dissertation forward engineering can be viewed as a composition of *reifications and code generation* enabling a transformation of a high-level viewpoint into a lower level viewpoint.
- **Reverse engineering** is "the process of analyzing a subject system to (i)identify the system's components and their interrelationships and (ii)create representations of the system in another form or a higher level of abstraction" [Chik90, Byrne92]. In this dissertation reverse engineering can be seen as a composition of *abstractions* and *code reverse engineering* enabling a transformation of a low-level viewpoint into a higher level viewpoint.
- **Context (of use) adaptation** is the process of adapting a UI specification for another context from the one it was designed for. Context adaptation can be obtained from a *translation* of a UI model at any level.

Other development paths like:

- **Retargeting.** This transition is useful in processes where an existing system should be retargeted, that is migrated from one source computing platform to another target computing platform that poses different constraints. Retargeting can be composition of reverse engineering, context adaptation and forward engineering. In other words a UI code is abstracted away into a CUI (or an AUI). This CUI (or AUI) is reshuffled according to specific adaptation heuristics. From this reshuffled CUI (or AUI) a new interface code is created along a forward engineering process.

- **Middle-Out development** is a term coined by [Luo95]. It refers to a situation where a developer starts a development by a specification of the UI (no task or concept specification is priory built). Several contributions have shown that, in reality, a development cycle is rarely sequential and even rarely begins by a task and domain specification. The literature in rapid prototyping converges with similar observations. Middle-out development shows a development path starting in the middle of the development cycle e.g., by the creation of a CUI or AUI model. After several iteration at this level (more likely until customer's satisfaction is reached) a specification is reverse engineered. From this specification the forward engineering path is followed.

- **Leapfrog development** refers to the situation where an intermediary viewpoint is bypassed in the transformation process. In our framework for instance, it might not be needed to define an AUI if only one modality is targeted.

Development steps may be decomposed into *development sub-steps*. Some of these activities have been identified by [Luo95]. It can consist, for instance, of the selection of concrete interaction objects, the definition of the navigation, the definition of the container structure. In appendix C the set of sub-steps associated to each development step are described. All of them described in terms of directed graph transformational rules. In this section we show the required rules for such transformation from the concrete model to the abstract model. As the definition of development sub-steps may depend on the designer's practice, the organization rules, the type of artifact that is built, etc, there are not good, bad, better or worse rules, just appropriate ones for our purposes.

4.6.1 An introduction to graph grammars

Graph grammars provide us with an intuitive formalism for manipulating graph structures. A graph grammar is a set of graph rewriting rules (called in this work *graph transformation rules*), a graph to transform (called *host graph* or *initial graph*) and a set of parameters (called *embed*) defining how to apply the rules on the host graph.

Three are the components of graph grammars, assuming that the have a graph source *s*, the result of applying such transformational rules (graph grammars) will end on a target graph *t* that corresponds to the desired resulting model. The transformational rules are composed of three elements:

[1] Left hand side (LHS). It expresses a graph pattern that, if it matches in the source graph *s*, it will modify the target graph *t,* where the patter is matched. A LHS may be seen as a condition under which a transformation rule is applicable.

[2] Right hand side (RHS). Express how the target graph *t* will change when the condition LHS is matched.

[3] Negative application pre-conditions (NAP), also called forbidden contexts, are assertions that have to hold false before the application of a rule. Indeed this group of graph avoids the infinite repetition of graph transformation, or the execution of the transformation on a specific instance.

A complete description of graph definitions can be found in appendix B. The important components for the transformational rules are described above.

Graph transformations are used to perform model-to-model transformations i.e., reifications, abstractions and translations (see Figure 4-29). We transform a UI specification with a set of transformation rules taken from a transformation catalog (for 2D GUIs see the catalog in appendix C). In the next section through a case study, some of the transformational rules are shown. All examples use the graphical formalism of the graph transformation tool AGG [Ehri99] presented hereafter. Graph grammars provide us with an intuitive formalism for manipulating graph structures. A graph grammar is a set of graph rewriting rules (called in this work graph transformation rules), a graph to transform (called host graph or initial graph) and a set of parameters (called embed) defining how to apply the rules on the host graph.

4.7 Method for developing 3d user: Architecture continuum

Several tools have been exploited or developed in the context of [USIX06]. They all play a certain role in making model-based development (MDE) a reality. They were briefly showed in table 4-5. There is still an extension to the actual effort to support 3D development. Since the method should be compliant with MDE and its principle of separation of concerns, the method (see Figure 4-30) is itself decomposed into a sequence of four steps.

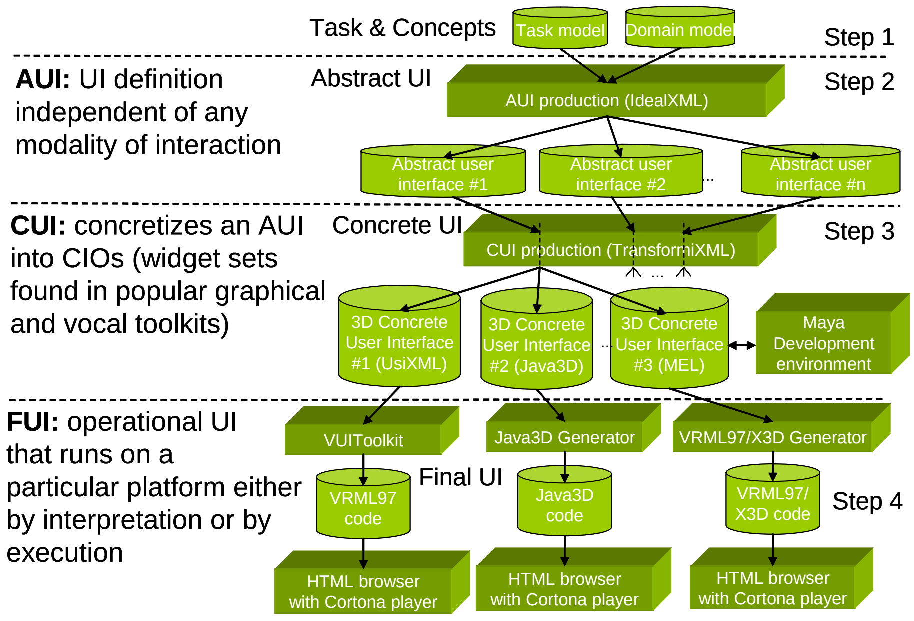

Task & Concepts
Task model
Domain model
Step 1
AUI: UI definition independent of any modality of interaction
Abstract UI
AUI production (IdealXML)
Step 2
Abstract user interface #1
Abstract user interface #2
...
Abstract user interface #n
CUI: concretizes an AUI into CIOs (widget sets found in popular graphical and vocal toolkits)
Concrete UI
CUI production (TransformiXML)
Step 3
...
3D Concrete User Interface #1 (UsiXML)
3D Concrete User Interface #2 (Java3D)
3D Concrete User Interface #3 (MEL)
Maya Development environment
FUI: operational UI that runs on a particular platform either by interpretation or by execution
VUIToolkit
Java3D Generator
VRML97/X3D Generator
Final UI
VRML97 code
Java3D code
VRML97/ X3D code
Step 4
HTML browser with Cortona player
HTML browser with Cortona player
HTML browser with Cortona player

Figure 4-30 Outline of the method for developing 3D user interfaces

From step one to three there are two tools used to design task and concepts (domain model) and generate its UsiXML corresponding code, using IdealXML [Mont05]. Similarly, IdealXML is used to define the concrete user interface in an editor. Ideally this transformation should be done automatically through the application of transformation rules to graph definitions for task and concepts. This step is under development, the transformation engine for graphs just works manually, using AGG editor. For this reason the building block for TransformiXML is still undone. Even so, as stated before, AGG graph can be

used to test the feasibility of the approach and the test of the transformational rules.
In step three, our need is on a software tool that could support the design and manipulation of 3D UIs, for this purpose we proposed the use on Maya software, which is a very powerful commercial editor for 3D content. A concrete UI can be exported from its definition in UsiXML and imported to Maya, or any other toolkit, Anark, Max 3D, or a non-commercial, such as Blender. In the review of the state of the art we identify that the two most advanced and extended toolkits are Max 3D and Maya, as they have too many plug-ins for importing / exporting in several formats, some of them of our interest for web content, such as VRML and X3D.

Actually we draw our 3D UIs in Maya then we export them to VRML and X3D (the plug-in produces low results for complex interfaces). Then using an editor we add the behavior to the interfaces. This is a tedious task, as you will see in next chapter, our example has more than 30 thousand code lines, and it is difficult to identify shapes and add behavior in VRML. As our CUI model is based is the abstract of X3D, an XSLT transformation can be applied to our UsiXML description of the CUI model and pass it to X3D, then this definition can be imported in a high level editor, such as Maya. The designer will be able then to organize the user interface is different position, adjust the size of shapes, change the colors, etc.

A second high level toolkit that we used to model the UIs is Alice that generates Java 3D content. Import and export to Alice is still a job undone. We need to explore how the Java 3D API is structured and how Alice format is structured. The advantage of Alice is that they do not just have primitive objects but also a Gallery of predefined objects. Unfortunately, Alice have a reduce set of objects dedicated to the UI control, button, switch, text and no more.

Finally, the last toolkit that we used to generate the FUI is the VUIToolkit [Moli05]. This library of widgets does not have neither an editor to design the 3D UI, nor an automatic generation from the concrete model to the FUI in VUI components.

4.8 Conclusion

In this chapter, a model-based development method based on graph transformation has been introduced, defined and illustrated. The method extends an already defined methodology [USIX06] that considers the development of UIs in 2D and vocal. We propose an extension of such approach that implies new features at all levels of the methodology.

The ontology was extended in all the levels

- Task Model, new action task definitions were added. Some task patterns for the most common User task in virtual environments.
- Concrete Model, the definition of new classes for concrete interaction objects, dividing the actual definition in 2D CIOs and 3D CIOs. The specification of a meta-model for 3D UIs.
- Context model, a new environmental model was introduced accordingly not just to the physical space but to the virtual space also.

The mapping to achieve an Abstract model to 3D Concrete model transformation was also introduced. The software tool requirements for doing such transformation automatically were discussed. In next chapter we will prove the feasibility of following our method.

Chapter 5 Case Studies

5.1 Introduction

This chapter applies multi-path development of user interface to two different case studies. The two cases are progressive in terms of complexity. Their presentation relies on a series of illustrations showing how artifacts are progressively transformed according to various development sub-steps, steps, and paths.

The process adopted to develop the case studies of this chapter consists of: (1) Building initial models. Such models have been edited with their associated editing tool. For instance, IdealXML [Mont04] has been used to edit the task and domain model. Most of the rules have been elicited prior to realizing these case studies by a theoretical analysis of development sub-steps as illustrated in Chapter 4. (2) Exporting resulting models to UsiXML and illustration.

The first case study is devoted to the development of an opinion polling system, a reasonable scaled example of a typical information system. A forward engineering path is applied that starts from the definition of the task and domain models to produce both an AUI and a CUI. The CUI is reshuffled by hand in Maya editor; these modifications do not change the AUI.

5.2 Study Case 1: The virtual polling system

This case study is devoted to the development of an opinion polling system, a reasonable scaled example of a typical information system. The development scenario is the following: a forward engineering path is applied from a definition of the task and domain viewpoint to produce both an AUI and a CUI.

5.2.1 Step 1: The task and domain models

The task model, the domain model, and the mappings between, are all graphically described using IdealXML tool [Mont05], an Interface Development Environment for Applications specified in UsiXML. Figure 5-1 depicts the domain model of our UI as produced by a software engineer. A participant participates to a questionnaire. A questionnaire is made of several questions. A question is attached to a series of answers. The domain model has the appearance of a class diagram.

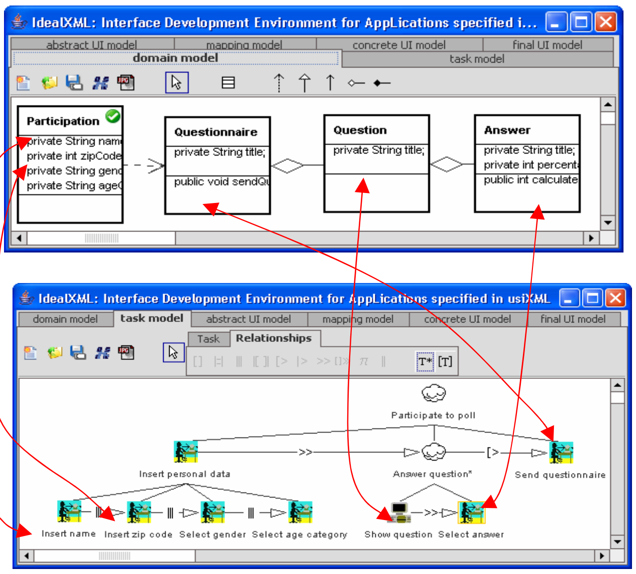
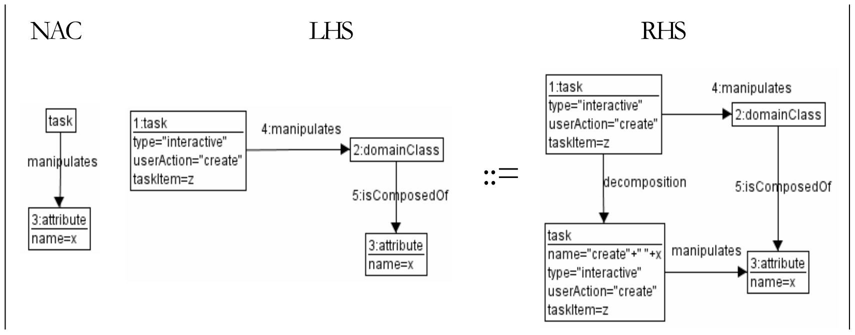


Figure 5-1 Mapping between domain concepts and task model

The Figure 5-1 illustrates a CTT representation of the task model envisioned for the future system. The root task consists of participating to an opinion poll. In order to do this, the user has to provide the system with personal data. After that, the user iteratively answers some questions. Answering a question is composed of a system task showing the title of the question and of an interactive task consisting in selecting one answer among several proposed ones. Once the questions are answered, the questionnaire is sent back to its initiator. All temporal relationships are enabling which means that the source task has to terminate before the target task can be initiated.

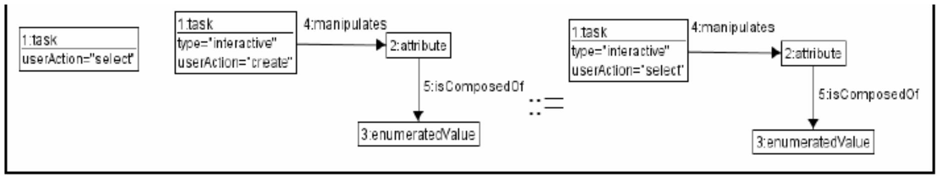
The dashed arrows between the two models in Figure 5-1 depict the model mappings, such as **manipulates** relationships between the task and the domain model as dashed arrows. **Provide Personal Data** is mapped onto **Participant** class. Show **Question** is mapped onto the attribute title of class **Question**. The task **Select Answer** is mapped onto the attribute **title** of the class **Answer**. Finally, the task **Send Questionnaire** is mapped onto the method **sendQuestionnaire** of the class **Questionnaire.** The initial task may be considered as not precise enough to perform transformations. Indeed, the task **Provide Personal Data** is an interactive task consisting in creating instances of **Participant**. In reality, this task will consist in providing a value for each attribute of Participant. This could mean that the task model is not detailed up to the required level of decomposition.

Rule 5-1 is applied to the task and domain models. The Left-Hand Side (LHS) contains an **interactive** task (1) where the user action required to perform the task is of type **create**. This task manipulates a class from the domain model (2), which is composed, of an attribute that takes the value of a variable **x**. The Negative Application Condition (NAC) specifies that a task manipulates an attribute (3) whose name is stored in the same variable **x**. The Right Hand Side (RHS) specifies the decomposition of the task described in LHS (1) into an interactive task (2), which requires a user action of type **create**. Note the way they are named using a post-condition on their **name** attribute. The mappings between nodes and between edges belonging to the three components of a rule (NAC, LHS, RHS) are specified by attached numbers. The application of this rule on the task and domain model represented in the form of a graph G is the following: when the LHS matches into G and the NAC does not match into G, the LHS is replaced by the RHS, resulting a transformed graph G'. Therefore, Rule 5-1 decomposes the task **Provide Personal Data** into four new sub-tasks, each of them manipulating an attribute of class **Participant**.



Rule 5-1. Consolidation of the task model

Consequently, to the execution of this rule, four new tasks are created: create name, create zipCode, create ageCategory and create gender. Figure 5-1 shows the mapping model containing the mappings between the refined task model and the domain model of the opinion polling system. Each of the four new sub-tasks will be mapped on the corresponding attribute of the class **Participant**, the rest of the mappings remaining the same. Due to the fact that "create" is a very general action type and that both **ageCategory** and **gender** attributes hold an enumerated domain, "create" can be specialized into "select". Rule 5-2 is applied in order to achieve this goal. Rule 5-3 provides a default temporal relationship (set to enabling) when two sister tasks have no temporal relationship.



Rule 5-2. Specializing a user action.

5.2.2 Step 2: From The task and domain models to Abstract Model

The actual development path follwoed in [USIX06] consider a one-to-one mapping from model to model. However for this dissertation there is a need to clearly distinguish (see figure 5-2) between an abstract description for the 3D Rendering of 2D UIs, which is quite similar to the followed actually, and for a genuine 3D UI.

We have three different objective Concrete UIs, which are organized differently physically. In figure 5-2 we show the different paths (a and b) that we could follow. Due to the constraints that we could find in future transformation by keeping one single AUI and then from it derivate several CUIs, we select path b.

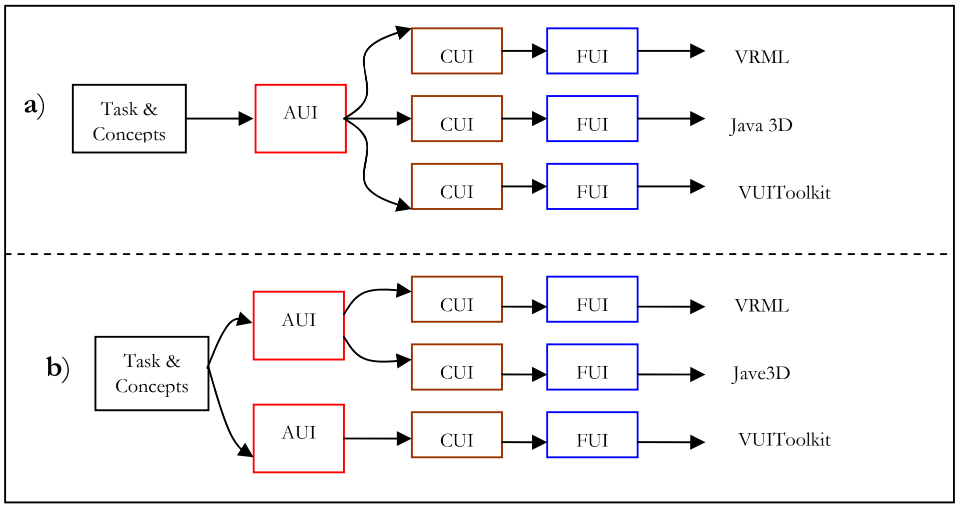
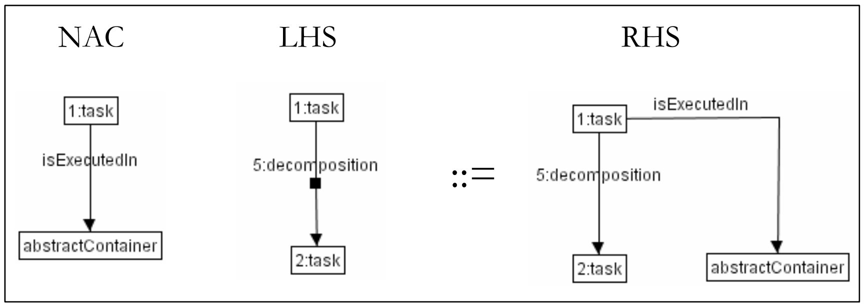


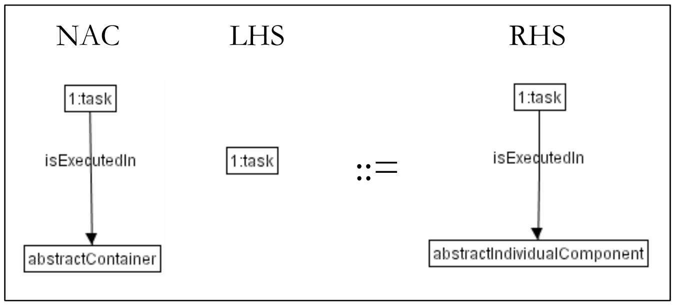
Figure 5-2 Development Paths

5.2.2.a Identification of abstract UI structure for AUI "A"

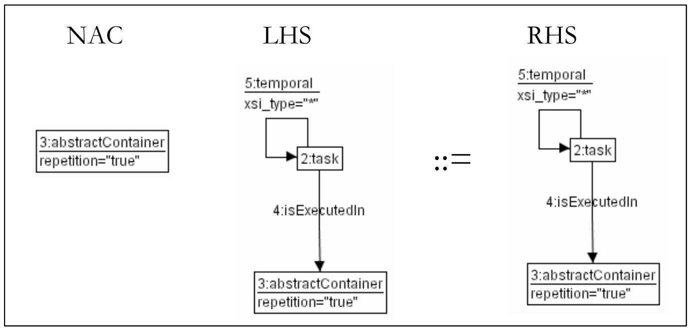
The identification of AUI, from [Limb04c] structure is ensured by applying Rule 5-3, Rule 5-4, Rule 5-5, Rule 5-6, and Rule 5-7. These rules essentially recreate the task model structure by a hierarchical decomposition of abstract containers and abstract individual components.



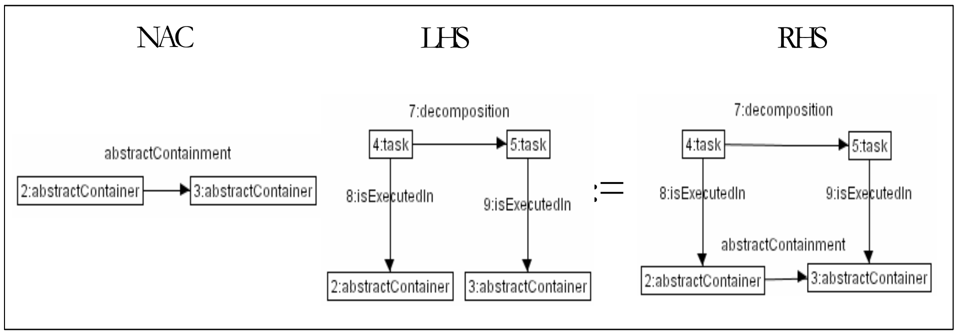
Rule 5-3 Create an AC for task that has task children



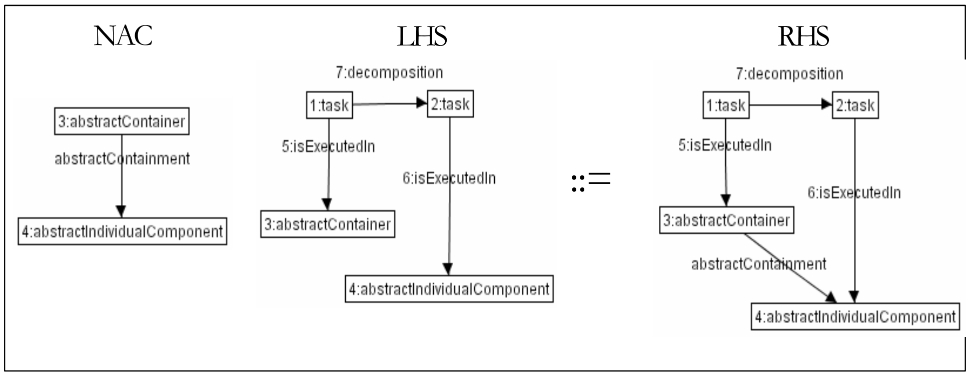
Rule 5-4 Create an AIC for leaf tasks



Rule 5-5 Iterative tasks are mapped onto repetitive AC



Rule 5-6 Reconstruct containment relationships between AC

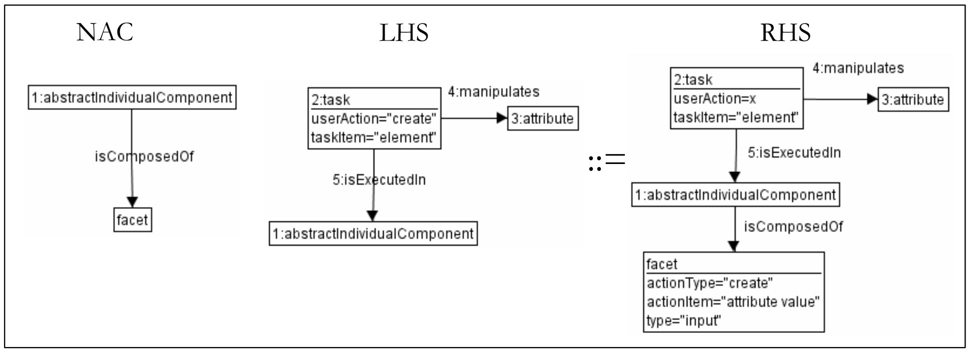


Rule 5-7 Reconstruct containment relationships between AIC

5.2.2.b Selection of AIC

Each AIC can be equipped with facets describing its main purpose/functionality. As explained in Chapter 4, these facets are derived from the combination of the task model, the domain model, and the mappings between them. The mappings between the task and the domain models have been described above. We illustrate some of the rules applicable to the present case study. From these mappings it can be derived that:

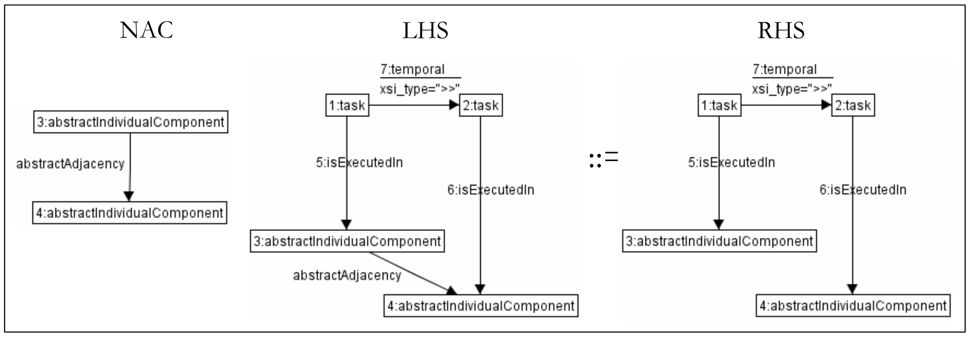
- AICs create name and create zipCode are equipped with an input facet of type "create attribute value".
- AICs select sex and select ageCategory are equipped with an input facet of type "select attribute value". The enumerated values associated to the attribute are transferred as selection value of the facet from the domain model.
- AIC Show Question is equipped with an output facet of type "output attribute value" (i.e., the question title).
- AIC Select Answer is equipped with an input facet of type "select attribute value". It is also set to repetitive as the amount of instances of answer is only known at run-time: no enumerated values are provided nor attribute instances.
- AIC Send Questionnaire is equipped with a facet control that references the name of the method on which it is associated, here sendQuestionnaire



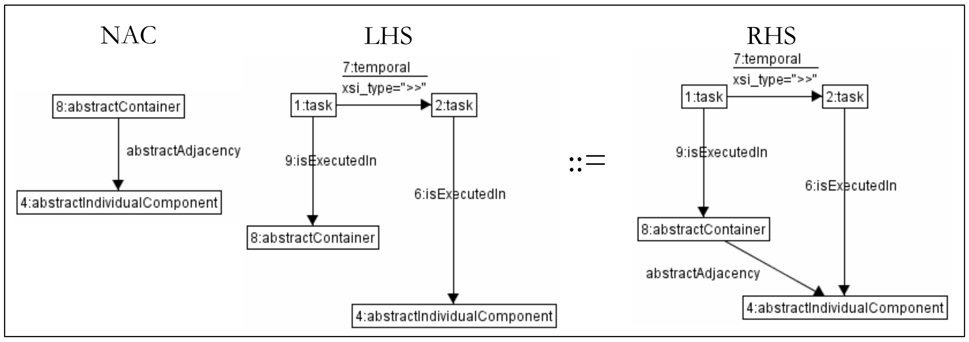
Rule 5-8 Create an input facet to AICs that realize creation tasks

5.2.2.c Spatio-Temporal arrangement of abstract interaction objects

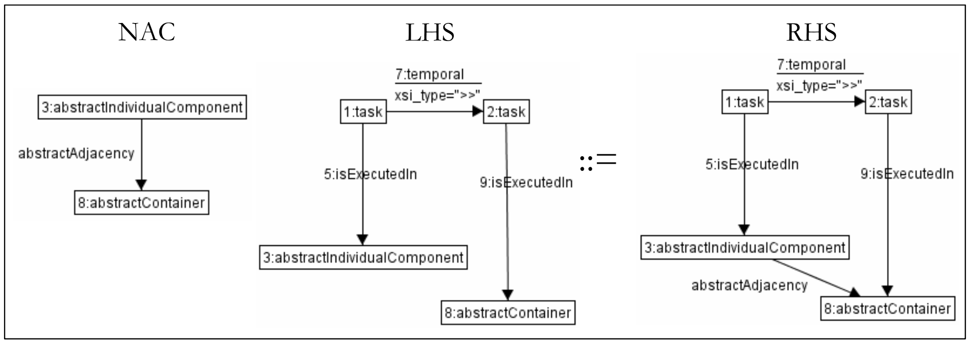
We apply Rule 5-6 (reproduced as Rule 5-9Rule 5-), Rule 5-10, Rule 5-11, Rule 5-12. These rules reveal how implementing hierarchical rules in AGG could be repetitive: one rule should be introduced for each possible couple with AC and AIC as elements, that is a total of four rules.


NAC
LHS
RHS
3:abstractIndividualComponent
abstractAdjacency
4:abstractIndividualComponent
7:temporal
xsi_type=">>"
1:task
2:task
5:isExecutedIn
6:isExecutedIn
3:abstractIndividualComponent
abstractAdjacency
4:abstractIndividualComponent
::=
7:temporal
xsi_type=">>"
1:task
2:task
5:isExecutedIn
6:isExecutedIn
3:abstractIndividualComponent
4:abstractIndividualComponent

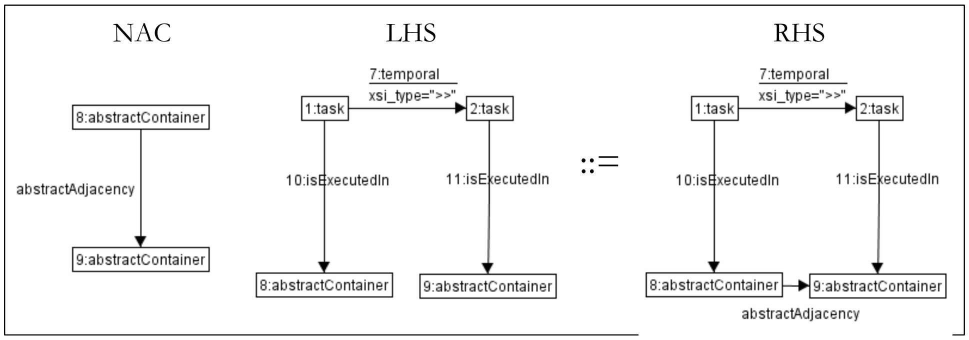
Rule 5-9 Deriving abstract adjacency for <AIC,AIC> couple



Rule 5-10 Deriving abstract adjacency for <AC,AIC> couple



Rule 5-11 Deriving abstract adjacency for <AIC,AC> couple



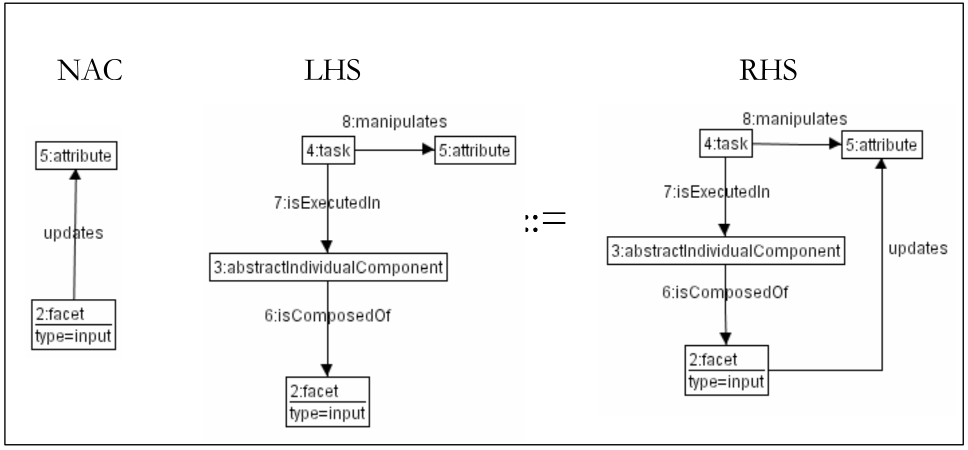
Rule 5-12 Deriving abstract adjacency for <AC,AC> couple

5.2.2.d Definition of abstract dialog control

We apply Rule 4-9 and the like to realize this sub-step. Similarly to the previous step, a rule is defined for each combination of couple with AC and AIC as elements.

5.2.2.e Derivation of AUI to domain mappings

Rule 4-10 is one of the rules applied in this sub-step. Rule 5-15 is another rule that is applicable to our case. It creates an updates relationship between the input facet of an AIC and the attribute manipulated by its associated task.



Rule 5-13 Derivation of the updates relationship for an input facet

5.2.2.f Resulting specification

Following the same mechanism of rule transformation, an abstract individual component (AIC) is created for every leaf task found the task model, insert name, insert zip code, select gender, select age category, show question, select answer and send questionnaire. Each AIC can be equipped with facets describing its main purpose/functionality. These facets are derived from the combination of task model, domain model and the mappings between them. Task definitions have information that is relevant for the mappings, such as: userAction, which could be: create, delete, modify, among others. According to these mappings it can be derived that AICs create name and create zipCode are equipped with an input facet of type "create attribute value". The generated abstract user interface is shown in Figure 5-3.

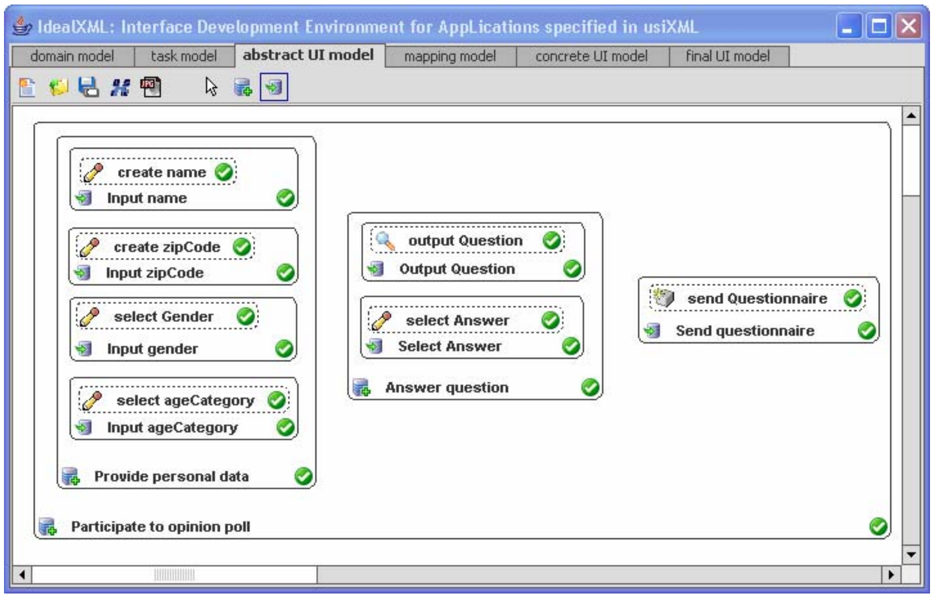
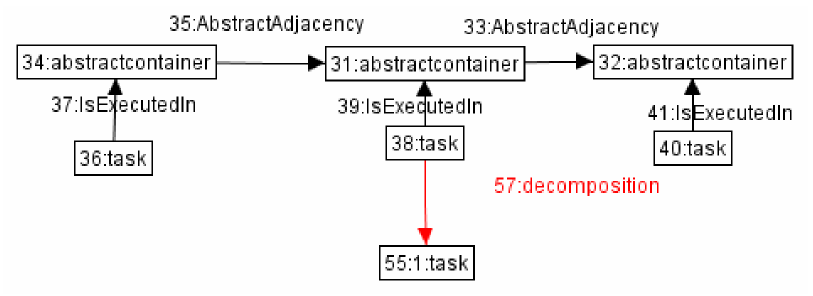


Figure 5-3 IdealXML Mapping from Task and Domain model to Abstract Model A

5.2.2.g Identification of abstract UI structure for AUI "B"

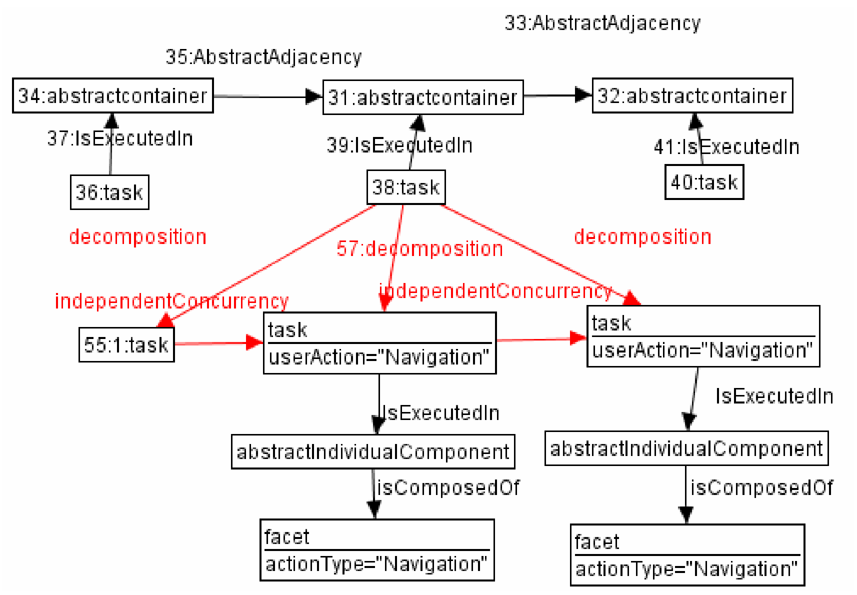
Two new rules are required to generate a different version for the In 3d there is clearly a new for navigation between containers, as our approach considers, we propose rule 5-14 to create abstract individual components for each abstract container, with navigation facet on it.

We need to consider three cases for the navigation the containers in the middle that could need forward and backward navigation, the last one will need just one navigation and the first one is a special case, that we propose will need both navigations, so as the introduction of a new task that will start the application. The LHS for intermediate tasks is below, notice that we are not interested in the relation between them but to the fact that their correspondent containers are related, this means that that are related in some way. If so, and a task is in the middle as task 8 The RHS rule extends the source graph by adding the task facets, adjacencies, etc.



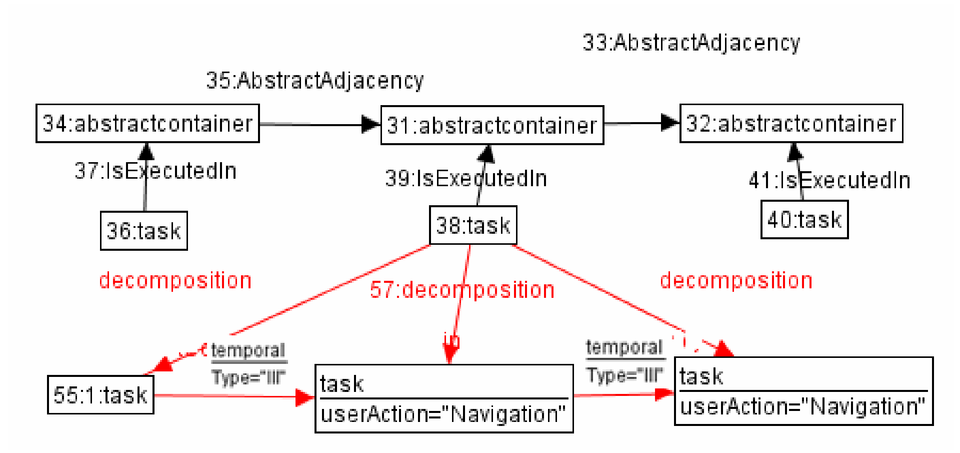
LHS Rule for creating navigation facet for abstract containers

The RHS rule considers also, the connection between the new tasks and any of the subtask of task (38), depicted as task (55:1) in the rule, AGG engine will match the first task non deterministically. The RHS will connect the new navigation tasks to the task (55:1).



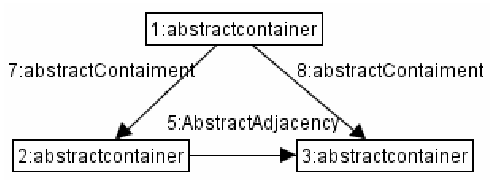
RHS Rule for creating navigation facet for abstract containers

The NAC for this transformation rule, points to avoid the repetition of the same transformation. Below the NAC in which we avoid the repetition of such transformation when the same pattern of transformation is matched.



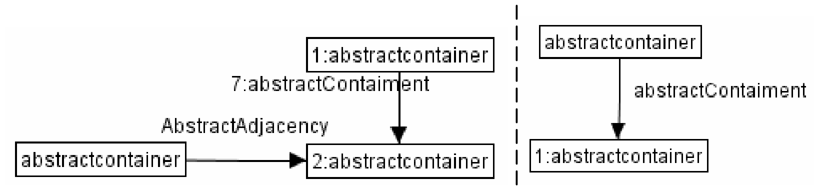
NAC for creating navigation facet for abstract containers

Analogous to this step is the correspondent to the last container, with the difference just on the adding and comparing, we just need two abstract containers and an adjacency between them. Identifying not just the last but also the first requires the same LHS condition but different NAC. More clearly, the LHS rule below we show the requirement to identify either the first or the last abstract container in a task tree. When an abstract container (2) with abstract adjacency (5) to another abstract container (3) an related with an abstract containment (7) with an upper level abstract container.



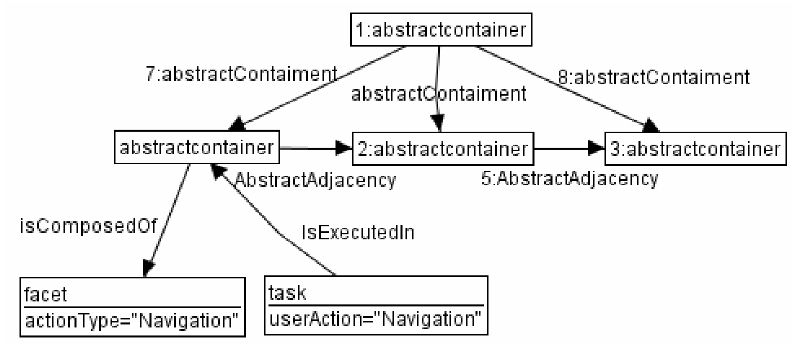
LHS Rule for creating navigation facet for the first abstract container

This initial state can be then compare to the left to check if there is no more abstract containers related to the abstract container (2). A second NAC required is to check if the abstract container (1) is the root of abstract containers.



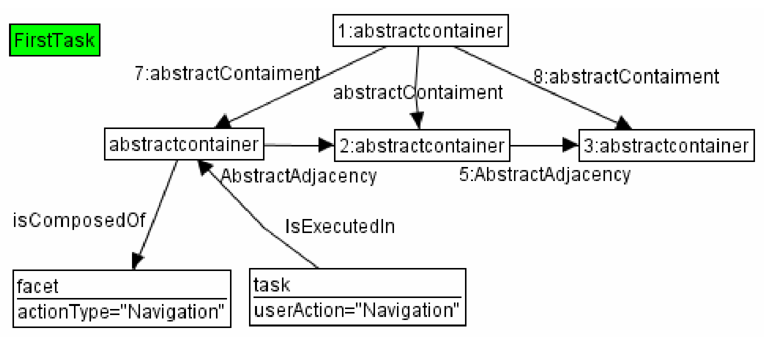
NAC for creating navigation facet for the first abstract containers

The RHS rule to generate the navigation component is as follows



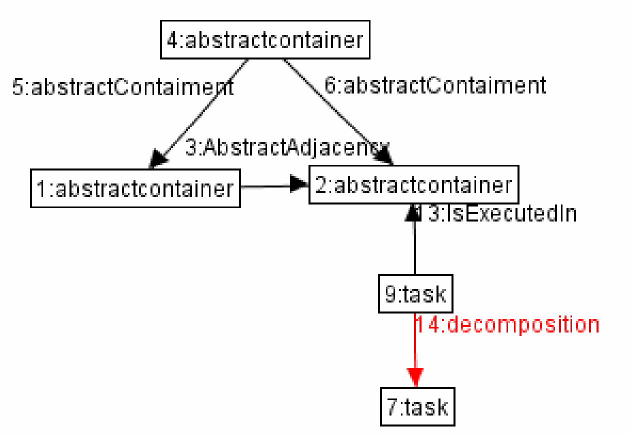
RHS Rule for creating navigation facet for the first abstract container

Notice that the creation of this rule requires the creation of a new NAC, if not, infinitely the system could generate abstract containers for the new task created. Flags can be generated for this purpose, dummy nodes that are not related to the system but just to control the flow can be added and/or removed. So the appropriate RHS is below. Finally the new NAC required is FirstTask.



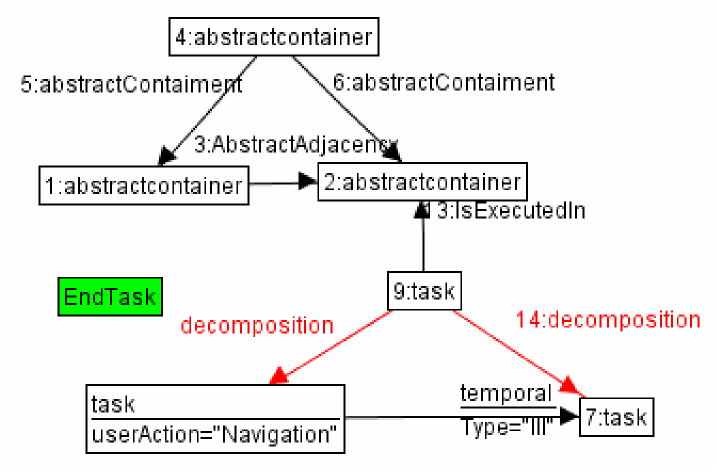
RHS Rule for creating navigation facet for the first abstract container

Analogous to the first task, to create navigation for the last task, we will need the almost the same LHS rule, but some differences are required, such as to identify the task that is executed in the abstract container (2) to connect it to the new navigation task.

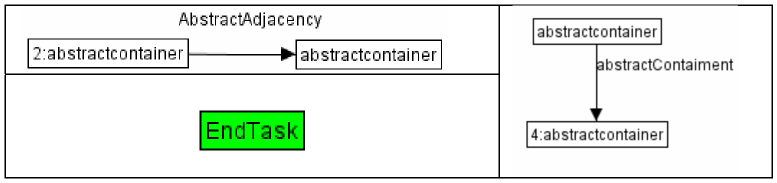


LHS Rule for creating navigation facet for the last abstract container

The RHS applied to the above LHS rule, we assume in this case that the last task is decomposed in at least two other subtasks, and we are just interested in linking our new navigation task to any sub-task (9). The three NAC are listed below.



RHS Rule for creating navigation facet for the last abstract container



NAC for creating navigation facet for the last abstract containers

The generated abstract user interface B is shown in Figure 5-4.

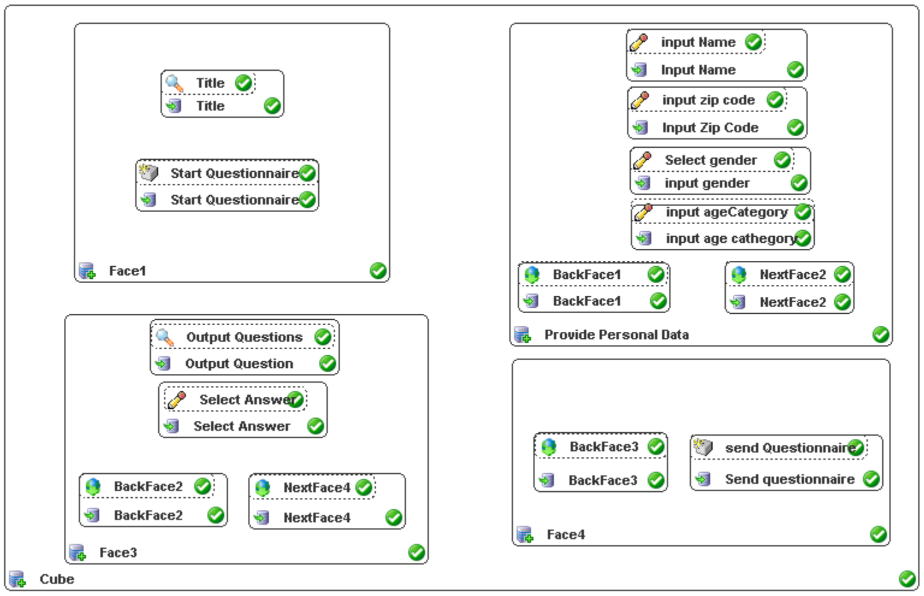


Figure 5-4 IdealXML Mapping from Task and Domain model to Abstract Model B

The corresponding UsiXML specifications, generated in IdealXML, correspond to the AIO decomposition.

```
<abstractContainer id="idao0" name="Cube" splittability="true">
<abstractContainer id="idao1" name="Face1">
    <abstractIndividualComponent id="idao5" name="Title">
        <output id="idao6" name="Title" outputContent="Welcome to the Virtual Polling system" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao7" name="Start Questionnaire">
        <control id="idao8" name="Start Questionnaire" actionType="interaction"
                event="startQuestionnaire " />
    </abstractIndividualComponent>
</abstractContainer>
<abstractContainer id="idao2" name="Provide Personal Data">
```

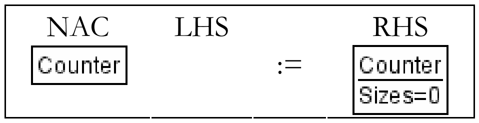
```
    <abstractIndividualComponent id="idao9" name="Input Zip Code">
        <input id="idao15" name="input zip code" actionType="interaction" dataType="String"
              attributeDomainCharacterization="zipCode" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao10" name="Input Name">
        <input id="idao14" name="input Name" actionType="interaction" dataType="String"
              attributeDomainCharacterization="name" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao11" name="input gender">
        <input id="idao16" name="Select gender" actionType="interaction" dataType="String"
              attributeDomainCharacterization="gender" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao12" name="input age cathegory">
        <input id="idao17" name="input ageCategory" actionType="interaction" dataType="String"
              attributeDomainCharacterization="ageCategory" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao13" name="BackFace1">
        <navigation id="idao18" name="BackFace1" actionType="interaction" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao19" name="NextFace2">
        <navigation id="idao20" name="NextFace2" actionType="interaction" />
    </abstractIndividualComponent>
</abstractContainer>
<abstractContainer id="idao3" name="Face3">
    <abstractIndividualComponent id="idao26" name="Output Question">
        <output id="idao29" name="Output Questions" actionType="interaction"
                outputContent="Questions" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao27" name="Select Answer">
        <input id="idao28" name="Select Answer" actionType="interaction" dataType="String"
              attributeDomainCharacterization="answer" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao31" name="BackFace2">
        <navigation id="idao34" name="BackFace2" actionType="interaction" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao32" name="NextFace4">
        <navigation id="idao33" name="NextFace4" actionType="interaction" />
    </abstractIndividualComponent>
</abstractContainer>
<abstractContainer id="idao4" name="Face4">
    <abstractIndividualComponent id="idao22" name="Send questionnaire">
        <control id="idao23" name="send Questionnaire" actionType="interaction"
                event="sendQuestionnaire" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao24" name="BackFace3">
        <navigation id="idao25" name="BackFace3" actionType="interaction" />
    </abstractIndividualComponent>
</abstractContainer>

</abstractContainer>
```

5.2.3 Step 3: From Abstract model to Concrete User Interface model

The third step implies a transformational system that is composed of necessary rules for realizing the transition from AUI to CUIs. For this purpose, other design rules could be encoded in UsiXML so as to transform the AUI into different CUI depending the options decided. Since the AUI model is a CIM, it is supposed to remain independent of any implementation. However, when it comes to transform this AUI into a corresponding CUI or several variants of it, platform concerns come into consideration. For this purpose, several design rules exist that transform the AUI into CUIs with different design options that will then be turned into final code when generated.

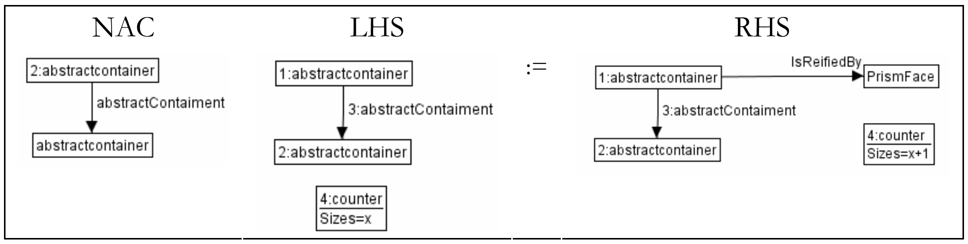
5.2.3.a Reification of AC into CC

The simple heuristic in 2D solving the current problem consists of representing all tasks into one single window. Each abstract container becomes a prism face or any surface but the top level could become the virtual environment (used for the variant in which objects will be floating in the space) or any other container such a prisms (the quantity of sides for the prisms would be based on the sublevel of containers required). Each abstract container at level "leaf-l" is transformed into a prism face. The main container is mapped to a prism based on a square, in this case a predefined cube (accordingly to the fact that at least four containers are required). This could be change in run time, as we will show but this consequently will increase the amount of prim faces and as a consequence the polygon-based number of sizes. First, we create the rule 5-14 to create the counter for the prism sizes. If the is no counter, create it an initialize it to 0



Rule 5-14 Derivation of the updates relationship for an input facet

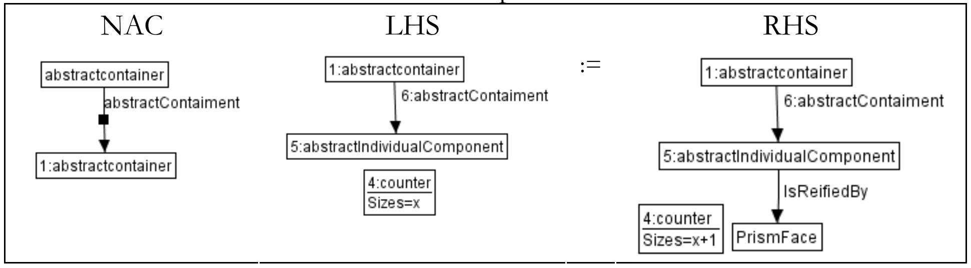
Then we mapped to a prism face each container at level "leaf-1" with rule 5-15. We count also the number of AC to determine at the



Rule 5-15 A creation of Prism face derived from containment relationships at the abstract

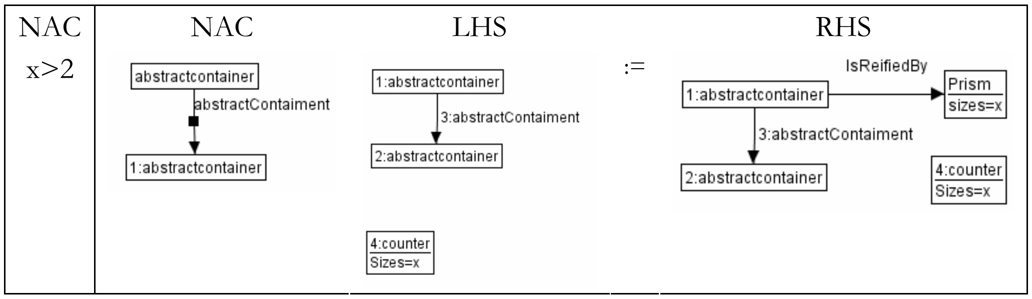
level

There is also a need to add a similar rule but for AIC which, as in our example the first task to start the application, will be in a last face but is executed in an AIC and not in a AC. The rule 5-16 solves this problem.



Rule 5-16 A creation of Prism face derived from containment relationships at the abstract level

Finally we create a prism with *x* sizes.



Rule 5-17 A creation of Prism face derived from containment relationships at the abstract level

5.2.3.b Selection of CICs

This sub-step involves the highest number of rules of all transformation sets as the different combinations of facet types, data types, cardinalities,…, are numerous. Table 5-1 provides the subset of rules applied in this case study. The designer can choose among the different alternatives provided by these rules.

Abstract Interaction Component	**Facet Specification**	**Information to take into account**	**Possible Concrete Interaction Component**
“start”	Control	Feedback	A trigger
“create name” and	Create attribute	Data type, domain	A text output

"create zipCode"	value	characteristics	with a text input associated to it
"select gender and select ageCategory"	Select attribute value + selection values known	Data type, domain characteristics, selection values	A dropdown list , a group of radio buttons textual or characters.
"Show Questionnaire"	Output (value unknown)	Attribute, data type, domain characteristics	An output text
"Select Answer"	Select attribute value + repetitive (selection values not known)	Data type, domain characteristics	A dropdown list, a group of option buttons
"Send Questionnaire"	Control	Feedback	A trigger
"Navigation"	Navigation	Feedback	A trigger

Table 5-1 Correspondence between AIO types and CIO types

5.2.3.c CIC placement

Physical constraints related to the size of the container and 3DCIC are considered. Each face of the cube could have just as much a pair of CIC at the same level, consequently this depends on the size of the components. In the case of text component this also depends on the size of the string that will display.

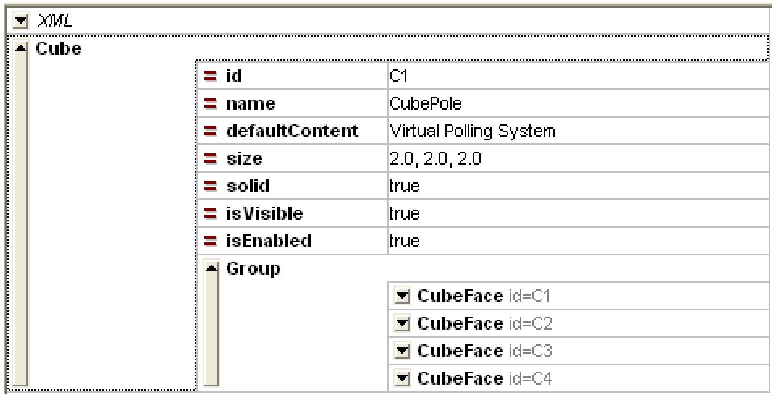
5.2.3.d Navigation definition

Navigation specifies how the visibility property of CCs is set and, consequently, defines transitions between them. Since all elements are not presented simultaneously into the same prism face, there is a particular need to define a sophisticated navigation scheme. Some schemes can be added for this purpose, so we identify that for any prism there is a need to add triggers that allows navigation, i.e. go back and forward to each face of the prism (this will be restricted depending on the navigation info from the task model, maybe certain information could not be accessible while other is accessed). In this particular case just the fourth faces allows navigation.

5.2.3.e Resulting specification

The resulting specifications are obtained by realizing the above development sub-steps. Figure 5-5 presents a mock-up of the graphical UI. For the section start one face of the cube will launch this task, the second face corresponds to the provide personal data task. In this part the rectangles next to the name and zip code, corresponds to the surface zone that will render the input text. The arrows for navigation are shown in the bottom of faces two-four.

A global view of the UsiXML mapping could be seen below, the cube as the container is divided in 6 faces but the top and the bottom are useless in this case. Each of the fourth faces of the fourth sized prism (cube) has a purpose that is to render each of the 3D graphical individual components (3DGIO). As four AC were required the cube clearly works as an option to render each AC in each of its four faces. Later we will show another option to render the same problem so as to increment the quantity of question to use another shape instead of the cube.


XML
Cube

id	C1
name	CubePole
defaultContent	Virtual Polling System
size	2.0, 2.0, 2.0
solid	true
isVisible	true
isEnabled	true
Group	CubeFace id=C1
	CubeFace id=C2
	CubeFace id=C3
	CubeFace id=C4

The UsiXML resulting from this process is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Cube id="C1" name="CubePole" defaultContent="Virtual Polling System" size="2.0, 2.0, 2.0"
solid="true" isVisible="true" isEnabled="true">
    <Group>
        <CubeFace id="C1">
            <SphereTrigger defaultContent="Start" radious="1.5" solid="True" isVisible="true"
                        isEnabled="true">
                <Transform scale="8.23 8.23 8.23" translation="0.27 12.14 18.30"/>
                <TouchSensor id="TS1" enabled="True"/>
                <Appearance name="ButtonAppe" id="App1">
                    <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                            emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
                </Appearance>
            </SphereTrigger>
        </CubeFace>
        <CubeFace id="C2">
            <Outputtext3D defaultContent="Name" id="T1">
                <Appearance name="ArrowAppe3" id="Back3">
                    <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                            emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
                </Appearance>
                <Transform translation="-1.51 -0.11 0.19"/>
```

```
</Outputtext3D>
<Inputtext3D defaultContent="" id="IT1">
     <Transform translation="0.025 -0.11 0.19"/>
     <Appearance name="ArrowAppe3" id="Back3">
          <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                       emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
     </Appearance>
     <TouchSensor id="TST1" enabled="True"/>
</Inputtext3D>
<Outputtext3D defaultContent="Zip Code" id="T2">
     <Transform translation="-1.39 -0.22 0.02"/>
     <Appearance name="ArrowAppe3" id="Back3">
          <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                       emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
     </Appearance>
</Outputtext3D>
<Inputtext3D defaultContent="" id="IT1">
     <Transform translation="0.09 -0.22 0.02"/>
     <Appearance name="ArrowAppe3" id="Back3">
          <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                       emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
     </Appearance>
     <TouchSensor id="TST2" enabled="True"/>
</Inputtext3D>
<Outputtext3D defaultContent="Gender" id="T3">
     <Transform translation="-1.60 -0.33 0.0"/>
     <Appearance name="TEXT" id="TEXT">
          <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                       emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
     </Appearance>
</Outputtext3D>
<radioButton id="" defaultContent="M" solid="True" isVisible="true" isEnabled="true"
                  defaultState="True" groupName="Gender"/>
<Transform translation="-.60 -0.33 0.0"/>
<Appearance name="TEXT" id="TEXT">
     <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                  emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
</Appearance>
<TouchSensor id="TSRB1" enabled="True"/>
<radioButton/>
<radioButton id="" defaultContent="F" solid="True" isVisible="true" isEnabled="true"
                  defaultState="False" groupName="Gender"/>
<Transform translation="0.0 -0.33 0.0"/>
<Appearance name="TEXT" id="TEXT">
     <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                   emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
</Appearance>
<TouchSensor id="TSRB2" enabled="True"/>
<radioButton/>
<Outputtext3D defaultContent="Age" id="T4">
     <Transform translation="-1.6 -.44 0.0"/>
     <Appearance name="TEXT" id="TEXT">
          <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                       emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
     </Appearance>
</Outputtext3D>
<radioButton id="" defaultContent="18-25" solid="True" isVisible="true" isEnabled="true"
                  defaultState="True" groupName="Age"/>
<Transform translation="-.60 -0.44 0.0"/>
<Appearance name="TEXT" id="TEXT">
```

```
        <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
    <TouchSensor id="TSRB1" enabled="True"/>
    <radioButton/>
    <radioButton id="" defaultContent="25-45" solid="True" isVisible="true" isEnabled="true"
                defaultState="False" groupName="Age"/>
    <Transform translation="0.0 -0.44 0.0"/>
    <Appearance name="TEXT" id="TEXT">
        <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
    <TouchSensor id="TSRB1" enabled="True"/>
    <radioButton/>
    <radioButton id="" defaultContent="45+" solid="True" isVisible="true" isEnabled="true"
                defaultState="False" groupName="Age"/>
    <Transform translation="0.6 -0.44 0.0"/>
    <Appearance name="TEXT" id="TEXT">
        <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
    <TouchSensor id="TSRB1" enabled="True"/>
    <radioButton/>
    <ArrowTrigger id="A1" defaultContent="Back" solid="True" isVisible="true"
                isEnabled="true">
        <Transform scale="0.76 0.18 1.0" translation="17.3 5.9 7.08" rotation="0.0 1.0 0.0
                                                                        1.570796"/>
        <TouchSensor id="TS2" enabled="True"/>
        <Appearance name="ArrowAppe1" id="Back1">
            <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                    emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
        </Appearance>
    </ArrowTrigger>
    <ArrowTrigger id="A2" defaultContent="Next" solid="True" isVisible="true"
                isEnabled="true">
        <Transform scale="8.23 8.23 8.23" translation="-18.6 5.4 7.5"/>
        <TouchSensor id="TS3" enabled="True"/>
        <Appearance name="ArrowAppe2" id="App2">
            <Material diffuseColor="0.8 0.0 0.0" specularColor="0.11 0.11 0.11"
                    emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
        </Appearance>
    </ArrowTrigger>
</CubeFace>
<CubeFace id="C3">
    <Outputtext3D defaultContent="The professor teach what it was expected?" id="T4">
        <Transform translation="-0.62 0.0 -0.52"/>
        <Appearance name="TEXT" id="TEXT">
            <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                    emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
        </Appearance>
    </Outputtext3D>
    <radioButton id="" defaultContent="Yes" solid="True" isVisible="true" isEnabled="true"
                defaultState="True" groupName="YesNo1"/>
        <Transform translation="0.0 -0.5 0.0"/>
        <Appearance name="TEXT" id="TEXT">
            <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                    emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
        </Appearance>
        <TouchSensor id="TSRB1" enabled="True"/>
    <radioButton/>
```

```
      <radioButton id="" defaultContent="No" solid="True" isVisible="true" isEnabled="true"
                  defaultState="False" groupName="YesNo1"/>
          <Transform translation="0.5 -0.5 0.0"/>
          <Appearance name="TEXT" id="TEXT">
              <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
          </Appearance>
          <TouchSensor id="TSRB2" enabled="True"/>
      <radioButton/>
      <Outputtext3D defaultContent="Did you like the course?" id="T4">
          <Transform translation="-0.62 -1.0 -0.52"/>
          <Appearance name="TEXT" id="TEXT">
              <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
          </Appearance>
      </Outputtext3D>
      <radioButton id="" defaultContent="Yes" solid="True" isVisible="true" isEnabled="true"
                  defaultState="True" groupName="YesNo2"/>
      <Transform translation="0.0 -1.5 -0.52"/>
      <Appearance name="TEXT" id="TEXT">
          <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                    emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
      </Appearance>
      <TouchSensor id="TSRB3" enabled="True"/>
      <radioButton/>
      <radioButton id="" defaultContent="No" solid="True" isVisible="true" isEnabled="true"
                  defaultState="False" groupName="YesNo2"/>
      <Transform translation="0.5 -1.5 -0.52"/>
      <Appearance name="TEXT" id="TEXT">
          <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                    emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
      </Appearance>
      <TouchSensor id="TSRB4" enabled="True"/>
      <radioButton/>
      <ArrowTrigger id="A3" defaultContent="Back" solid="True" isVisible="true"
                    isEnabled="true">
          <Transform scale="0.76 0.18 1.0" translation="3.46 5.27 -19.06" rotation="0.0 1.0
                                                                          0.0 1.570796"/>
          <TouchSensor id="TS4" enabled="True"/>
          <Appearance name="ArrowAppe2" id="Back2">
              <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
          </Appearance>
      </ArrowTrigger>
      <ArrowTrigger id="A4" defaultContent="Next" solid="True" isVisible="true"
                    isEnabled="true">
          <Transform scale="8.23 8.23 8.23" translation="-18.6 5.4 7.5"/>
          <TouchSensor id="TS5" enabled="True"/>
          <Appearance name="ArrowAppe2" id="App2">
              <Material diffuseColor="0.8 0.0 0.0" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
          </Appearance>
      </ArrowTrigger>
  </CubeFace>
  <CubeFace id="C4">
      <Outputtext3D defaultContent="Did you enjoy the course?" id="T4">
          <Transform translation="-1.60 -0.33 0.0"/>
          <Appearance name="TEXT" id="TEXT">
              <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
```

```
                </Appearance>
            </Outputtext3D>
            <radioButton id="" defaultContent="Yes" solid="True" isVisible="true" isEnabled="true"
                         defaultState="True" groupName="YesNo3"/>
            <Transform translation="-.60 -0.33 0.0"/>
            <Appearance name="TEXT" id="TEXT">
                <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                          emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
            </Appearance>
            <TouchSensor id="TSRB5" enabled="True"/>
            <radioButton/>
            <radioButton id="" defaultContent="No" solid="True" isVisible="true" isEnabled="true"
                         defaultState="False" groupName="YesNo3"/>
            <Transform translation="0.0 -0.33 0.0"/>
            <Appearance name="TEXT" id="TEXT">
                <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                          emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
            </Appearance>
            <TouchSensor id="TSRB6" enabled="True"/>
            <radioButton/>
            <ArrowTrigger id="A5" defaultContent="Back" solid="True" isVisible="true"
                          isEnabled="true">
                <Transform scale="0.76 0.18 1.0" translation="-19.15 5.3 -6.5" rotation="0.0 1.0 0.0
                                                                                  1.570796"/>
                <TouchSensor id="TS6" enabled="True"/>
                <Appearance name="ArrowAppe3" id="Back3">
                    <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                              emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
                </Appearance>
            </ArrowTrigger>
            <SphereTrigger id="B2" defaultContent="Send" radious="1.5" solid="True"
                           isVisible="true" isEnabled="true">
                <Transform scale="8.23 8.23 8.23" translation="-18.6 5.4 7.5"/>
                <TouchSensor id="TS7" enabled="True"/>
                <Appearance name="ButtonAppe2" id="App2">
                    <Material diffuseColor="0.8 0.0 0.0" specularColor="0.11 0.11 0.11"
                              emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
                </Appearance>
            </SphereTrigger>
        </CubeFace>
    </Group>
</Cube>
```

How each face could look in a 2D view is shown below. The arrows show the need for navigation.

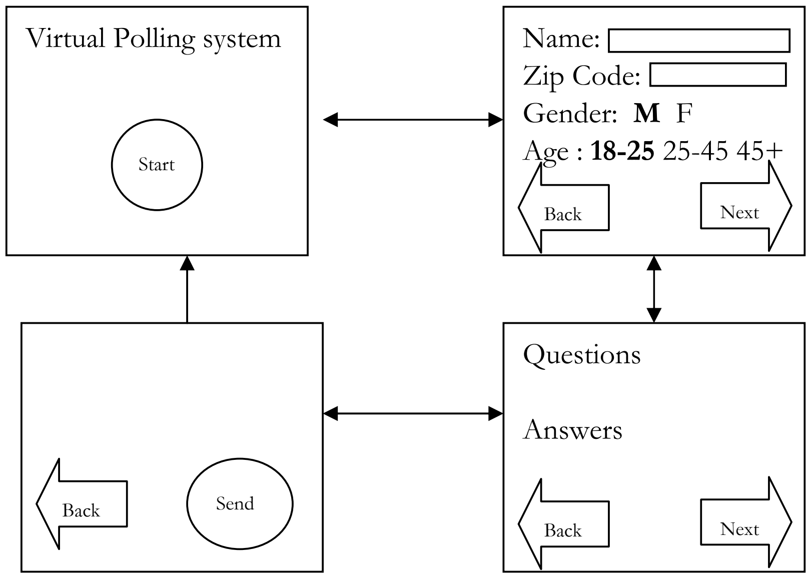


Figure 5-5 Mock-up of the UI

5.2.4 Step 4: From Concrete model to Final User Interface

The fourth step involves the transformation from CUI to FUI. In screenshot of Figure 5-6, the decomposition of ACs (option B) is fine-grained: the information related to the person are first acquired in a rotating cube (which was selected as the container), then each pair of questions is presented at a time with the facilities of going forward or backward like a wizard (using arrow triggers). Since only 3 questions and one set of person information are considered, a cube is selected to present each of the fourth part. If for any reason, more questions were defined, let us say 5, a regular volume with 6 faces would be generated instead.

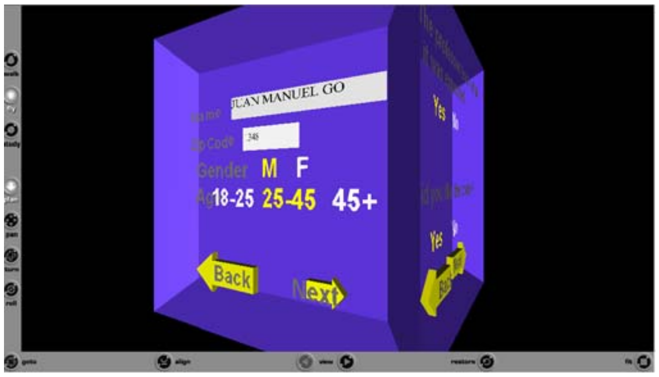


Figure 5-6 Polling system rendered in VRML

Each face of the cube is mapped to each of the ACs. The first face of the cube for the default content of the cube (i.e., the title) and the start button, see Figure 5-7:

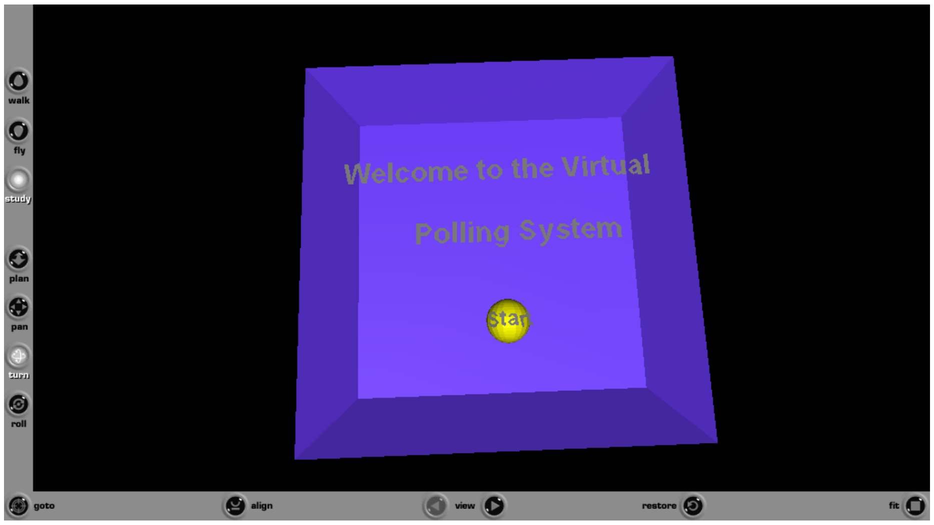


Figure 5-7 Polling system rendered in VRML

Below we show the UsiXML code generated for the CUI, corresponding to the first AC, that just have one AIC, which is mapped to a SphereTrigger. The cube as the principal container has its title, the attribute *defaultContent* (inherited from CIO model). This title is part of the first face of the cube. The second cube attribute is its size.

```
<Cube  id="C1" name="CubePole" defaultContent="Virtual Polling System" size="2.0, 2.0, 2.0"
       solid="true" isVisible="true" isEnabled="true">
```

```
   <Group>
      <CubeFace id="C1">
         <SphereTrigger defaultContent="Start" radius="1.5" solid="True" isVisible="true"
                           isEnabled="true">
            <Transform scale="8.23 8.23 8.23" translation="0.27 12.14 18.30"/>
            <TouchSensor id="TS1" enabled="True"/>
            <Appearance name="ButtonAppe" id="App1">
               <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                           emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
            </Appearance>
         </SphereTrigger>
      </CubeFace>
      <CubeFace id="C2">
      <CubeFace id="C3">
      <CubeFace id="C4">
   </Group>
</Cube>
```

Continuing with the example, the second AC is mapped to the second face of the cube; Figure 5-8 shows the FUI resulting.

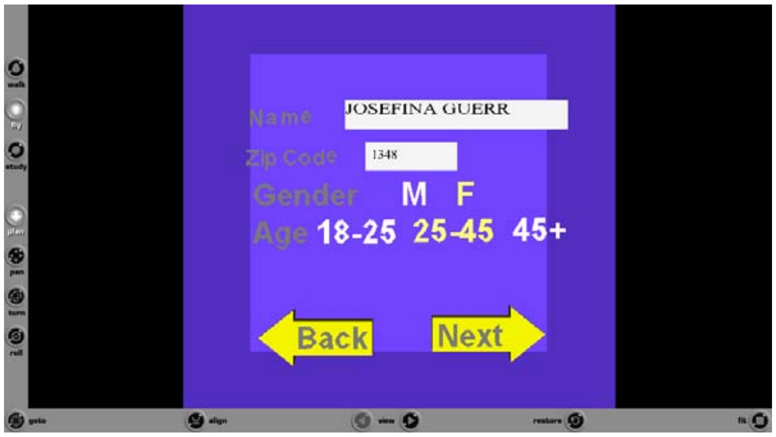

Name JOSEFINA GUERR
Zip Code 1348
Gender M F
Age 18-25 25-45 45+

Figure 5-8 Appearance meta-model

As in this example we have three different groups of CIO, which are:

1. A text output and a text input for the create name and zip code tasks.
2. A text output and a group of radio buttons for the select gender and age category tasks.
3. Arrow triggers for navigation.

We show The CUI UsiXML code related to each of the above two situations but showing just one of its two cases. Below the section that corresponds to the name creation. OutputText (specialized in any kind of output text) and InputText (specialized in any kind of input text) are two components from the 3DGIC. For an InputText there is a need to declare a sensor (similar than registering the event handlers in java) to listen to events that could be triggered by the InputText. Such events could be, is Over (the pointer is on the input text), key Down (to identify the key pressed from the keyboard). The rest of the code describes the appearance for the text and its position (transform).

```
<Outputtext3D defaultContent="Name" id="T1">
    <Appearance name="ArrowAppe3" id="Back3">
        <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
    <Transform translation="-1.51 -0.11 0.19"/>
</Outputtext3D>
<Inputtext3D defaultContent="" id="IT1">
    <Transform translation="0.025 -0.11 0.19"/>
    <Appearance name="ArrowAppe3" id="Back3">
        <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
    <TouchSensor id="TST1" enabled="True"/>
</Inputtext3D>
```

The second part of the face shows an output text followed by a radio buttons group. Below the code related to the radio buttons group. We use the same attributes that are described in UsiXML for this component, as in 3D there is no difference. We need a radio group name, a default state (whether is selected or not). The rest of the code describes the appearance for the text and its position (transform).

```
<radioButton id="" defaultContent="Yes" solid="True" isVisible="true" isEnabled="true"
            defaultState="True" groupName="YesNo1"/>
            <Transform translation="0.0 -0.5 0.0"/>
            <Appearance name="TEXT" id="TEXT">
                <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
            </Appearance>
            <TouchSensor id="TSRB1" enabled="True"/>
        <radioButton/>
        <radioButton id="" defaultContent="No" solid="True" isVisible="true" isEnabled="true"
                    defaultState="False" groupName="YesNo1"/>
            <Transform translation="0.5 -0.5 0.0"/>
            <Appearance name="TEXT" id="TEXT">
                <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                        emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
            </Appearance>
            <TouchSensor id="TSRB2" enabled="True"/>
        <radioButton/>
```

Finally the arrow triggers used for navigation purposes. The default content of the trigger defines the text attached to them. The rest of the code describes the appearance and its position (transform). There is a need for a sensor to trigger an action, in this case will be the translation of the cube +- 90 digress related to Y axis.

```
<ArrowTrigger id="A1" defaultContent="Back" solid="True" isVisible="true"
              isEnabled="true">
    <Transform scale="0.76 0.18 1.0" translation="17.3 5.9 7.08" rotation="0.0 1.0 0.0
                                                                     1.570796"/>
    <TouchSensor id="TS2" enabled="True"/>
    <Appearance name="ArrowAppe1" id="Back1">
        <Material diffuseColor="0.8 0.8 0.0" specularColor="0.11 0.11 0.11"
                  emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
</ArrowTrigger>
<ArrowTrigger id="A2" defaultContent="Next" solid="True" isVisible="true"
              isEnabled="true">
    <Transform scale="8.23 8.23 8.23" translation="-18.6 5.4 7.5"/>
    <TouchSensor id="TS3" enabled="True"/>
    <Appearance name="ArrowAppe2" id="App2">
        <Material diffuseColor="0.8 0.0 0.0" specularColor="0.11 0.11 0.11"
                  emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
    </Appearance>
</ArrowTrigger>
```

The third and four faces for the cube, do not have any different object that could be interesting to describe, just output text, radio buttons groups and a Sphere trigger, see Figure 5-9. What is relevant is just that the mapping to each FUI, in this case in VRML corresponds to the CUI model, describe in UsiXML.

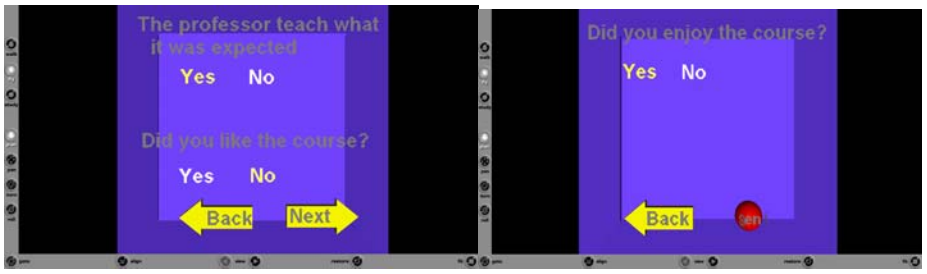
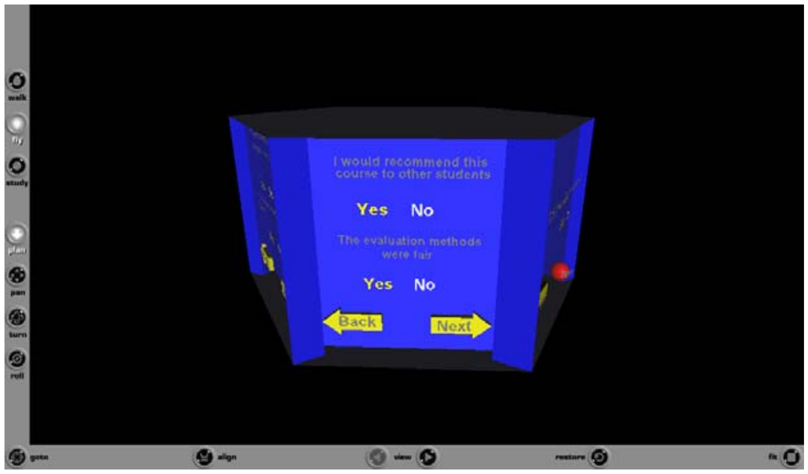

The professor teach what it was expected
Yes
No
Did you like the course?
Yes
No
Back
Next
Did you enjoy the course?
Yes
No
Back

Figure 5-9 Faces 3 and 4

If the set of questions could be more extended, automatically the sides of the prism are expanded. In the example above, the cube fits the necessities of a pool of questions. In Figure 5-10 a 6-sizes prism serves as container instead of the cube. The path through obtain the six-sizes shape is analogous to the previous one, the difference is the quantity of questions at the when generating the final user interface. In this second scenario, instead of four containers there is a need for 2 more, as four more questions were added to the pool. As a consequence the shape required to handle this information varies.


I would recommend this course to other students

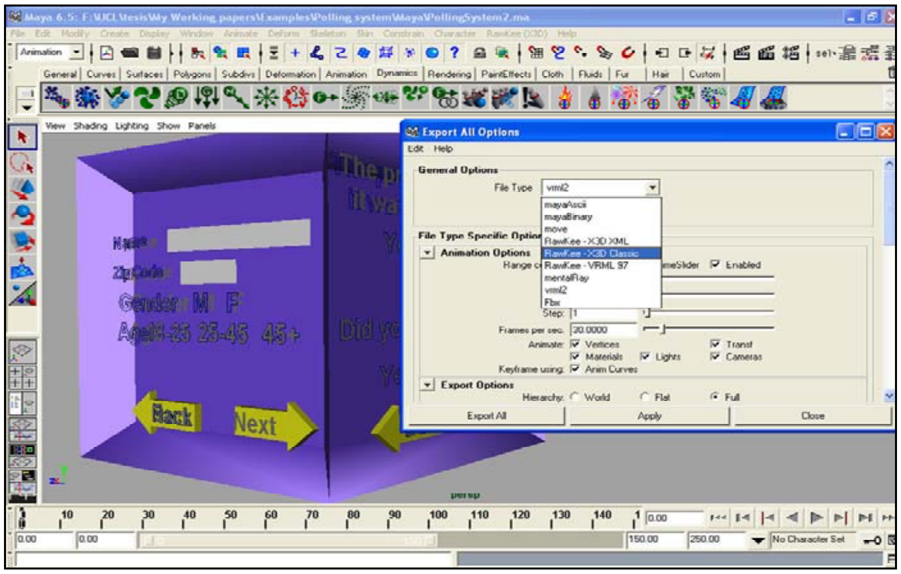
The evaluation methods were fair

Figure 5-10 Hexahedron Polling System

5.2.4.a Sub-Step: From Concrete to a High Level Editor

The UsiXML description of the UI is not enough; we need an editor to manipulate the 3D objects easily with an automatic feedback of the modifications done by the user, related to the position in the virtual space of the CUI. A high level editor could be used for this purpose, so instead of transforming a UsiXML CUI description directly to its FUI representation we propose to transform it to code corresponding to high level editor than allows a WYSWYG (what you see is what you get) visualization. In the chapter 2 we discussed some of the editors that allows this kind of manipulation, so as we analyzed which could be more suitable for import/export operations to the most used 3D languages, such as: Java3D, VRML, or X3D.

We found Maya ASCII file one option. The files is opened in the Maya editor, objects could be manipulated, and finally exported, Figure 5-11,. Maya plug-ins offers, among others exporters, RawKee (http://rawkee.sourceforge.net/), an open source (LGPL) X3D plug-in, that exports Maya's 3D data as an X3D or VRML file with support for scripting.



General Options
File Type: vrml2
mayaAscii
mayaBinary
move
RawKee - X3D XML
RawKee - X3D Classic
RawKee - VRML 97
mentalRay
vrml2
Fbx

File Type Specific Options
Animation Options
Step: 1
Frames per sec: 30.0000
Animate: ☑ Vertices ☑ Transf ☑ Materials ☑ Lights ☑ Cameras
Keyframe using: ☑ Anim Curves
☑ Enabled
Export Options
Hierarchy: World Flat [✓ Full]
Export All | Apply | Close

Figure 5-11 Edition of the 3DUI in Maya

5.2.4.b Sub-Step: From High Level Editor to FUI

Once edited in Maya the User Interface can be exported to VRML or X3D. There will be a need to do reverse engineering so the coordinates specified at the CUI level could correspond to the new specification done in Maya. The FUI produced by Maya generates more than 30 thousand lines of code. There is a final need to add some script coding to add the behavior to the example such as rotating when clicking on the arrows for the navigation.

5.2.5 Reconstruction of the case study in Java 3D

The scenario proposed in the previous example uses containers to render the information that will be shown in the virtual space. Normally controls and any CIO of the UI in 2D are attached to any kind of container. In virtual space the counter part of the window is the virtual space itself. So, object could be rendered in the virtual space, floating, with out any need of containers.

Designers are allowed to decide whether they prefer to use containers (as we did in the face of the prism) or attach directly the components to the virtual space. This design decision should be taken when passing from the Abstract model to the concrete model. We could decide whether container contained in the main container (the virtual space in 3D) will be attached to a surface that then will corresponds to a prism or will be attached directly to the virtual space. The second option will be used to show the results of this kind of decision.

If we do not mapped the containers to surfaces then there will not be a final instance of any kind of prism to render each surface that serves as container. The virtual space will render all the content of the container describe in the CUI model.

In Figure 5-12, the screenshot reproduces the worlds generated in Java3D where each AC (provision personal data and answer question) is mapped onto the virtual space. All AICs belonging to each AC are then mapped recursively onto Java3D widgets depending on their data type. In this particular case, the designer selected also the graphical representation if any, along with the textual representing. In this visualization, we propose another way to represent the category selection. Instead of using a comboBox, or the traditional view of icons attached to radio button, we proposed the use of 3D personages instead of icons. This 3D graphic representation of the option could reinforce the understanding, notice that we keep the text below the personages.

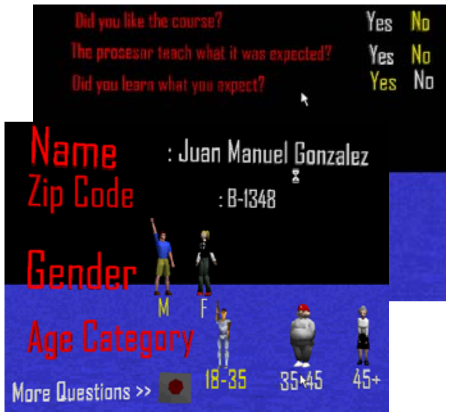


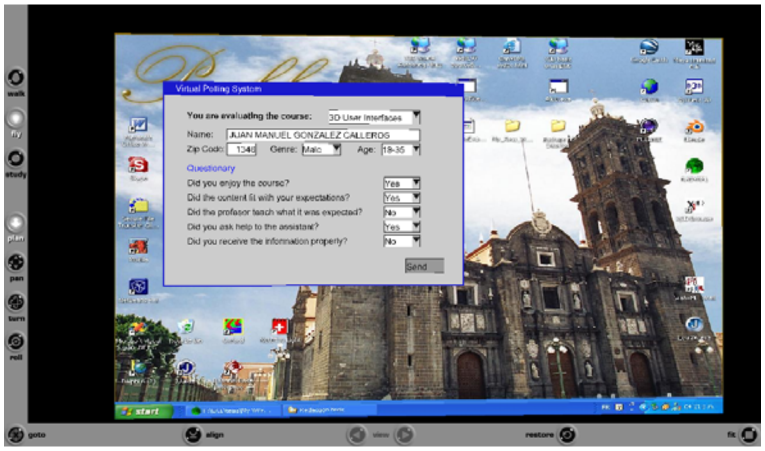
Figure 5-12 Edition of the 3DUI in Maya

To obtain this result there is a need to have a look at the CUI model. Each 3DCIO has an attribute called icon. This icon in 2D generally corresponds to a bitmap image that is attached to the controls in a UI. In 3D we propose that icons could be any shape, so in this case we use the same definition as in the previous example to define radio button but adding the url corresponding to each radio button. The code could be seen as follows:

```
<radioButton id="" defaultContent="18-35" solid="True" isVisible="true" isEnabled="true"
                         defaultState="True" groupName="YesNo1" icon="youngMan.java"/>
                        <Transform translation="0.0 -0.5 0.0"/>
                        <Appearance name="TEXT" id="TEXT">
                              <Material diffuseColor="0.3 0.3 0.3" specularColor="0.11 0.11 0.11"
                                             emissiveColor="0.0 0.0 0.0" shininess="0.3"/>
                        </Appearance>
                        <TouchSensor id="TSRB1" enabled="True"/>
<radioButton/>
```

5.2.6 Reconstruction of the case study for the 3D rendering of its corresponding 2DUI

The UsiXML specifications at the CUI could also be interpreted in VUIToolkit, a rendering engine for 3D UIs specified in UsiXML in VRML97/X3D. In the screenshot of the Figure 5-13, we show the result of using the Toolkit that generates the 3D rendering of how our polling system could look in a 2D user interface. The 2D components have been enriched with volumes. One can discuss that the components are rendered as 3D widgets in a way that remains similar to the "Look & Feel" of 2D widgets, except that the "Feel" is a genuine 3D behavior. According to this view, this kind of FUI can be interpreted only as a 3D rendering of 2D UIs, even if their specifications are toolkit-independent [Moli05]. This approach provides an option to the use of Java applets UIs to manipulate virtual applications in the Web, instead, the use of the VUIToolkit would not disrupt the 3D "look".


Virtual Polling System

You are evaluating the course: 3D User Interfaces

Name: JUAN MANUEL GONZALEZ CALLEROS

Zip Code: 1340 Genre: Male Age: 18-35

Questionary

Did you enjoy the course? Yes

Did the content fit with your expectations? Yes

Did the profesor teach what it was expected? No

Did you ask help to the assistant? Yes

Did you receive the information properly? No

Figure 5-13 Three-D rendering of the 2D interface for the polling system

5.3 Case Study 2: the Virtual interactive Office

This case study refers to a virtual interactive environment that corresponds to the description of virtual objects (chairs, table, walls, and doors) and interactive objects for the 3D UI. The user task, depicted in Figure 5-14, that the user can interact with a table, navigate through the room and interact with a screen. For the **navigate room,** the task model uses the identified task pattern (section 4.3.1.b) travel and Wayfinding (which can be seen as a hidden task when the implementation do not considers virtual environment contents control, for instance internet plug-in for browser already control the system tasks for wayfinding). The **interact with big Screen** indicate as the **interact with screen** task refers to a turn on/off a screen than renders video or images.

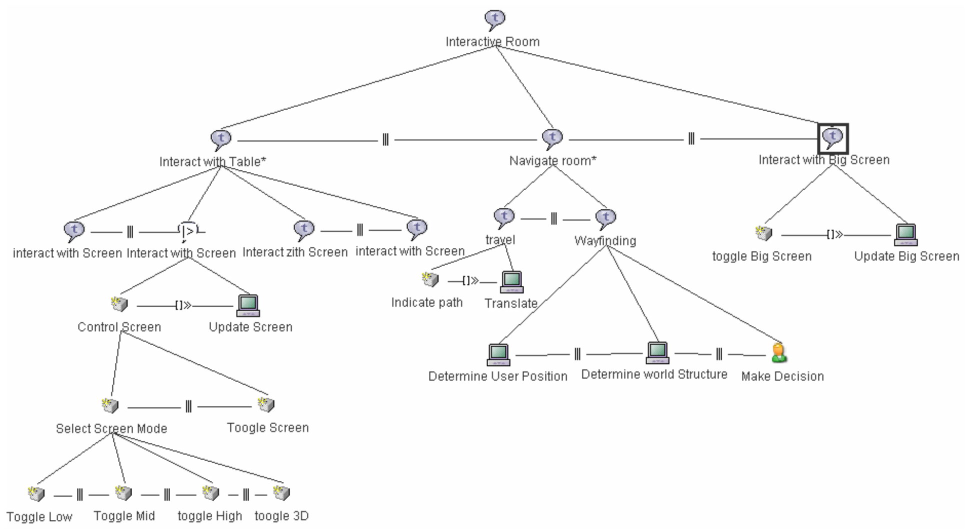


Figure 5-14 Virtual Office task Model

The task model is reified into the CUI model, using IdealXML, the rules are similar as in the previous example so there is no need to explain the internal process of the tool. What is relevant is to notice, that even that we put in the task model the **make decision task** for wayfinding, at the CUI model this kind of task do not appear, as they are not part of the UI.

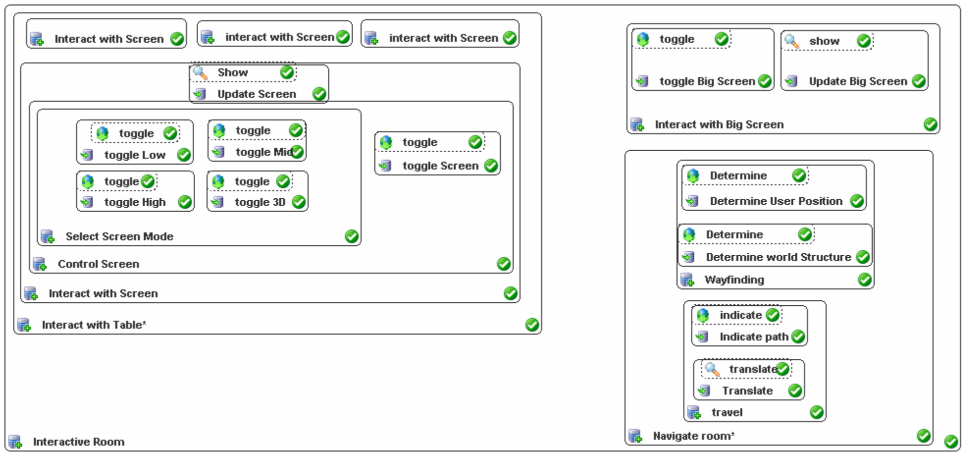


Figure 5-15 Virtual Office Concrete Model

5.3.1 Step 3: From Abstract model to Concrete User Interface model

The third step implies a transformational system that is composed of necessary rules for realizing the transition from AUI to CUIs. We won't consider in this example the attachment of objects to any surface, we just create a direct mapping, for each component and then in the high level editor each component is put in the corresponding shape.

This sub-step involves the highest number of rules of all transformation sets as the different combinations of facet types, data types, cardinalities,…, are numerous. Table 5-2 provides the subset of rules applied in this case study. The designer can choose among the different alternatives provided by the rules.

Abstract Interaction Component	Facet Specification	Information to take into account	Possible Concrete Interaction Component
"Navigation"	Navigation	The platform used, an internet Browser with any pug-in,	There is no need for any concretization of this task or any subtask
Interact with big screen/ Screen	Toggle + Element (Screen/Button)	The big screen reacts on touch, the small screen react with a toggle button.	Toggle button, touch screen

		This is also a design rule.	
"select screen mode"	Select attribute value + selection values known	The set of possible values are in subtask instead of a domain list of values.	A group of toggle button acting as a radio group.
"Toggle Low/Mid/High/3D"	Toggle + element		Toggle button
"Update Screen"	Communicate (Show) + Element (Image or video)	Attribute, data type, domain characteristics	An image component or video component

Table 5-2 correspondence between AIO types and CIO types

Physical constraints related to the size of the container and 3DCIC are considered with default values. Remember that if we could pass this specification to a high level editor allows us to arrange manually the objects. Ideally we expect to expand a virtual environment, such as Maya, in order to have import/ exports to our format, so; in this sense changes made in the toolkit can be track in the CUI model and backward, to have a consistent model at all the levels.

The resulting specifications are obtained by realizing the above transformational development sub-steps. Figure 5-16, 5-17, 5-18 present a mock-up of the graphical UI.

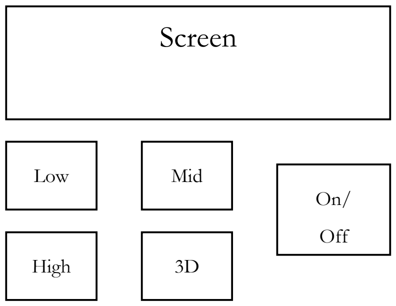


Figure 5-16 Mock-up of the Control Screen UI

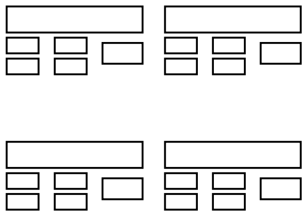


Figure 5-17 Mock-up of the Interact with table UI

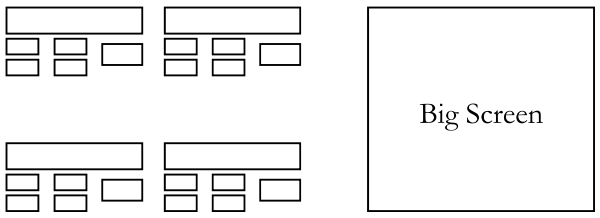


Figure 5-18 Mock-up of the Control Screen UI

5.3.2 Step 4: From Concrete model to Final User Interface

In screenshot of Figure 5-19, the Big screen is shown, in Figure 5-20 the control screen and interact with table in Figure 5-21, 5-22, each figure shows alone each FUI corresponding to each task. All the components together are in the screenshot of Figure 5-24 and 5-25, that is composed of 4 screenshots of the virtual office, in which the top left image corresponds to the hall and the entry to the virtual Room. All the content was specified manually in VRML, reusing components such as chairs, tables, etc. The navigation task is controlled by the Cortona player plug-in., which support the navigation with the mouse and keyboard, as input devices, the user just decide where to go. In figure 5-26, we introduce a new task, which is the virtual polling system (Case study 1). Notice that we can place this task anywhere in the virtual space, in this example the cube is on the table in the back of the room.

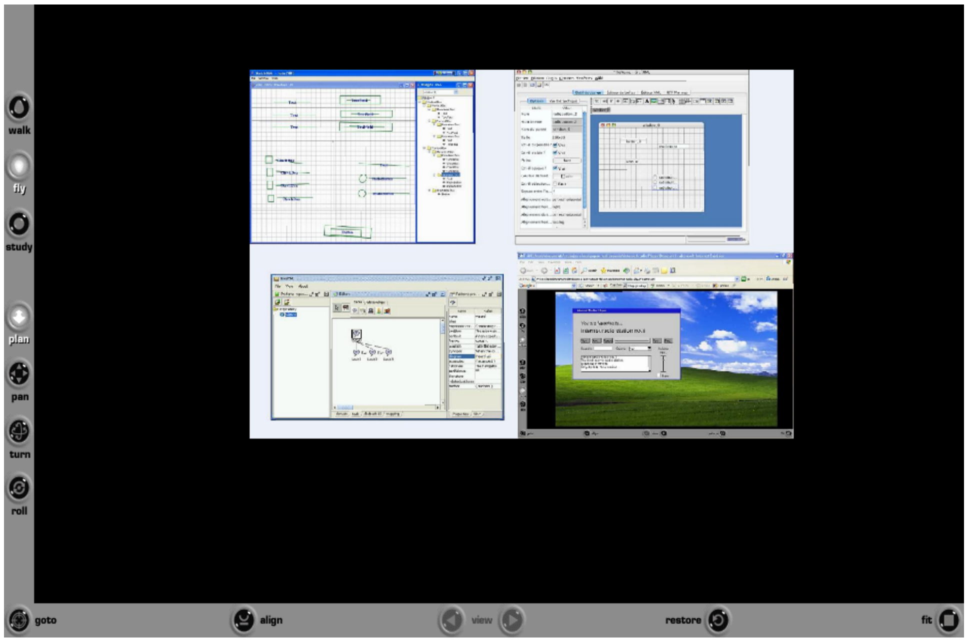


Figure 5-19 Big Screen rendered in VRML

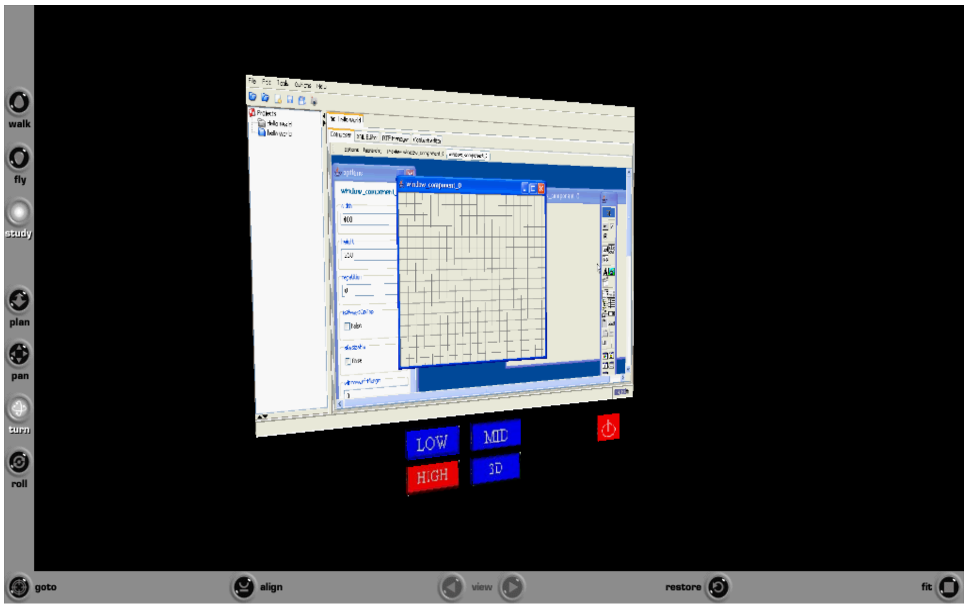


Figure 5-20 Control Screen rendered in VRML

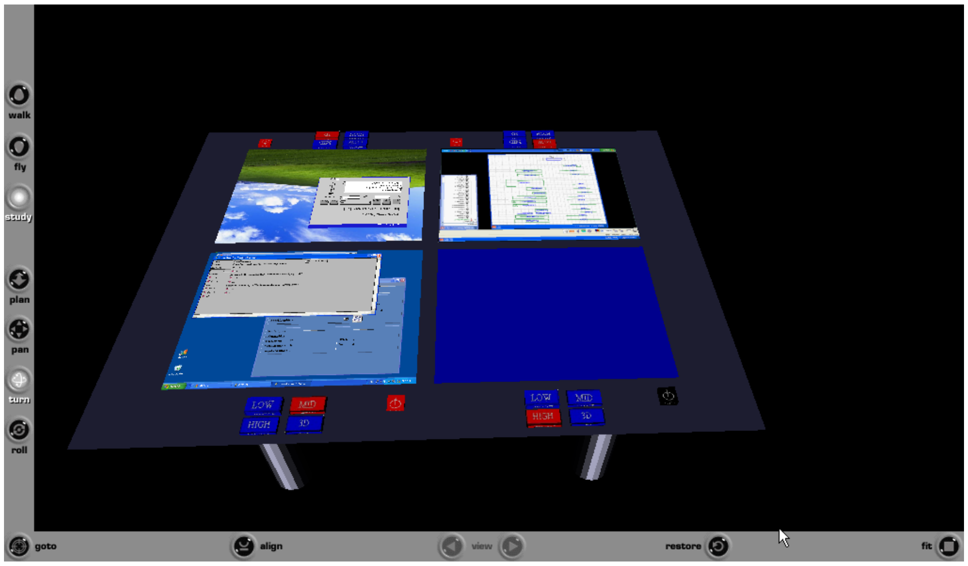


Figure 5-21 Interact with table task rendered in VRML

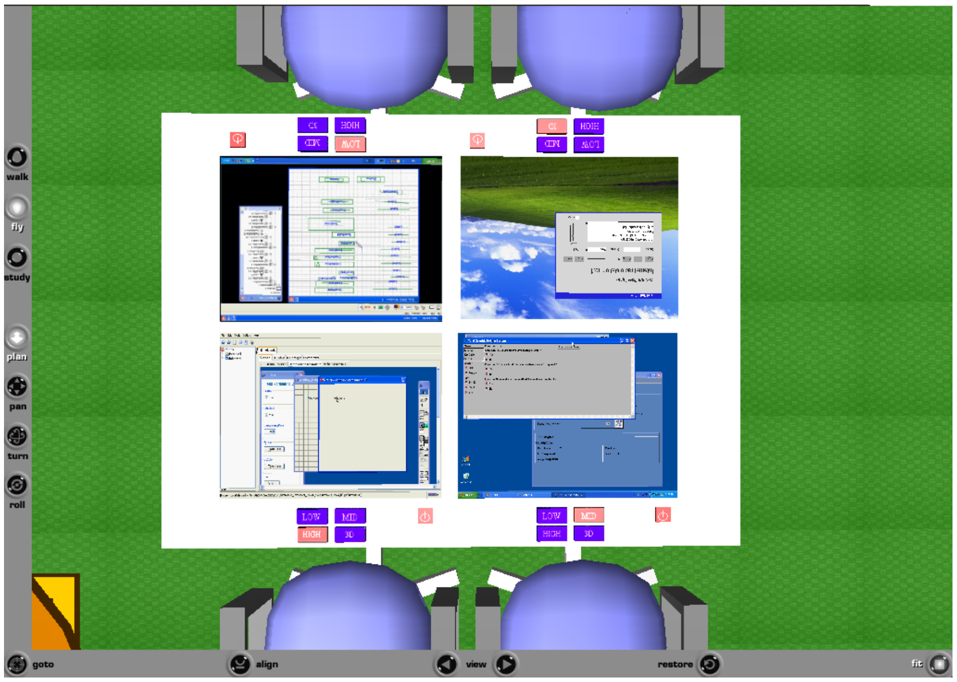


Figure 5-22 Top View of the virtual table rendered in VRML

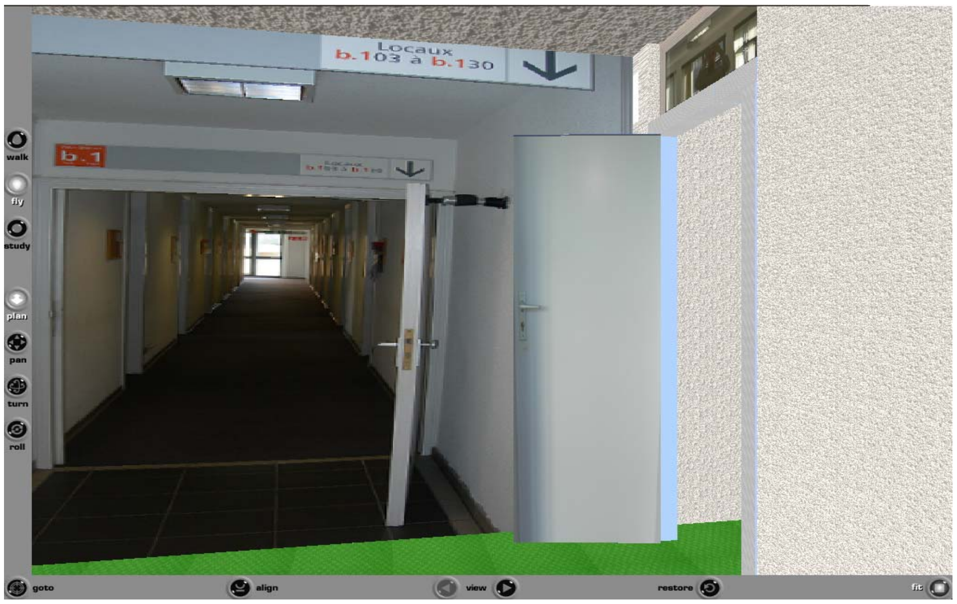


Figure 5-23 Hall Virtual Office

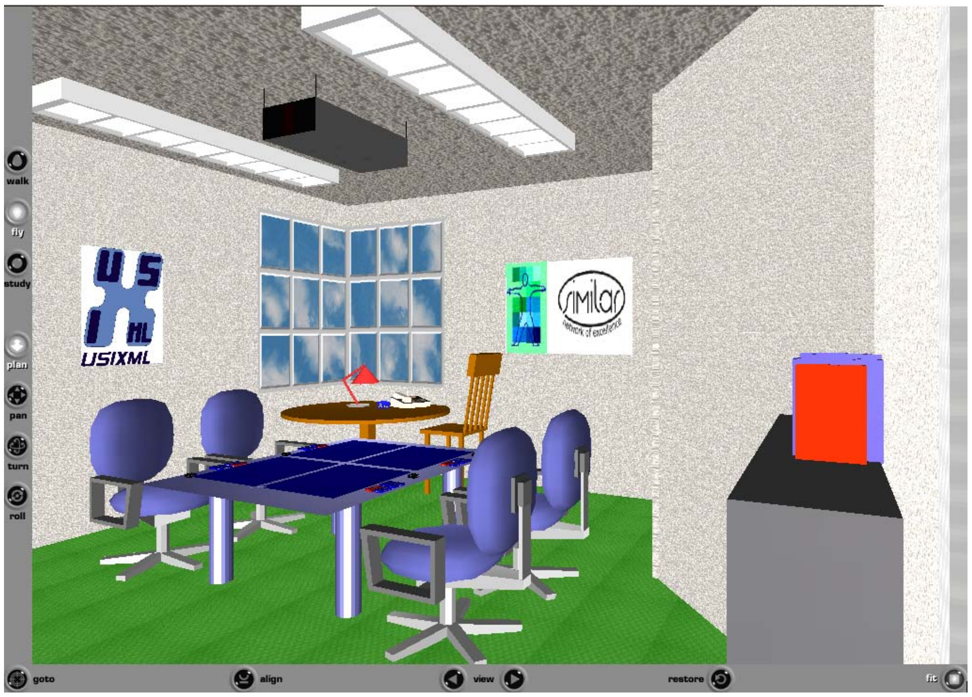


Figure 5-24 Entry view of the Virtual Office

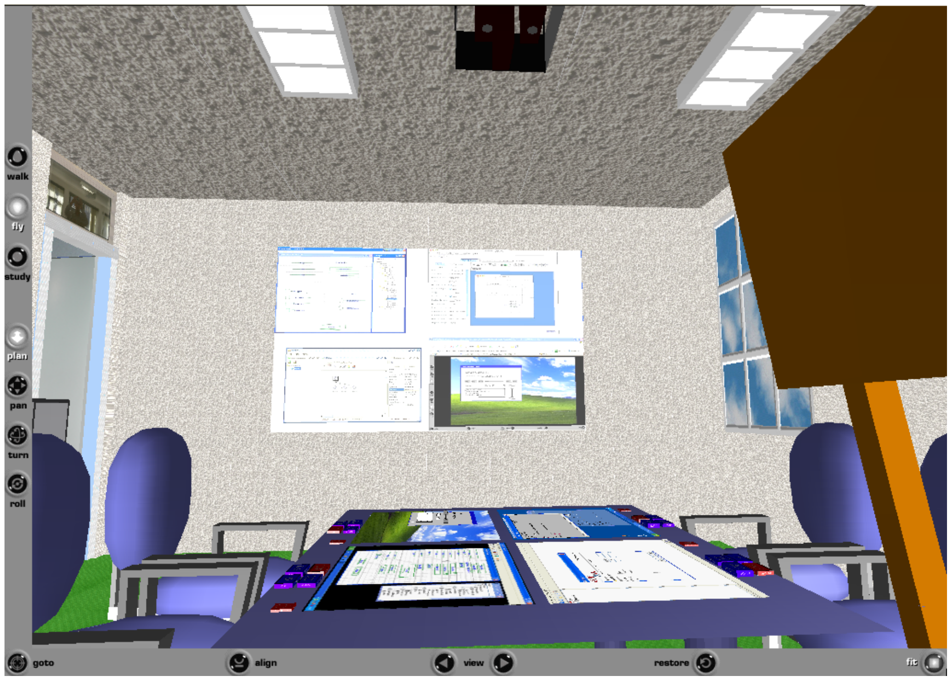
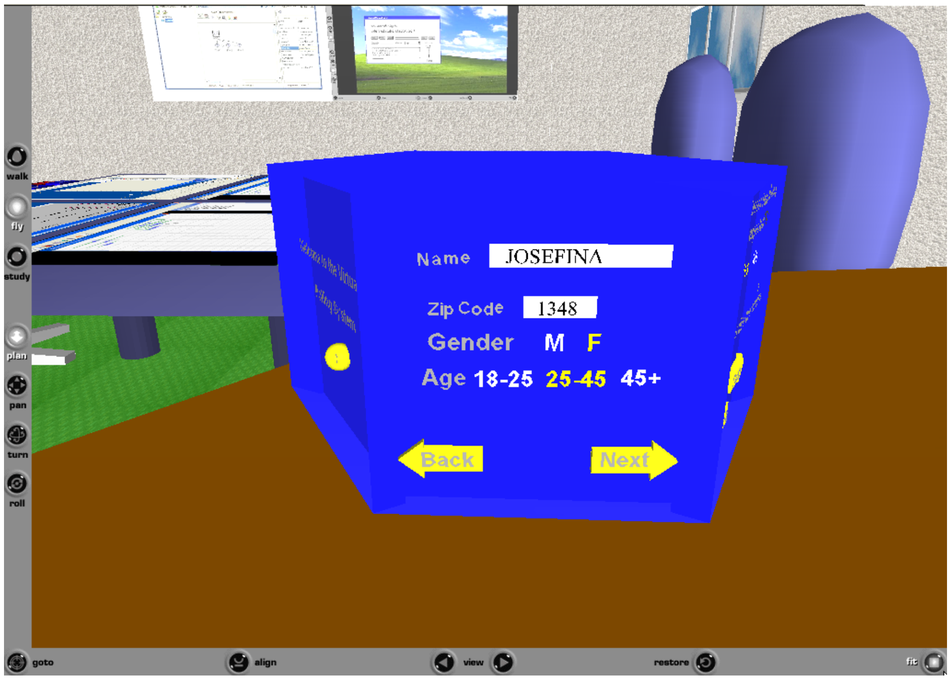


Figure 5-25 Back view Virtual Office



Name JOSEFINA
Zip Code 1348
Gender M F
Age 18-25 25-45 45+

Figure 5-26 Virtual Office

5.4 Conclusion

In this chapter we introduce two case studies to show the feasibility of the method proposed for the development of 3D UIs. The two case studies show how model-based development applies to two low-complex examples.

To solve these case studies we have followed the following procedure: (1) Building initial models. Such models have been edited with their associated editing tool. (2) Editing and debugging of rules within the AGG. (3) Importing initial models into the AGG graphical environment. (4) Selecting a transformation set and firing the rules contained in this set. (5) Generate UsiXML specifications and (6) generate 3D UIs.

This process led us to deduce the following conclusions regarding the strengths and weaknesses of our method.

Our case studies showed the **feasibility** of developing a 3D UI in a model-based relying on explicit transformation catalogs at any time. The diversity of development paths that have been presented highlight the possibility of manipulating 3D UIs related artifacts according to different development scenarios and pave the way to consider multiple other alternatives. The reuse of transformations and components has been illustrated when transformation systems have been straightforwardly reused from one case study to another one.

The feasibility of the approach is much depending on the amount and the quality of the design rules that are also encoded in UsiXML. If a reasonably extensive set of rules is used, the generated results are usable. If this is not the case, the model resulting from the transformation could be considered as underspecified. It is then required to manually edit within a XML-compliant editor. Future work will therefore be dedicated to exploring more design options and encode them in UsiXML so as to serve better transformations. This does not mean that a generated 3D UI is as usable or more usable than a manually-produced one, but at least it could be obtained in a very fast way. Moreover, the exploration of alternative design options could be facilitated since they are operated at a higher level of abstraction than the code level.

Chapter 6 Validation

6.1 External Validation

External validation is realized by the application of our method on case studies. The main goal of these case studies is to show the feasibility i.e., the capability to solve the problems raised by the presented case studies.

Our case studies showed the **feasibility** of developing a 3D UI in a model-based relying on explicit transformation catalogs at any time. The diversity of development paths that have been presented highlight the possibility of manipulating 3D UIs related artifacts according to different development scenarios and pave the way to consider multiple other alternatives. The reuse of transformations and components has been illustrated when transformation systems have been straightforwardly reused from one case study to another one.

Further evaluation can be done related to the external evaluation such as usability test, questionnaires applied to domain experts or any other source of external feedback that could enlarge the quality of the results provided here.

6.2 Internal Validation

The internal validation of a methodology consists in assessing its characteristics against a set of selected criteria. The relevant criteria, called requirements, for our methodology have been elicited and motivated after the state of the art of Chapter 2. This section proposes a discussion for each of these requirements.

Requirement 1: Expressivity – means that a conceptual framework should provide enough details to address problems that motivated the elicitation of its constituent concepts. In our context models should, at least, provide enough details to allow an implementation of the system it describes. This essential requirement is not fulfilled by many formal methods, for instance those focusing on verifying state properties of the system that is being built (Motivation: general principle in software engineering, Obs. 6).

Comment. This requirement was partially achieve with the ontology of concepts. Further descriptions and components need to be added to enrich the expressivity regarding 3D UIs.

Requirement 2: Machine processable – states that the proposed ontology should be legible by a machine. To allow the transformational approach.

Comment. AGG tool and IdealXML, are both computer programs which are the base of the method.

Requirement 3: Human readable – means that the provided ontology should be proposed in a format that enables its legibility by a human agent. Such efforts are done in InTML, CoGenIVE, VR-WISE and Contigra. Although the main concern is on machine processable.

Comment. Graphs when rules are applied are almost impossible to track by the human. UsiXML specifications are based on concepts that are easily to understand. However the abstraction level used, sometimes could difficult to understand the method.

Requirement 4: Standards – states that the expression means used to represent our ontology should rely on well accepted standards in the software engineering community, maybe using X3D as target language.

Comment. The use of standards in 3D is complicated as there is no one. Efforts to develop a standard was done with VRML, and when a lot of work had been done related to VRML, its predecessor X3D was not compatible with the tools developed. A great effort has been dedicated to enlarge the capabilities of X3D but yet there is no better or worse approach until now for not just the FUI level but for the rest of the models also.

6.2.1 Methodological Requirements

Requirement 5: Methodological explicitness – states that the constituent steps of our methodology should be defined in a way that facilitates the comprehension of its internal logic and its application.

Comment. In chapter 4 a clear an complete description of concepts is provided.

Requirement 6: Methodological flexibility – refers to the ability to initiate the development from any development stage (i.e., multiple entry points) and to terminate it at any development stage (i.e., multiple exit points).

Comment. Even that we just consider forward engineering we describe the need in the case study for reverse engineering path, when a FUI is modified using a high level editor, such as Maya, in order to have a CUI description that is consistent with the FUI. Such kind of work is done in the CoGenIVE tool for the VRIXML.

Requirement 7: Executability – states that development steps should be expressed in such a level of accuracy that it is possible to execute them by an automaton.

Comment. This is done with the graphs.

Requirement 8: Methodological separation of concern. – refers to a partitioning of methodological steps according to the process types they realize (general principle in software engineering).

Comment. In chapter 4 a clear an complete description of concepts is provided.

Requirement 9: Methodological extendibility – refers to the ability left to the designer to extend the development steps proposed in a methodology.

Comment. This can be done, as the set of transformational rules is not fix or closed, there is always a way to change the rules, the path, etc. also it is possible to extends this methodology, as we are already extending it to 3D UIs.

Requirement 10: Methodological Homogeneity – refers to the property of methodological steps of being defined using a common syntax**.** All transformation steps should be described in a single formalism that facilitates their understanding and processing.

Comment. By expressing every step in terms of UsiXML and graph nodes we achieve this homogeneity of term at all level of the methodology.

Requirement 11: Methodological reuse – refers to the possibility in a methodology to capitalize on the knowledge defined by designers to perform development steps and re-using this knowledge for other developments.

Comment. Task patterns is an example of how knowledge about the 3D UIs can be reused once encoded in the model.

Chapter 7 Conclusion

7.1 Context of This Work

Transformational development is one of the answers provided by the Software Engineering (SE) community to tackle the problem of building software in a systematic and principle-based way.

Transformational development in SE defines the development of software as a progressive refinement of abstract models into concrete models, until program code [Somm99]. This transformational development relies on catalogs of transformations able to (semi-)automatically perform model-to-model and model-to-code transformations.

Transformational development of user-interfaces (TDUI) specializes principles of transformational development in the context of UI development. By analogy with transformational development in SE, it defines the development of user interface systems as a successive application of transformations to an initial representation. This generally implies a progressive refinement of an abstract model into a concrete model, until program (here UI) code, or vice versa.

Not enough effort has been conducted to the development of 3D UIs as to its counter part in 2D. Such lack of methodologies so as the fact that actual software tool are more dedicated to design virtual content rather than 3DUIs.

UsiXML is a markup language that describes the User Interface at many levels, task, abstract, concrete, and consider concepts relevant to UI, such as the context of use, the domain, the dialog.

7.2 Content of This Dissertation

The state of the art of Chapter 2 reveals a series of shortcomings in existing approaches for achieving transformational development of 3D UIs. Also the languages, toolkits, and render engines were examined. The shortcomings delineated our problem space. These shortcomings lead us to conclude that we can be improved the way 3D UIs is done along several dimensions.

Chapter 3 presents a taxonomy of 3D UIs that covers actual ways to generate 3D UIs and that serves as a container way to organize examples found in the literature an then to generate the ontology of 3D UIs.

Chapter 4. Provides the solution to the shortcoming identified in chapter 2. For this purpose, this dissertation proposes (1) an ontological framework extending the actual description that considers just 2D UIs (2) a methodological framework, called model-based development, based on the ontological framework previously introduced. This methodological framework introduces a new paradigm for 3D UI development**.**

This development method decomposes any development activity (i.e, a development scenario) in a series of *development steps* consisting in the transformation of the artifact(s) in the scope of a *development stage* (here referred as *viewpoint*) into other development artifacts. In this context, a *development path* is defined as an archetypal composition of development steps. We identified two development paths: forward engineering, reverse engineering, even that we just show how to achieve one of those.

Chapter 5 introduces two case studies that showed the **feasibility** of developing a 3D UI in a model-based relying on explicit transformation catalogs at any time.

7.3 Summary of Contributions

A **model-based development method for 3D UIs.** The method based on graph transformation has been introduced, defined and illustrated. The ontology was extended in all the levels

- **Task Model**, new action task definitions were added. Some task patterns for the most common User task in virtual environments.
- **3D UIs Concrete Model**, the definition of new classes for concrete interaction objects, dividing the actual definition in 2D CIOs and 3D CIOs. The specification of a meta-model for 3D UIs.
- Context model, a **new environmental model** was introduced accordingly not just to the physical space but to the virtual space also.

The **mapping to achieve an Abstract model to 3D Concrete model transformation** was also introduced.

A 3D UIs Taxonomy. We propose a taxonomy that corresponds not just to more than mixed reality applications but also to a new reality of Web-based virtual applications, including the real world and 2D GUIs. The taxonomy proposed tries to cover not just the types of VR applications but also two of the most important sources of inspiration. One of the goals of this taxonomy is to provide design ideas when a certain kind of control want to be developed in a specific type of VR, then designers could see how have been done, if there is a solution, or, if not, how at other levels of the taxonomy authors solved the problem. Ideally a software tool with a repository of solutions that can be reused can be the best but the work to do that is really considerable. First, because of the hardware required to manipulate the VR application, second because of the programming language used, and third because of the information sharing, most solutions described in papers are not open source so there is no way to have access to their code. As a consequence, there is a need to start from scratch.

7.4 Future works

A lot of things remain to be done around the framework presented in this dissertation. We point out the following things as the most interesting issues for us:

- Extend the set of 3D UIs components. The actual description cover a small set of 3D widgets, we need not just to have the implementation but also explore more genuine objects, which is one of our goals.
- Define, develop, design and implement a transformation engine that support the whole path, and includes the mappings from CUI model to a high level virtual content editor.
- Design evaluation strategies based on the concepts managed by the methodology. It can be more easily to identify ergonomic rules, at a high level of the methodological path than on code.
- The interaction modeling (such as details of navigation and manipulation metaphors) is beyond the context of your current study. It

Dear reader, thank you very much for reading this dissertation.

References

A

[Abra99]
Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., and Shuster, J. "*UIML: An appliance-independent XML user interface language*". In Proceedings of 8th International World-Wide Web Conference WWWs, pp. 661-665, 1999.

[Anar06a]
Anark Corporation. *"Products: Anark Studio*". Available on: http://www.anark.com/products/products_studio.asp

[Anar06b]
Anark Corporation. *"Anark Gameface Evaluation Supplement*". Available on: http://update.anark.com/Gameface/EvaluationSupplement.html#examples

[Anne67]
Annett J., and Duncan K., *Task analysis and training design* in Occupational Psychology, 41, 1967, pp. 211-227.

[Alam02a]
Alambik © 1994 - 2OO2 ALAMBIK Ltd. "*Welcome-Introduction*". Available on: http://www.alambik.com/site/

[Alam02b]
Alambik © 1994 - 2OO2 ALAMBIK Ltd. "*What is Alambik?*". Available on: http://www.web-language.com/

[Alam03]
Alambik © 1994 - 2OO3 ALAMBIK Ltd. "*Alambik net viewer*". Available on: http://www.net-viewer.com/

[Auto06a]
Autodesk, Inc. "*Autodesk Inventor Series*". 2006. Available on: http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=5182857

[Auto06b]
Autodesk, Inc. "*Autodesk Maya*". 2006. Available on: http://usa.autodesk.com/adsk/servlet/index?id=6871843&siteID=123112

[Auto06c]
Autodesk, Inc. "*Autodesk Maya*". 2006. Available on: http://usa.autodesk.com/adsk/servlet/index?id=6871843&siteID=123112

[Azev00]
Azevedo, P., Merrick, P., and Roberts, D. "*OVID to AUIML user oriented interface modeling*". In Proceedings of 1st International Workshop Towards a UML Profile for Interactive Systems Development TUPIS00. York, October 2000.

B

[Bake06a]
Baker, M., J. "*EuclideanSpace - building a 3D world*". 2006. Available on : http://www.euclideanspace.com/

References

[Bake06b]
Baker, D. "*The database for freeware Max R8 plug-ins*". 2000-2006. Available on: http://www.maxplugins.de/max8.php

[Balz95]
Balzert, H., *From OOA to GUI - The JANUS-System*, in Nordbyn, K., Helmersen, P.H., Gilmore, D.J., Arnesen, S.A. (Eds.), Proceedings of the Fifth IFIP TC13 Conference on Human-Computer Interaction *INTERACT'95* (Lillehammer, June 25-29, 1995), Chapman & Hall, London, 1995, pp. 319–324.

[Bart88]
M.-F. BARTHET, "*Logiciels interactifs et ergonomie*", Dunod Informatique, Paris, 1988

[Bles90]
Bleser, Teresa W. & Sibert, John."*Toto: a tool for selecting interaction techniques*". In: *Proceedings of user interface software and technology (Snowbird, Utah, Oct.3-5,1990)* . New York: ACM, 1990, pp. 135-142.

[Blit06a]
Blitz Research Ltd. "*Blitz Research Home page*". Available online: http://www.blitzbasic.com/Home/_index_.php.

[Blit06b]
Blitz Research Ltd. "*About Blitz3D*". Available online: http://www.blitzbasic.com/Products/blitz3d.php.

[Blit06c]
Blitz Research Ltd. "*About BlitzMax*". Available online: http://www.blitzbasic.com/Products/blitzmax.php.

[Blit06d]
Blitz Research Ltd. "*Maplet*". Available online: http://www.blitzbasic.com/Products/maplet.php

[Boda95]
Bodart F. , Hennebert A. , Lheureux J. , Provot I. , Sacré B. , and Vanderdonckt J., "*Towards a systematic building of software architecture: The TRIDENT methodological guide".* In Proceedings of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (Vienna), Springer Verlag, 1995.

[Boeh88]
Boehm, B. "*A Spiral Model of Software Development and Enhancement.*" IEEE Communications 21, 5 (May 1988): 61-72

[Boms98]
Bomsdorf B. and Szwillus G., "*From task to dialogue: Task-based user interface design".* In *SIGCHI Bulletin 30*, 1998, pp. 40-42. Available online: http://www.acm.org/sigchi/bulletin/ 1998.4/szwillus.html.

[Boms99]
Bomsdorf, B., and Szwillus, G., "*Tool support for task-based user interface design*". In SIGCHI Bulletin, 31(4), 1999, pp. 27–29. Available online: http://www.uni-paderborn.de/cs/ag-szwillus/chi99/ws/

[Bori97]
Boritz, J. and Booth, k. "*A study of interactive 3D point location in a computer simulated virtual environment*", In Proceedings of VRST'97, pp. 181-187, 1997.

[Bosw02]
Boswell, D., King, B., Oeschger, I., Collins, P., and Murphy, E. "*Introduction to XUL".* In "Creating Applications with Mozilla", O'Reilly, Sebastopol, September 2002.

References

[Boui04]
Bouillon, L., Vanderdonckt, J., and Chow, K.C., Flexible Re-engineering of Web Sites in Proceedings of 8th ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, Portugal, 13-16 January 2004), ACM Press, New York, 2004, pp. 132-139.

[Bowm97a]
Bowman D. and Hodges L. "*An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments*", In Proceedings of Symposium on Interactive 3D Graphics, pp. 35-38, 1997.

[Bowm97b]
Bowman D. Koller, D., and Hodges L. "*Travel in Immersive Virtual Environments: an Evalution of Viewpoint Motion Control Techniques*", In Proceedings of Virtual Reality Annual International Symposium, 1997.

[Bowm98]
Bowman D. Wineman, J., Hodges L., and Allison, D. "*Desogning Animal Habitats Within an Immersive VE*". IEEE Computer Graphics and Applications, 18(5), 9-13. 1998.

[Bowm00]
Bowman D. "*The Science of Interaction Design*", In Doug Bowman, Ernst Kruijff, Joseph LaViola, Ivan Poupyrev, and Mark Mine, SIGGRAPH 2000 Course 3D User Interface Design: Fundamental Techniques, Theory, and Practice, 2000.

[Bowm01a]
Bowman D. "*Basic 3D Interaction Techniques*", In Doug Bowman, Ernst Kruijff, Joseph LaViola, Ivan Poupyrev, and Mark Mine, SIGGRAPH 2001 Course Advanced Topics in 3D User Interface Design, 2001.

[Bowm04]
D.A. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. "*3D User Interfaces: Theory and Practice*", Addison Wesley, Boston, July 2004.

[Bull03]
Bullard L., "*Extensible 3D: XML Meets VRML",* August 06, 2003, available on http://www.xml.com/pub/a/2003/08/06/x3d.html.

[Byrne92]
Byrne E. J. "*A conceptual foundation for software re-engineering".* In Proceedings of the Conference on Software Maintenance, IEEE Computer Society Press, November 1992 pages 216--235.

C

[Calh84]
Calhoun, G. C.; Arbak, C. L. & Boff, K. R. "Eye-controlled switching for crew station design". In: *Proceedings of the Human Factors Society 28th annual meeting*, Santa Monica (CA): Human Factors Society, 1984, pp. 258-262.

[Calv03]
G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. "*A Unifying Reference Framework for Multi-Target User Interfaces*". Interacting with Computers, 15(3): 289–308, 2003.

[Card83]
Card S.K., Moran T.P., and Newell A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, New York, 1983.

[Care97]

References

Carey R., Bell G., "*The annotated VRML 2.0 Reference Manual*", available on http://accad.osu.edu/~pgerstma/class/vnv/resources/info/AnnotatedVrmlRef/Book.html, (1997).

[Carn06]
Carnegie Mellon University. "*Alice: Learn to Program Interactive 3D Graphics*". Available on: http://www.alice.org/

[Cele01]
A. Celentano and F. Pittarello (2001). "*A content centered methodology for authoring 3d interactive worlds for cultural heritage*". D Bearman, F Garzotto, Eds., International Cultural Heritage Informatics Meeting, ICHIM, Cultural Heritage and Technologies in the Third Millennium, Milán, 3-7 September 2001, Vol. 2 pp. 315-324, Politecnico di Milano, Italia y Archives & Museum Informatics, Pittsburgh, PA, USA, Italia, 2001.

[Chap05]
O. Chapuis and N. Roussel. "*Metisse is not a 3D desktop*". In Proc. of ACM Conf. on User Interface Software Technology UIST'2005 (October 2005). ACM Press, New York, 2005.

[Chik90]
Chikofsky E.J. and Cross J.H., *Reverse Engineering and Design Recovery: A Taxonomy* in IEEE Software, 1(7), January 1990, pp. 13-17.

[Cock00]
Cockburn, A. and McKenzie, B. "*An evaluation of cone trees*". In People and Computers XV (Proceedings of the 2000 British Computer Society Conference on Human-Computer Interaction.), University of Sunderland, September 2000, Springer-Verlag, In Press.

[Cock01]
Cockburn, A. and McKenzie. "*3D or not 3D? Evaluating the Effect of the Third Dimension in a Document Management System*". In Proceedings of the SIGCHI conference on Human factors in computing systems. Seattle, Washington, United States. Pages: 434 – 441, 2001. Available on: http://www.cosc.canterbury.ac.nz/andrew.cockburn/papers/chi01DM.pdf

[Conn92]
DB Conner, SS Snibbe, KP Herndon, DC Robbins, RC Zeleznik, A van Dam. "*Three-Dimensional Widgets*". Special Issue of Computer Graphics (Proceedings of the 1992 Symposium on Interactive 3D Graphics), pages 183-188. ACM Press.

[Cons03]
Constantine L. L., "*Canonical Abstract Prototypes for Abstract Visual and Interaction".* In Proceedings of the 10th International workshop on Design, Specification and Evaluation of Interactive Systems DSV-IS 2003 (june 11-13 2003, Funchal, Portugal), LNCS 2844, Springer Verlag, Berlin, 2003, pp. 1-15.

[Corr97]
Corradini, Ehrig H. , Heckel R. , Korff M. , Löwe M. , Ribeiro L. , and Wagner A., *Algebraic approaches to graph transformation - part I: Single pushout approach and comparison with double pushout approach*, in Rozenberg G. (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations, World Scientific,1997, pp. 247-312.

[Craz05]
Crazy Eddies wiky page. "*Crazy Eddie's GUI System Namespace List*". 2005. Available on: http://www.cegui.org.uk/api_reference/namespaces.html

[Craz06]
Crazy Eddies web site. "*Welcome to Crazy Eddie's GUI System*". 2006. Available on: http://www.cegui.org.uk/wiki/index.php/Main_Page

[Cupp04]
E. Cuppens, Ch. Raymaekers, and K. Coninx. "*Vrixml: A User Interface Description Language for Virtual Environments"*. In Proc. of the 1st ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" UIXML 2004 (Gallipoli, May 25, 2004), pages 111–118, 2004.

[Cupp05]
E. Cuppens, and K. Coninx. "*CoGenIVE: Code Generation for Virtual Environments"*. In Proc. of the the Future of User Interface Design Tools, workshop of ACM Conference on Human Factors in Computing Systems (CHI 2005) (Portland, Apr 04, 2005).

[Czar00]
Czarnecki, K., and Eisenecker, U.W., "*Generative Programming: Methods, Tools, and Applications"*. In Addison-Wesley, Reading, 2000.

D

[Dach01]
Dachselt, R., Ebert, J. "*Collapsible Cylindrical Trees: A Fast Hierarchical Navigation Technique"*, infovis, vol. 00, no. , p. 79, IEEE 2001.

[Dach02]
Dachselt, R., Hinz, M. and Meißner, K. 2002. CONTIGRA: "*An XML-Based Architecture for Component-Oriented 3D Applications*". In Proceedings of 7th International Conference on 3D Web Technology Web3D'2002 (Tempe, February 24-28, 2002). ACM Press, New York, 155–163.

[Dach03]
Dachselt, R. and Rukzio, E. „*Behavior3d: an xml-based framework for 3d graphics behavior"*. In Proceeding of the eighth international conference on 3D web technology, pages 101–ff. ACM Press, 2003.

[Diet01]
Diettrich O. "*Virtual Reality and cognitive process*". In: Riegler, A., Peschl, M., Edlinger, K., Fleck, G. & Feigl, W. (eds.) Virtual Reality: Cognitive Foundations, Technological Issues & Philosophical Implications. Frankfurt am Main: Peter Lang. 2001.

[Dijk76]
Dijkstra, E. W., *The discipline of programming*, Prentice Hall, Engelwood Cliffs, NJ, 1976.

[Dsou99]
D'Souza, D.F., and Wills, A.C. "*Objects, Components and Frameworks with UML: The Catalysis Approach*". Addison-Wesley, Reading, 1999.

E

[Edli01]
Edlinger K. "*Virtual Reality, Cyberspace and Living Organisms. Towards a new understanding of perception and cognition?*". In: Riegler, A., Peschl, M., Edlinger, K., Fleck, G. & Feigl, W. (eds.) Virtual Reality: Cognitive Foundations, Technological Issues & Philosophical Implications. Frankfurt am Main: Peter Lang. 2001.

[Ehri99]
Ehrig, H., Engels, G., Kreowski, H-J., and Rozenberg, G. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation, Application, Languages and Tools*, Vol. 2, World Scientific, Singapore, 1999.

[Eise01]

Eisenstein, J., Vanderdonckt, J., and Puerta, A. "*Model-based User-Interface Development Techniques for Mobile Computing*". In Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001. (Santa Fe, 14-17 January 2001), ACM Press, New York, 2001, pp. 69-76.

F

[Fair93]

Fairchild, K.M. "*Information Management Using Virtual Reality-Based Visualizations*". In Virtual Reality: Applicationsand Explorations", Alan Wexelblat (ed.), Academic Press Professional, Cambridge, MA, pp. 45-74, 1993.

[Fenc99]

C Fencott (1999) "*Towards a Design Methodology for Virtual Environments*". User Centered Design and Implementation of Virtual Environments (UCDIVE) Workshop. University of York, 30 September 1999.

[Fenc01]

C Fencott y J Isdale (2001) "*Design Issues for Virtual Environments"*. International Workshop on Structured Design of Virtual Environments and 3D-Components at athe Web3D 2001 Conference. Paderborn, Alemania, 19 Febrero 2001.

[Figu02]

Figueroa, P.; Green, M.; Hoover, H. J. "*InTml: A Description Language for VR Applications*". Web3D'02, February 24-28, 2002. Tempe, Arizona, USA.

[Figu04]

Figueroa, P. "*Retargeting of Virtual Reality Applications*". PH. D. Thesis. University of Alberta Canada, 2004.

[Figu06]

Figueroa, P., Dachselt, R., Lindt, I. "*A Uniform Specification of Mixed Reality Interface Components*". Proceedings of the IEEE Virtual Reality Conference 2006, March 25-29, 2006, Alexandria (Virginia, USA), pp. 289-290.

[Fitz95]

Fitzmaurice, G. W.,, Ishii, H., Buxton, W., "*Bricks : Laying the Fundations for Graspable User Interfaces*". In Proceedings of CHI'95 Conference on human Factors in Computing Systems, 442-449, New York, ACM, 1995.

[Fole84]

Foley, V. Wallace and V. Chan, "*The human factors of computer graphics interaction techniques*", In IEEE Computer Graphics & Applications, (4), pp. 13-48 (1984).

[Fokk92]

Fokkinga M.M,. "*A gentle introduction to category theory: the calculational approach"*. In Lecture Notes of the STOP 1992 Summerschool on Constructive Algorithmics, University of Utrecht, September 1992, pp. 1-72.

G

[Gamm95]

Gamma E., Helm R., Johnson R., Vlissides J. and Booch G. "*Design Patterns : Elements of Reusable Object-Oriented Software"*, Addison-Wesley Professional Computing (1995).

[Geig01]

Geiger, C., Paelke, V., Reimann, C. C., and Rosenbach, W. "*Structured Design of Interactive Virtual and Augmented Reality Content*". International Workshop on Structured Design of Virtual Environments and 3D-Components at the Web3D 2001 Conference. Paderborn, Alemania, 19 February 2001.

[Geno04]

Genova development Environment, Genera, Trondheim, Norway, 2004. Available online: http://www.genera.no.

[Glas84]

Glasersfeld, E. v. "*An introduction to radical constructivism*". In P. Watzlawick (Ed.), The Invented Reality, pp. 17-40. New York: Norton, 1984.

[Glas91]

Glasersfeld, E. v. "*Knowing without metaphysics. Aspects of the radical constructivism Position*". In F. Steier (Ed.), Research reflexivity, pp. 12-29. London; Newburry park, CA: SAGE Publishers. 1991.

[Glas95]

Glasersfeld, E. v. "*Radical constructivism: a way of knowing and learning*". London: Falmer Press. 1995.

[Glas03]

Glasersfeld, E. v. "*An Exposition of Constructivism: Why Some Like it Radical*". In the internet encyclopedia of personal construct psychology of the Scientific Reasoning Research Institute. 2003 Available on http://www.oikos.org/constructivism.htm.

[Göbe96]

Göbel 1996, M. Göbel, "*Industrial Applications of VEs*", IEEE computer graphics and applications, 16(1), pp 10-13, 1996.

[Gome04]

Gomes de Sousa, L., and Leite, J.C. "*XICL- An Extensible Mark-up Language for Developing User Interface and Components*". In Proceedings of the Fifth International Conference on Computer-Aided Design of User Interface CADUI'2004.

[Grac05]

D. Gracanin and J. Ying, "*An approach to formal description of user interfaces based on X3D content*". In Advances in Virtual Environments Technology: Musings on Design, Evaluation, & Applications (K. Stanney and M. Zyda, eds.), Human Factos/Ergonomics Series, Lawrence Erlbaum Associates, 2005.

[Gree88]

Greenstein, Joel S. & Arnaut, Lynn Y. "*Input devices*". In: M. Helander, (Ed.), Handbook of Human-Computer Interaction, Amsterdam: North-Holland, 1988, pp. 495-519.

[Gros99]

Grosjean, J. - Coquillart, S. "*The Magic Mirror: A Metaphor for Assisting the Exploration of Virtual Worlds*". Pp. 125-129 in Proceedings of 15th Spring Conference on Computer Graphics. Bratislava: Comenius University 1999.

[Grub93]

Gruber T. R., *A translation Approach to Portable Ontologies,* in Knowledge Acquisition, 5(2), 1993, pp. 199-220.

[Guer06]

Guerrero Garcia, J. "*Conseptual Modeling of User Information systems of Workflow Information Systems*". In depth studies thesis. University Catholic of Louvain, 2006.

[Guil95]

Guillen, M. "Five equations that change the world: the power and Poetry of Mathematics". New York: hyperion, 1995.

H

[Heck02]
Heckel, R., Küster, J.M., Taenzer, G., Confluence of Typed Attributed Graph Transformation Systems, in [Corr02], pp. 161-176.

[Heim93]
Heim M. "*The Metaphysics of Virtual Reality*". Oxford press, 1993.

[Hewe92]
Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank, "*ACM SIGCHI Curricula for Human-Computer Interaction*". 1992,1996 ACM SIGCHI

[Hirs89]
Hirschheim, R., Klein, H. K. "*Four Paradigms of Information Systems Development*". In Rob Kling (ed), Social Aspects of Computing. Communications of the ACM, volume 32, issue 10. October 1989.

[Heyl93]
Heylighen, F. "*Epistemology, introduction. Principia Cybernetica*", 1993. Available on http://pespmc1.vub.ac.be/EPISTEMI.html.

[Hoff03]
Hoffmann, H.; Dachselt, R.; Meißner, K.: "*An Independent Declarative 3D Audio Format on the Basis of XML*". In Proceedings of the 2003 International Conference on Auditory Display, Boston, MA, USA, 6-9 July 2003

[Hutc89]
Hutchinson, Thomas E.; White, Jr., K.Preston; Martin, Worthy N.; Reichert, Kelly N. & Frey, Lisa A. "*Human-Computer Interaction Using Eye-Gaze Input*". IEEE Transactions on systems, man, and cybernetics, 19(6), 1989, p. 1527-1533.

I

J

[John92]
Johnson P., Markopoulos P., and Johnson H., "*Task knowledge structures: A specification of user task models and interaction dialogues".* In Proceedings of Task Analysis in Human-Computer Interaction, 11th Int. Workshop on Informatics and Psychology (Schraeding, June 9-11), 1992.

[John01]
Johnson M., and Dampney, *O. "category theory as a (meta) ontology for information systems".* In Proceedings of the international conference on Formal Ontology in Information Systems FOIS 01 (Ogunquit, Maine, USA), 2001, pp. 59-69.

[Jorg99]
Jorgensen, Charles, Kevin Wheeler, and Slawomir Stepniewski. "*Bioelectric Control of a 757 Class High Fidelity Aircraft Simulation*". Available on: http://ic.arc.nasa.gov/publications/index.html, 1999.

K

[Kats03]
K. Katsurada, Y. Nakamura, H. Yamada, and T. Nitta. XISL: A Lan-guage for Describing Multimodal Interaction Scenarios. In Proc. of 5th Int. Conf. on Multimodal Interfaces ICMI'2003 (Vancouver, 5-7 November 2003), ACM Press, New York, pages 281–284, 2003.

[Khaz00]
Khazanchi D., Munkvold B. E. "Is Information Systems a Science? An Inquiry into the Nature of the Information Systems Discipline". The DATA 24 BASE for Advances in Information Systems. (Vol. 31, No. 3). Summer, 2000.

[Kiyo00]
Kiyokawa, K., Takemura, H., Yokoya, N., "*Seamless Design for 3D object creation*". IEEE multimedia, pp. 22-33, 2000.

[Krei01]
Kreitler, S. "Psychological Perspective on Virtual Reality". In: Riegler, A., Peschl, M., Edlinger, K., Fleck, G. & Feigl, W. (eds.) Virtual Reality: Cognitive Foundations, Technological Issues & Philosophical Implications. Frankfurt am Main: Peter Lang. 2001.

[Krui00]
Kruijff, E. "*Wayfinding*". In Doug Bowman, Ernst Kruijff, Joseph LaViola, Ivan Poupyrev, and Mark Mine, SIGGRAPH 2000 Course 3D User Interface Design: Fundamental Techniques, Theory, and Practice, 2000.

L

[Lari03]
D. Larimer and D. Bowman. Vewl: A Framework for Building a Windowing Interface in a Virtual Environment. In Proc. of IFIP TC13 Int. Conf. on Human-Computer Interaction Interact'2003 (Zürich, Sept. 1-5, 2003), IOS Press, Amsterdam, pages 809–812, 2003.

[Leno84]
Lenorovitz, D.R.; Phillips, M.D.; Ardrey, R.S. & Kloster, G.V. "*A taxonomic approach to characterizing human-computer interaction*". In: G. Salvendy (Ed.), Human-Computer Interaction. Amsterdam: Elsevier Science Publishers, 1984, pp.111-116.

[Limb04a]
Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V., "*USIXML: a Language Supporting Multi-Path Development of User Interfaces*". In Proceedings of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Kluwer Academic Press, Dordrecht, 2004.

[Limb04b]
Limbourg, Q., Vanderdonckt, J., UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence, in Matera, M., Comai, S. (Eds.), «Engineering Advanced Web Applications», Rinton Press, Paramus, 2004, pp. 325-338. Available on http://www.isys.ucl.ac.be/bchi/publications/2004/Limbourg-JWE2004.pdf

[Limb04c]
Limbourg, Q., "*Multi-Path Development of User Interfaces*". PhD thesis, University Catholic of Louvain. Louvain-la-Neuve, Belgique. 2004. Available on: http://www.isys.ucl.ac.be/bchi/publications/Ph.D.Theses/Limbourg-PhD2004.pdf

[Luo95]

Luo, P., "*A Human-Computer Collaboration Paradigm for Bridging Design Conceptualization and Implementation*". In Paternò F. (Ed.), Interactive Systems: Design, Specification, and Verification, Proc. of the 1st Eurographics Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'94, (Bocca di Magra, June 8-10, 1994), Springer-Verlag, Berlin, pp. 129–147.

[Luyt04]

Luyten, K., Abrams, M., Limbourg, Q., Vanderdonckt, J., Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" UIXML'04 (Gallipoli, May 25, 2004), Gallipoli, 2004.

M

[Mape95]

Mapes, D. and Moshell, J. "*A Two-Handed Interface for Object Manipulation in virtual Environments*". In Proc. Presence: Teleoperators and Virtual Environments, 4(4), 403-426. 1995.

[Maqu04]

Maquil, V. "*Automatic Generation of Graphical User Interfaces in Studierstube*". Bachelor thesis. Interactive Media Systems Group, Institute for Software Technology and Interactive Systems, Vienna University of Technology, 2004. Available on https://www.ims.tuwien.ac.at/media/documents/publications/MaquilBachelorThesis.pdf.

[Marq97]

Marquis J.-P., *Stanford encyclopedia of philosophy: Category theory*, 1997. Available online: http://plato.stanford.edu/entries/category-theory/.
[Meye97]
Meyer B., *Object-Oriented Software Construction*, Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1997.

[Mens99]

Mens T., *A Formal Foundation for Object-Oriented Software Evolution*, PhD thesis, Vrije Universiteit Brussel, 1999.

[Mile79]

Miles, M., B., "*Qualitative Data as an Attractive Nuisance: The Problem of Analysis*". In Administrative Science Quarterly, vol. 24, no. 4, Qualitative Methodology. pp. 590-601. 1979.

[Milg94]

Milgram, P. and Kishino, F. "*A Taxonomy of Mixed-Reality Visual Displays*". In IEICE Transactions on Informations Systems, vol. E77-D, no. 12. Available on: http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html.

[Mill03]

Miller, J., Mukerij J., "*MDA Guide version 1.0.1*", 2003. Available online : www.omg.org.

[Mine95]

Mine M., "*Virtual environment interaction techniques*", UNC Chapel Hill CS Dept., Technical Report TR95-018, 1995.

[Moli02]

Molina, P.J., Meliá, S., Pastor, O,. "*Just-UI: A User Interface Specification Model*", in: Kolski C., Vanderdonckt J. (Eds.), Computer-Aided Design of User Interfaces III, Kluwer Academic Publishers, Dordrecht, 2002, pp. 63-74.

[Moli05]

Molina Massó J.P., Vanderdonckt J., Montero Simarro F., González López P. "*Towards Virtualization of User Interfaces based on UsiXML*". Proceedings of Web3D 2005 Symposium, 10th International Conference on 3D Web Technology (March 29 – April 2005, University of Wales, Bangor, UK). A publication of ACM SIGGRAPH, ACM ISBN: 1-59593-012-4. Pp. 169-179, 2005.

[Moli06]
Molina Massó J.P., Vanderdonckt J., González López P., Fernández Caballero, A., Lozano Pérez, D. "*Rapid Prototyping of Distributed User Interfaces*". Chapter 12, in G. Calvary, C. Pribeanu, G. Santucci, J. Vanderdonckt (eds.), "Computer-Aided Design of User Interfaces V", Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Information Systems Series, Springer-Verlag, Berlin, 2006, pp. 85-100.

[Mont70]
Montanari, U.G., *Separable Graphs, planar Graphs and Web Grammars*, in Inf. Contr., 16, 1970, pp. 243-267.

[Mont05]
F. Montero, V. López-Jaquero, J. Vanderdonckt, P. Gonzalez, and M.D. Lozano. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSVIS'2005 (Newcastle upon Tyne, July 13–15, 2005), Springer-Verlag, Berlin, 2005, to appear.

[Murp97]
Murphy E., "Constructivism from Philosophy to Practice". 1997. Available on http://www.cdli.ca/~elmurphy/emurphy/cle.html.

[Myer92]
Myers, B., Rosson, M. "*Survey on User Interface Programming*". Proceedings CHI'92, New York, ACM, 1992, pág. 195-202.

N

[Nade97]
Nadeau, D. R. Introduction to VRML 97. Eurographics 97, tutorial notes sections, 1997.

[Neal01]
H Neale and S Nichols (2001) Designing and developing Virtual Environments: methods and applications. Visualization and Virtual Environments Community Club (VVECC) Workshop, Designing of Virtual Environments. 2001.

[Niga95]
Nigay L., Coutaz J., *A Generic Platform for Addressing the Multimodal Challenge*, in Proceedings of CHI'95, ACM Press, New York, 1995, pp. 98-105.

O

[Ogre05]
The Ogre team. "*About OGRE*". 2005. Available on:
http://www.ogre3d.org/index.php?option=com_content&task=view&id=19&Itemid=79

[Open04]

OpenGL The industry standard for 2D and 3D Graphics. "*OpenGL Overview*". 2004. Available on: http://www.opengl.org/about/overview.html

P

[Pate97]
Paternò F. , Mancini C. , and Meniconi S., "*ConcurTaskTree: A diagrammatic notation for specifying task models*". In Howard S. , Hammond J. , and Lindgaard G. (Eds.), Proceedings of IFIP TC 13 International Conference on Human-Computer Interaction Interact'97 (Sydney, July 14-18, 1997), Kluwer Academic Publishers, Boston, 1997, pp. 362-369.

[Pate00]
Paternò F. "*Model-Based Design and Evalution of Interactive Applications*", Applied computing, Springer, 2000.

[Paus95]
Pausch, R., T. Burnette, D. Brockway and M.E. Weiblen. "*Navigation and locomotion in virtual worlds via flight into hand-held miniatures.*" In Proceedings of SIGGRAPH 95, Los Angeles, CA, ACM: 399-400.

[Park98]
Parker, G., Franck, G. and Ware, C. "*Visualization of Large Nested Graphs in 3D: Navigation and Interaction*". In J. Visual Languages and Computing, 9(3):299--317, 1998.

[Pell05a]

Pellens, B. Olga De Troyer, Wesley Bille, Frederic Kleinermann, Raul Romero. "*An Ontology-Driven Approach for Modeling Behavior in Virtual Environments*". OTM Workshops 2005: 1215-1224

[Pell05b]
Pellens, B., Bille, W., De Troyer, O., Kleinermann, F.: "*VR-WISE: A Conceptual Modelling Approach For Virtual Environments*", In CD-ROM Proceedings of the Methods and Tools for Virtual Reality (MeTo-VR 2005) workshop, Ghent, Belgium (2005)

[Pesc01a]
Peschl M. F., Riegler A. "*Virtual Science. Virtuality and knowledge acquisition in science and cognition*". In: Riegler, A., Peschl, M., Edlinger, K., Fleck, G. & Feigl, W. (eds.) Virtual Reality: Cognitive Foundations, Technological Issues & Philosophical Implications. Frankfurt am Main: Peter Lang. 2001.

[Pesch1b]
Peschl M. F. "Constructivism, Cognition, and Science-An Investigation of its Links and Possible Shortcomings". In: Riegler (ed.) Foundations of Science, special issue on "The impact of Radical Constructivism on Science". Vol 6, no. 1-3: 125-161. 2001.

[Pier97]
Pierce, J. S., Forsberg, A. S., Conway, M. J. "*Image Plane Interaction Techniques in 3D Immersive Environments*". In Proceedings of the 1997 symposium on Interactive 3D graphics, pp. 39-ff, 1997.

[Poup96]
Poupyrev, I., Billinghurst, M., Weghorst, S., Ichikawa, T. "*The Go-Go Interaction Technique: NonLinear Mapping for Direct Manipulation in VR*". In Proc. UIST'96, pp. 79--80.

[Poup98]

Poupyrev, I., Weghorst, S., Billinghurst, M., Ichikawa, T., "*Egocentric Object Manipulation in Virtual Environments : Empirical evaluation of Interaction Techniques*". In EUROGRAPHICS' 98, vol 17, no. 3, 1998.

[Poup00]
Poupyrev I. "*3D Manipulation Techniques*", In Doug Bowman, Ernst Kruijff, Joseph LaViola, Ivan Poupyrev, and Mark Mine, 3D UI Course Notes, 2000.

[Poup01]
Poupyrev, I. "*3D Interaction Techniques and Metaphors*" In Doug Bowman, Ernst Kruijff, Joseph LaViola, Ivan Poupyrev, and Mark Mine, 3D UI Course Notes, 2001.

[Puer97]
Puerta, A.R., *A Model-Based Interface Development Environment*, in IEEE Software 14(4), 1997, pp. 41–47. Available online: http://www.arpuerta. com/pubs/ieee97.htm

[Puer02]
Puerta, A., and Eisenstein, J. XIML: "*A common representation for interaction data*". In Proceedings of the 7th International Conference on Intelligent User Interfaces, pp. 69-76. ACM Press, January 2002.

R

[Raut00]
Rauterberg G. W. M. "*How to characterize a research line for user-system interaction*". In IPO Annual Progress Report 35, 2000.

[Reki95]
Rekimoto, J., Nagao, K., "*The World through the Computer: Computer Augmented Interaction with Real World Environments*". In Proceedings of IUST'95. ACM, pp. 29-36, 1995.

[Reki99]
Rekimoto, J., Saitoh, M., "*Augmented surfaces: A spatially continuous work space for hybrid computing environments*". In Proceedings of CHI'99. 1999. ACM. pp. 378-385.

[Robe83]
Roberts N., Andersen D., Deal R., Garet M., and Shaffer W. "*Introduction to Computer Simulation, a system dynamics modeling approach*". Productivity press. 1983.

S

[Shne02]
Shneiderman B. "*3D or Not 3D: When and Why Does it Work?*". Human-Computer Interaction Laboratory & Department of Computer Science University of Maryland. Talk in Web3D. Phonix, February 26, 2002.

[Shne03]
Shneiderman B. "*Why Not Make Interfaces Better than 3D Reality*". Virtualization Viewpoints. Editor: Theresa-Marie Rhyme, November-December, 2003.

[Sili03]
Silicon Graphics. "*Open Inventor web site*". 2003. Available on : http://oss.sgi.com/projects/inventor/

[Somm99]
Sommerville, I. "*Software Engineering*", 5th edition, Addisson Wesley, 1999.

[Souc03]

Souchon, N. and Vanderdonckt, J. "*A review of XML-compliant user interface description languages*". DSV-IS2003, 2003.

[Sowa92]

Sowa J. F., "*Conceptual Graphs Summary*", in Eklund P., Nagle T., Nagle J., and Gerholz L. (Eds.), Conceptual Structures: Current Research and Practice, Ellis Horwood, 1992, pp. 3-52.

[Spai91]

Spain E., Holzhauzen K. "*Stereoscopic versus orthogonal view displays for performance of a remote manipulation task*", Proceedings of Stereoscopic Displays and Applications II, SPIE, pp. 103-110. 1991.

[Sphe05]

Sphere site. Available at: http://www.spheresite.com/

[Stoa95]

Stoakley, R., M.J. Conway and R. Pausch. "*Virtual Reality on a WIM: Interactive Worlds in Miniature.*" Proceedings of CHI 95, Denver, CO, ACM: 265-272. 1995.

[Stan06]

Stanciulescu, A. "*A Transformational approach for developing multimodal web interfaces"*. Master Thesis, University catholic of Louvain, 2006. To appear.

[Stre86]

Streibel, M. J. "*A critical analysis of the use of computers in education*". Educational Communications and Technology Journal, 34 (3), 1986.

[Stud]

Studierstube Augmented Reality Project, http://www.studierstube.org/.

[Sutc03]

A Sutcliffe. "*Multimedia and Virtual Reality: Designing Multisensory User Interfaces*". Lawrence Erlbaum Associates, 2003.

[Sun05]

Sun Micro systems. "*The Java^{TM} tutorial: Listeners Supported by Swing Components*". Available on:

http://java.sun.com/docs/books/tutorial/uiswing/events/eventsandcomponents.html

T

[Tan01]

D. S. Tan, G. G. Robertson, and M. Czerwinski, "*Exploring 3D navigation: Combining speed-coupled flying with orbiting*". In Conference on Human Factors in Computing Systems CHI 2001.

[Teor86]

Teory T.J., Yang D., Fry J.P. "*A Logical Design Methodology for Relational Databases using the Extended Entity-Relationship Model*", ACM computing surveys 18(2), june 1986, pp. 197-222.

[Thev01]

Thevenin, D. "*Adaptation en Interaction Homme-Machine: le cas de la Plasticité*", Ph.D. thesis, Université Joseph Fourrier, Grenoble, France, 2001. Available online: http://iihm.imag.fr/publs/2001.

[Thié03]

Thiétart, R.A. et coll. "*Méthodes de Recherche en Management*" . Dunod, Paris. 2003.

U

[Unde98]
Underkoffler, J., Ishii, H., "*Illuminating light: an optical design tool with a luminous-tangible interface*". In Proceedings of CHI'98. ACM. pp. 542-549, 1998.

[USIX06]
UsiXML Consortium. UsiXML, a General Purpose XML Compliant User Interface Description Language, UsiXML V1.6.4, 1 March 2006. Available at http://www.usixml.org/index.php?view=page&idpage=6

V

[Verj95]
Verjans, S. "*State-of-the-art for Input Modalities*". Esprit BRA AMODEUS-II Working Paper TM-WP20. 1995. Available on http://perswww.kuleuven.be/~u0003438/.

[vonG03]
von Glasersfeld, E. "*An Exposition of Constructivism: Why Some Like it Radical*". In the internet encyclopedia of personal construct psychology of the Scientific Reasoning Research Institute. 2003 Available on http://www.oikos.org/constructivism.htm

W

[W3C95]
W3C consortium, *"VRML Virtual Reality Modeling Language",* 17 April 1995. Available at http://www.w3.org/MarkUp/VRML/

[Wals95a]
Walsham, G. "*Interpretive Case Studies in IS Research: Nature and Method*". European Journal of Information Systems, Vol. 4, pp. 74-81, 1995.

[Wals95b]
Walsham, G. "*The Emergence of Interpretivism in IS Research*" Information Systems Research, Vol. 6, No.4, pp. 376-394, 1995b.

[Wang05]
Wang, S., Poturalski, M., Vronay, D., "*Designing a Generalized 3D Carousel View*". In CHI 2005, april 2-7, Portland, Oregon, USA, 2005.

[Wate93]
J.A. Waterworth and L. Serra, "*VR Management Tools: Beyond Spatial Presence*". In Proc. of ACM Conf. on Human Aspects in Computing Systems Interchi'93 (Amsterdam, April 24-29, 1993), Addison-Wesley, Reading, pages 319–320, 1993.

[Web304a]
Web3D Consortium. "*Information technology — Computer graphics and image processing — Extensible 3D (X3D)*". Available on http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/

[Web304b]
Web3D Consortium. "*X3D Frequently Asked Questions*". Available on http://www.web3d.org/x3d/faq/

[Wiki05]

Wikipedia the free encyclopedia available on http://en.wikipedia.org/wiki/Main_Page (Accessed August 11, 2005).

[Wils78]
Wilson, E. O. "*On Human hature*". Cambridge, MA: Harvard University Press. 1978.

[Wils97]
Wilson, B. "*The postmodern paradigm*". In C. R. Dills and A. Romiszowski (Eds.), Instructional development paradigms. Englewood Cliffs NJ: Educational Technology Publications, 1997. Also available on: http://www.cudenver.edu/~bwilson/postmodern.html.

Z

[Zaki]
Zakiul, S. "*Week 15 report on Project 6*", Available on: http://www.public.asu.edu/~zakiul/vrml/week15/week15.htm

Annex A. Lotos Operators

Concur task tree CTT combines hierarchical structuring of tasks to temporal ordering of elements with a subset of LOTOS operators. LOTOS is a grounded formal notation in software engineering for specifying the ordering of processes in time [Pate97].

As proposed in [USIX06] we use LOTOS operators for concrete user interface **dialog control**. For each operator [Limb04c] defined what a task requires to be initiated and what it ensures. A termination condition is also provided for each operator. This condition tells when two temporally related tasks can be considered as terminated.

- Enabling (T1 has to be finished in order to initiate T2)

T1 >> T2	T1	Requires: Ø
		Ensures: ?
	T2	Requires: T1.Termination
		Ensures : ?
	Termination	T1.Termination AND T2.Termination

- Non-deterministic choice (Once one task is finished the other cannot be accomplished anymore)

T1 π T2	T1	Requires: NOT (T2.Termination)
		Ensures: ?
	T2	Requires: NOT (T1.Termination)
		Ensures : ?
	Termination	T1.Termination XOR T2.Termination

- Deterministic Choice (Once one task is initiated, the other cannot be accomplished anymore)

T1 [] T2	T1	Requires: NOT (T2.Initiation)
		Ensures: ?
	T2	Requires: NOT (T1.Initiation)
		Ensures: ?
	Termination	T1.Termination XOR T2.Termination

- Parallelism (T1 is interleaved with T2)

T1 ||| T2	T1	Requires: Ø
		Ensures: ?
	T2	Requires: Ø
		Ensures : ?
	Termination	T1.Termination AND T2.Termination

- Sequential independence (Is equivalent to (T1>>T2) OR (T2 >>T1))

T1 |=| T2	T1	Requires: NOT(T2.Initiation) XOR T2.Termination
		Ensures: ?
	T2	Requires: NOT(T1.Initiation) XOR T1.Termintaion
		Ensures : ?
	Termination	T1.Termination AND T2.Termination

- Deactivation (T2 may interrupt T1 before the termination of T1;. T1 cannot be resumed after T2 has terminated.)
-

T1 [> T2	T1	Requires: Ø
		Ensures: Ø
	T2	Requires: T1.Initiation AND NOT(T1.Termination)
		Ensures: ?
	Termination	Termination XOR T2.Termination

- Suspend/Resume (T2 may interrupt T1 before the termination of T1. Once T2 is finished, T1 may be resumed.)

T1 |> T2	T1	Requires: Ø
		Ensures: ?
	T2	Requires: T1.initiation
		Ensures: ?
	Termination	T1.termination OR T2.Termination

- Enabling with information passing (T1 has to be finished in order to initiate T2 and T2 is synchronized with T1 on some piece of data)

T1 >> T2	T1	Requires: Ø
		Ensures: ?
	T2	Requires: T1.termination and dataSynchronized
		Ensures : ?
	Termination	T1.termination AND T2.Termination

- Parallelism with information passing (T1 is interleaved with T2 while they synchronize on some data)

T1 |[]| T2	T1	Requires: NOT(T2.initiated) OR dataSynchronized
		Ensures: ?
	T2	Requires: NOT(T1.initiated) OR dataSynchronized
		Ensures : ?
	Termination	T1.termination AND T2.Termination

- T* (Iteration). T can be iterated an infinite number of times
- T(n) (Finite Iteration). T can be iterated n times
- [T] (optional). T is optional

Annex B. Graph Definitions

Definition 1. A **graph** g is defined by a quadruple $(V_g, E_g, source_g, target_g)$ such that:

1. V_g is a finite set of vertices (or nodes);
2. E_g is a finite set of edges (or arcs);
3. $source_g$: E $\rightarrow$ V, is an injective function that assigns a source to each edge of E;
4. $target_g$: E $\rightarrow$ V, is an injective function that assigns a target to each edge of E.

Definition 2. g is said to **directed** iff
$\forall e \in E_g, \exists ! v_i, v_j \in V_g \mid source(e) = v_i \wedge target(e) = v_j$

Notation 1. Implicit graph reference. The notation $[SetName]_g$ (e.g. V_g) or $[FunctionName]_g$ (e.g. $source_g$) will be replaced by $[SetName]$ (e.g. V) or $[FunctionName]$ (e.g. $source$) if no confusion is possible (i.e. only one graph is concerned).

Notation 2. Graph component or element. The expression graph component or element refers undiscernibly to vertex or edges.

Notation 3. Implicit function reference. Let x stand for a graph component, $[FunctionName](x)$ denotes a function applied to x. In case of ambiguity : $[FunctionName]_v(x) = \sigma_{v \in V}[FunctionName](x)$ and $[FunctionName]_e(x) = \sigma_{e \in E}[FunctionName](x)$

Category Theory and Graphs Morphisms

Category theory is a generalized mathematical theory of structures. One of its goals is to reveal the universal properties of structures of a given kind via their relationships with one another [Marq97].

A category describes a set of objects that have an identical mathematical structure, and for which there exists morphisms between those objects and preserving this structure [Fokk92]. The major benefit of working with categories is that any property established for a category can established for any structure of this category.

Graphs are objects of a category of graphs with morphisms as structure preserving mappings between them.

Definition 3. Let g = $(V_g, E_g, target_g, source_g)$ and h = $(V_h, E_h, target_h, source_h)$ be two Graphs; a **graph morphism** from g to h is a pair $m = (m_v, m_e)$ of mappings $m_v : V_g \to V_h, m_e : E_g \to E_h$, such that:

1. $\forall e \in E_g, source_h(m_e) = m_v(source_g(e))$ (source nodes are preserved);
2. $\forall e \in E_g, target_h(m_e) = m_v(target_h(e))$ (target nodes are preserved).

Other properties of interest of graphs morphisms are :

Definition 4. Interesting **graphs morphisms properties:**

1. If m_v and m_e are injective (resp. surjective) m is injective (resp. surjective).
2. If m is injective and surjective (i.e. bijective), m is said to be isomorphic (written $m : G \cong H$ or simply $G \cong H$).
3. If m_v, m_e are total functions, m is said to be a total graph morphism. Otherwise m is said to be a partial graph morphism.

Thanks to morphisms, our initial graph definition (definition 1) will be extended with several features (i.e., identifies, label, type, constraints) while being sure to benefit of all theoretical results provided for the graph category. All features are then consolidated into a single graph definition to form the mathematical basis of our language. Such a way to proceed is found in [Mens99].

Identified Graphs

An identification function is introduced in order to univocally identify each node or edge of a graph. This function is useful as it allows differentiating instances of a same node that would be considered identical without this identifier.

Definition 5. Let L = *(NodeId, EdgeId)* be a pair of disjoint and finite sets of predefined labels. g is said to be a **(I)-graph** iff g is a tuple (*g*, *Id*) such that:

1. g is a graph (see definition1);
2. *Id* is a pair of bijective functions, $Id = (Id_v, Id_e)$ where $Id_v : V \to NodeId$ and $Id_e : E \to EdgeId$.

Definition 6. Let g and h be two (I)-Graphs; Let m be a pair $m = (m_v, m_e)$ of mappings $m_v : V_g \to V_h, m_e : E_g \to E_h$; m is an **identifier preserving (I)-Graph morphism** if:

1. $\forall e \in E_g, source_g(m_e) = m_v(source_g(e))$ (source nodes are preserved);
2. $\forall e \in E_g, target_h(m_e) = m_v(target_g(e))$ (target nodes are preserved);
3. $Id_v(g) = Id_v(g) \circ m_g$ (nodes Id are preserved);
4. $Id_e(g) = Id_e(g) \circ m_g$ (edges Id are preserved).

From definition 5 and 6, it can be said that (I)-Graph is a category with (I)-Graphs as objects and identifier preserving morphisms as morphisms.

Note that Id_v and Id_e are bijective functions. Two nodes or edges cannot share the same identifier and for each identifier is univocally mapped onto an identifier. In mathematical term this can be expressed as follows:

$\forall x, y \in (V \cup E), Id(x) = Id(y) \Rightarrow x = y$ (Id is injective).
$\forall y \in NodeId \cup EdgeId, \exists x \in (V \cup E) \mid Id(x) = y$ (Id is a surjection).

Labeled Graphs

A label attached to each node and edge is introduced in order to label graph components with a name.

Definition 7. Let *L = (NodeLabel, EdgeLabel)* be a pair of disjoint and finite sets of predefined labels. g is said to be a **(L)-graph** iff g is a tuple (*g*, *Label*) such that:

1. *g* is a graph (see definition 1) ;
2. *Label* is a pair of functions, *Label* = $(Label_v, Label_e)$ where $Label_v : V \to NodeLabel$ and $Label_e : E \to EdgeLabel$.

Definition 8. Let g and h be two (L)-Graphs; Let m be a pair $m = (m_v, m_e)$ of mappings $m_v : V_g \to V_h, m_e : E_g \to E_h$; m is an **label preserving (L)-Graph morphism** if:

1. $\forall e \in E_g, source_g(m_e) = m_v(source_g(e))$ (source nodes are preserved);
2. $\forall e \in E_g, target_h(m_e) = m_v(target_g(e))$ (target nodes are preserved);
3. $Label_v(g) = Label_v(g) \circ m_g$ (node labels are preserved);
4. $Label_e(g) = Label_e(g) \circ m_g$ (edge labels are preserved).

From definition 7 and 8, it can be deduced that (L)-Graph is a category with (L)-Graphs as objects and label preserving morphisms as morphisms.

An important discussion on the nature of labeling functions is to be made. Indeed, the property of this function varies following the level of abstraction on which it is defined.

When our graph structure is exploited to describe a meta-model, a labeling functions $Label_v$ and $Label_e$ is totally bijective. This property can be mathematically expressed as follows:

$\forall x, y \in (V \cup E), Label(x) = Label(y) \Rightarrow x = y$ ($Label$ is injective).
$\forall y \in NodeLabel \cup EdgeLabel, \exists x \in (V \cup E) \mid Label(x) = y$ ($Label$ is a surjection).

This means that each graph component is univocally associated with a label and that each label is associated with a graph component. At this level identification and labeling functions are partly redundant.

But our graph language is supposed to describe meta-types as well as their instances (these instances being UI models). In this case the labeling functions $Label_v$ and $Label_e$ are only partial functions. This means that two UI model elements may share a same label.

Another important remark to be made is that the label is not used to specify a graph component type. An additional typing mechanism is introduced for this purpose.

Constrained Graphs

Constraining functions that operate on nodes or edges allow us to attach to any node or edge an arbitrary number of constraints. Constraints can consist in the expression of cardinality constraints, restrictions on the domain or the co-domain of certain functions, etc. It is proposed to express these constraints with first order logic expressions.

Definition 9. Let *C = (NodeConstraint, EdgeConstraint)* be a pair of disjoint and finite sets of node constraints and edge constraints. g is said to be a **(C)-graph** iif g is a tuple (g, Co) such that:

1. g is a graph (see definition 1);
2. Co is a pair of surjective functions, $Co = (Co_v, Co_e)$ where $Co_v : V \to NodeConstraint$ and $Co_e : E \to EdgeConstraint$.

Definition 10. Let g and h be two (C)-Graphs; Let m be a pair $m = (m_v, m_e)$ of mappings $m_v : V_g \to V_h, m_e : E_g \to E_h$; m is an **constraint preserving (C)-Graph morphism** if:

1. $\forall e \in E_g, source_h(m_e) = m_v(source_g(e))$ (source nodes are preserved);
2. $\forall e \in E_g, target_h(m_e) = m_v(target_g(e))$ (target nodes are preserved);
3. $Co_v(g) = Co_v(h) \circ m_g$ (nodes constraints are preserved);
4. $Co_e(g) = Co_e(g) \circ m_g$ (edges constraints are preserved).

From definition 9 and 10, it can be deduced that a (C)-Graph is a category with (C)-graphs as objects and constraint preserving morphisms as morphisms.

Typed Graphs

Typing allows classifying nodes and edges by attaching types to them. Attaching several nodes (or edges) to the same types indicates a commonality in terms of properties between these nodes (or edges).

Definition 11. Let *TY = (NodeType, EdgeType)* be a pair of disjoint and finite sets of predefined types. g is said to be a **(TY)-graph** iff g is a pair (g,Ty) such that :

1. g = is a graph (see definition 1);
2. *Ty* is a pair of total functions attaching a type to each node and edge of the graph. *Type* = (Ty_v, Ty_e) where $Ty_v : V \to NodeType$ and $Ty_e : E \to EdgeType$.

Definition 12. Let g and h be two (TY)-Graphs; Let m be a pair $m = (m_v, m_e)$ of mappings $m_v : V_g \to V_h, m_e : E_g \to E_h$; m is an **type preserving (TY)-Graph morphism** if:

1. $\forall e \in E_g, source_h(m_e) = m_v(source_g(e))$ (source nodes are preserved);
2. $\forall e \in E_g, target_h(m_e) = m_v(target_g(e))$ (target nodes are preserved);
3. $Ty_v(g) = Ty_v(g) \circ m_g$ (nodes types are preserved);
4. $Ty_e(g) = Ty_e(g) \circ m_g$ (edges types are preserved).

From definition 11 and 12, it can be deduced that a (L)-Graph is a category with (L)-graphs as objects and type preserving morphisms as morphisms.

The typing functions introduced here are total. This means that for all graph component there is a corresponding type. A same type may be assigned to several elements. A type may have no graph component of its type. This is mathematically expressed as follows:

$$\forall x \in (V \cup E), \exists y \in NodeType \cup EdgeType \mid Type(x) = y$$

Identified, Labeled, Constrained and Typed graph

All features defined above can be consolidated in a single graph category called (Identified, Labeled, Constrained, Typed)-Graphs (in short: (I,L,C,TY)-Graphs). Note that this consolidation could be modularized that is to say that features presented above can be consolidated "a la carte".

Definition 13. g is an **(Identified,Labelled,Constrained,Typed)-graph** iff:

1. g is a graph (see definition 1)
2. g is an identified graph (see definition 6)
3. g is a labeled graph (see definition 8)
4. g is a constrained graph (see definition 10)
5. g is a typed graph (see definition 12).

Definition 14. Let g and h be two (I,L,C,TY)-Graphs; Let m be a pair $m = (m_v, m_e)$ of mappings $m_v : V_g \to V_h, m_e : E_g \to E_h$; m is an **identifier, label, constraint, and type preserving (I,L,C,TY)-Graph morphism** iff:

1. m is a graph morphism (definition 5)
2. m is an identifier preserving morphism (definition 7)
3. m is a label preserving morphism (definition 9)
4. m is a constraint preserving morphism (definition 11)
5. m is a type preserving morphism (definition 13).

From definition 13 and 14, it can be deduced that (I,L,C,TY)-Graph is a category with (I,L,C,TY)-graph as objects and (I,L,C,TY)-Graph morphism as morphisms.

This consolidation has the advantage of being modular. This means that features presented above can be consolidated in an "a la carte" way to form other categories.

An Improved Typing Function

We want to have a better control on the typing mechanism. Graph types are introduced for this purpose ([Mont70, Corr97, Heck02]). Graph types contain all "type information" that is used to type the model level.

Types that are returned by the functions Ty_v and Ty_e (see definition 11) belong to two type sets ($NodeType$, $EdgeType$). These sets contain possible types.
The main idea with graph types is to replace type sets by graphs. In order to support this, the typing mechanism of definition 11 has to be slightly adapted.

Definition 15. Let $Type = (NodeType, EdgeType)$ be a pair of disjoint and finite sets of types. Let TG be a fixed (L,C)-graph (TG is called a *type graph*). g is said to be a **(I,L,C,TY) TG-Typed graph** iff g is a pair (g,Ty^{TG}) where:

1. g is a (I,L,C,TY,N)-graph (see definition 16).
2. $Ty^{TG} : g \rightarrow TG$ such that $type$ is a total (L,C)-graph morphism.

The above definition asserts that there must be a correspondence between, on the one hand, node and edge type at the model level and, on the second hand, node and edge labels at the meta-level. Furthermore, constraints defined on labels in TG are applicable to types in g. This situation is expressed in Fig. 3-17.

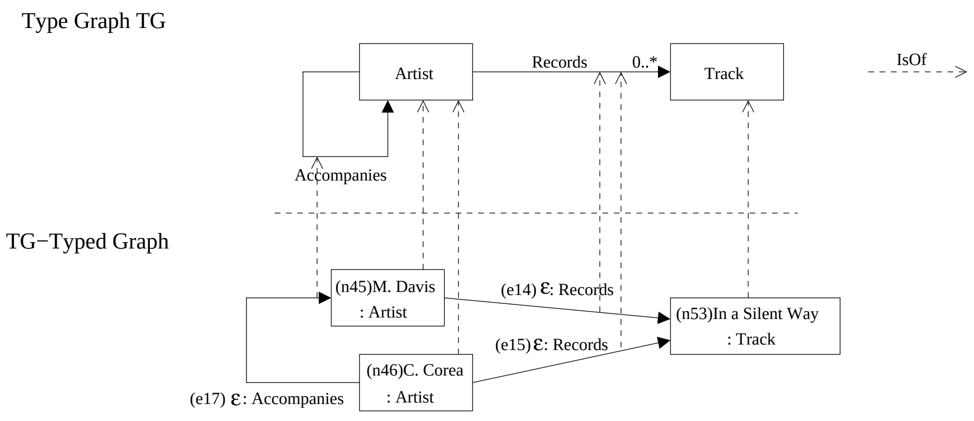


Figure C-1. Typed Graph and its Graph Type

In addition the following graph morphisms can be defined:

Definition 16. Let g and h be two (I,L,C,TY) TG-Typed Graphs; $m : g \to h$ is a (I,L,C,TY) **TG-Type preserving graph morphism** iff:

1. f is a (I,L,C,TY) graph morphism (see definition 17)
2. $\forall x \in dom(m) : type(h) \circ f = type(g)$.

From definition 15 and 16 it can be said that (L,C)-Graph is a category with (L,C)-graphs as objects and nesting preserving morphisms as morphisms.

From definition 15 and 16, it can be asserted that constraints defined in a type graph TG can also constrain the corresponding TG-Typed graph. For instance, a cardinality constraint on an edge between two types in a TG graph is effective on the TG-Typed graph. This could be expressed mathematically as follows:

$$\forall v \in V, ifTy(v) = "tutorial" \Rightarrow \exists E' \subseteq E \wedge$$
$$E' = \{e \in E \mid Label(e) = "isGivenBy" \wedge source(e) = v\} \wedge (1 \leq |E'| \leq 3)$$

In the above expression we define a cardinality constraint between a node representing an entity labeled "tutorial" and another entity labeled "speaker". The expressed constraint says that a tutorial cannot be given by more than three speakers.

Other constraints can limit the domain or the co-domain of source and target functions in order to avoid or force the use of certain type of edges with certain type of nodes. For instance an edge with the label "Is Husband Of" can only occur between two nodes with label "man" and "woman" (not the case anymore in Belgium). This example is mathematically expressed as follows:

$$\forall e \in E, label(e) = "isHusbandOf", \exists v_1, v_2 \in V \mid source(e) = v_1 \wedge$$
$$target(e) = v_2 \Rightarrow Label(v_1) = "man" \wedge Label(v_e) = "woman"$$

The reader may have noticed that examples of constraints have been defined on labels and not on types. Indeed, these examples are expressed at the concept level. They will be enforced at the model level. As labels at the concept level are types at the model level it is normal to express constraints on labels at the meta-level. In the second example, the "translation" of the constraint at the model level gives:

$$\forall e \in E, Type(e) = "isHusbandOf", \exists v_1, v_2 \in V \mid source(e) \wedge$$
$$target(e) = v_2 \Rightarrow Type(v_1) = "man" \wedge Type(v_2) = "woman"$$

In order, to simplify the expression of type graphs, types can be structured into partial orders. Organizing nodes and edges of the type graph into a partial order (see definition 18) presents the advantage of propagating constraints i.e., constraints applicable to one type can be directly inherited by all subtypes of this type.

Definition 17. A (L,C)-type graph TG is said to be **($\leq_v, \leq_e$)-ordered graph** if (NodeLabel, $\leq_v$) and (EdgeLabel, $\leq_e$) are partial order.

Definition 18. The set of $NodeLabel$ (see definition 8) is a **partial order** if $\exists \leq_v \in E$, such that:

1. Reflexivity: $\forall nodelabel \in NodeLabel \Rightarrow nodelabel \leq_v nodelabel$
2. Antisymmetry:
   $\forall nodelabel_i, nodelabel_j \in NodeLabel, if nodelabel_i \leq_v nodelabel_j$
   $\wedge nodelabel_j \leq_v nodelabel_i \Rightarrow nodelabel_i = nodelabel_j$
3. Transitivity:
   $\forall nodelabel_i, nodelabel_j, nodelabel_k \in NodeLabel, if\ nodelabel_i \leq_v$
   $nodelabel_j \wedge nodelabel_j \leq_v nodelabel_k \Rightarrow nodelabel_i \leq_v nodelabel_k$

Definition 19. The set of $EdgeLabel$ (see definition 8) is a **partial order** if $\exists \leq_e \in E$, such that:

1. Reflexivity: $\forall edgelabel \in EdgeLabel \Rightarrow edgelabel \leq_v n$
2. Antisymmetry:
   $\forall edgelabel_i, edgelabel_j \in EdgeLabel, if edgelabel_i \leq_v edgelabel_j \wedge$
   $edgelabel_j \leq_v edgelabel_i \Rightarrow edgelabel_i = edgelabel_j$
3. Transitivity:
   $\forall edgelabel_i, edgelabel_j, edgelabel_k \in EdgeLabel, if edgelabel_i \leq_v$
   $edgelabel_j \wedge edgelabel_j \leq_v edgelabel_k \Rightarrow edgelabel_i \leq_v edgelabel_k$

As said above the definition of type graphs can be exploited to propagate constraints among types. Such a propagation mechanism is expressed in definition 20.

Definition 20. If T is partial-ordered type graph then the following **inheritance mechanisms** must be defined:

1. $\forall v, w \in V_t : if vlabel(v) \leq_v vlabel(w)$ then
   $vconstraints(w) \subseteq vconstraints(v)$ (constraints of supertypes are inherited from subtypes)
2. $\forall e \in E_t : \forall s_1, s_2, t_1, t_2 \in V_t : if\ source(e) = s_1 \wedge target(e) = t_1 \wedge$
   $vlabel(s_2) \leq vlabel(s_1) \wedge vlabel(t_2) \leq_v vlabel(t_1) \Rightarrow$
   $\exists f \in E_t \mid source(f) = s_2 \wedge target(f) = t_2 \wedge$
   $elabel(f) = elabel(e) \wedge econstraints(f) = econstraints(e)$
   (edge constraints between supertypes are inherited by edges among subtypes).

Annex C. Transformational Rules

1. Basic nodes capabilities

Node creation

Figure C-1 represents the emptiness of the left hand side providing that no condition is necessary to create the node described in the right hand side.

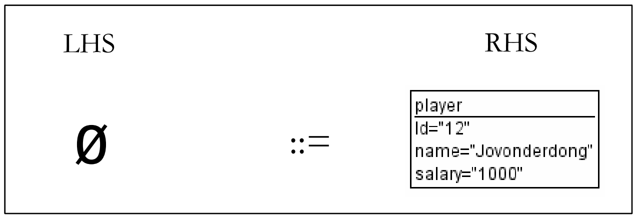


Figure C-1. Creation of a node with attributes

Node modification (identified instance)

Figure C-2 shows a rule selecting a specific node on the base of its id attribute and assigns to this node a specific attribute value.

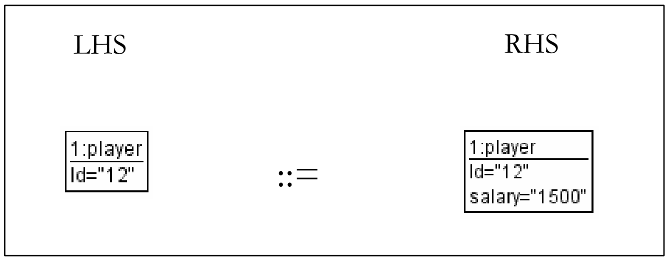


Figure C-2 Node modification (identified instance)

Node modification (unidentified instance)

Figure C-3 shows a rule that could be expressed as follows: "for all players that played the match on the 04/06/04, align their salary to 2000".

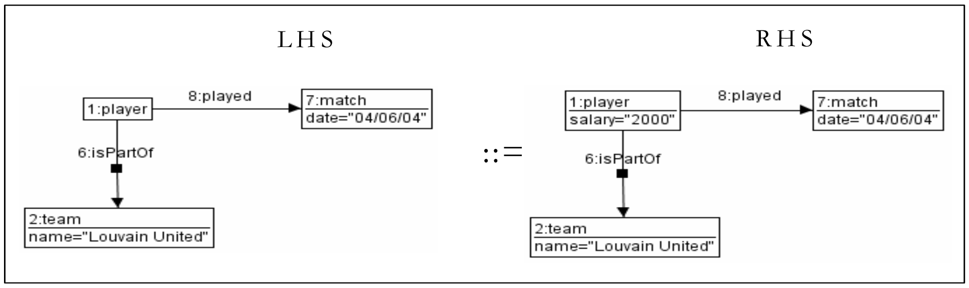


Figure C-3 Node modification unidentified instance

Negative application condition (1)

A negative application condition could be added to the preceding rule (Figure C-4). This negative application condition transforms the meaning of the rule into: "for all players that played the match on the 04/06/04, raise their salary to 2000 unless they played the match of the 10/10/03" (this last match was a very bad one!).

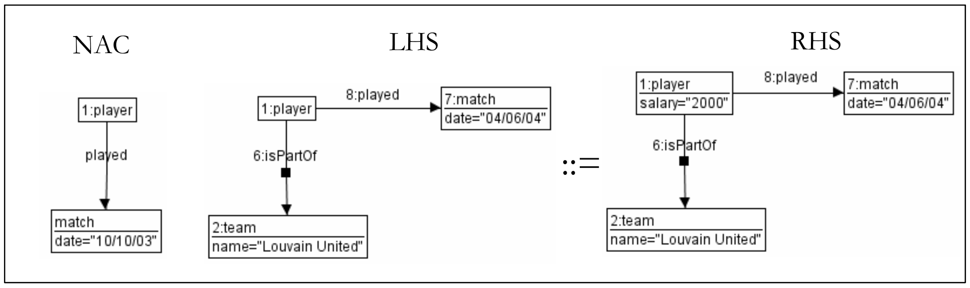


Figure C-4 Negative application condition

Negative application condition for iterative execution of rules (2)

Rules that detect patterns on a graph structure and make appropriate modifications depending on the presence of this pattern, is possible for the system that search iteratively for the left hand side. Consequently, there is a risk that the pattern matching algorithm will match several times on the same instances leading to an infinite looping of the execution of the rule. For this purpose a special negative application condition has to be introduced. "NAC2" in Figure C-5, is such an example. It says that the rule should not be applied if the salary of the player equals already "2000".

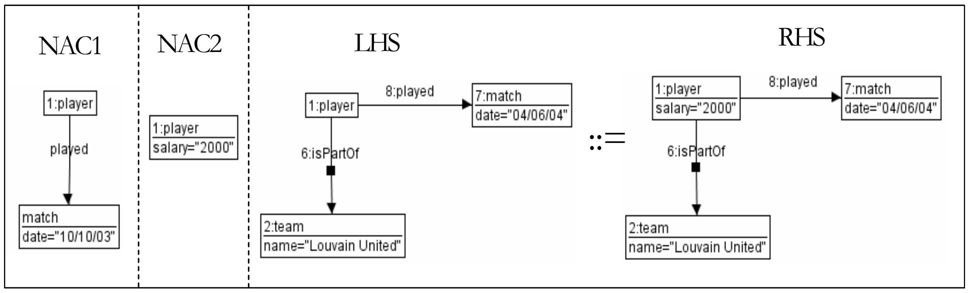


Figure C-5 Negative Application Condition (2)

Rule with variable and variable condition as positive application condition

Figure C-6 could be expressed as follows: "raise by 500 the salary of all players that played the match of the 04/06/04 only if their salary was inferior to 3000". This rule illustrates two different mechanisms. A first one consists in the use of a variable in the left hand side, this variable is incremented by a constant in the right hand side ("x:=x+500"). A second one consists in the use of a positive application condition that compares the value of a variable with a constant (note that x could have been compared with another variable).

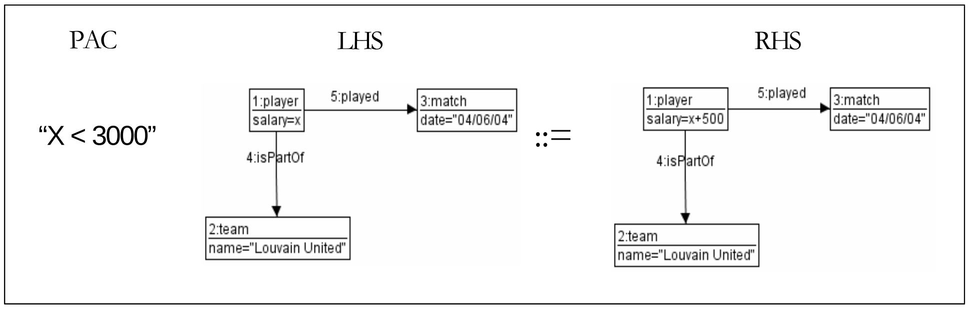


Figure C-6 Rule with variable and positive application condition

Transfer of an attribute value

Figure C-7 illustrates a very altruistic rule, which may be expressed as follows: "If two players of a same team are friends and one earns more than the other, then align their salaries". Here the value of a variable is transferred from one node (the richest player) to another one (the poor friend).

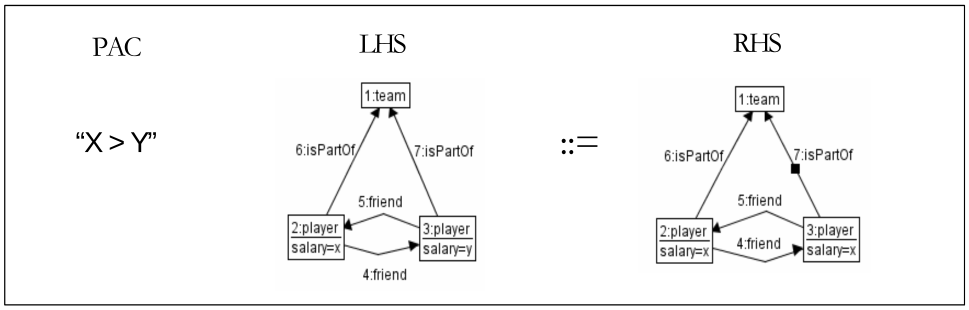


Figure C-7 Transfer of an attribute value

Edge creation

Figure C-8 illustrates a rule that could be expressed as follows "All players of Louvain United with a salary greater than 3000 should be assigned to the match of the 04/06/04" (It will be a tough match !)

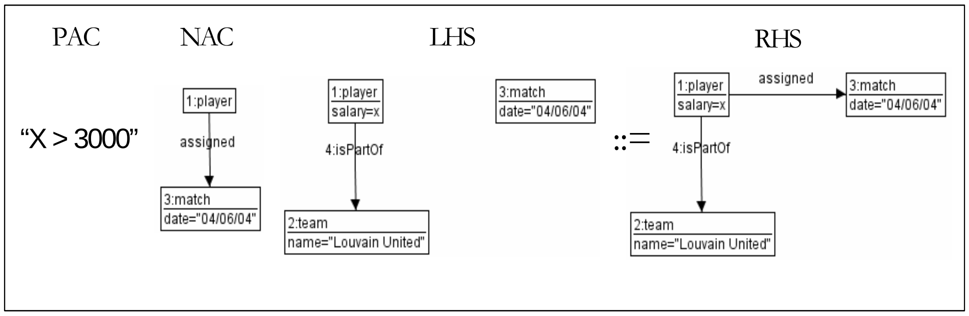


Figure C-8 Edge creation

Node deletion

Figure C-9 shows the most delicate operations of all: node deletion. Indeed, the problem with node deletion is that they raise the question of dangling edges. We adopt a very clear policy regarding this problem: all edges pointing to or originating from a deleted node should be erased. In other words, no dangling edges are allowed.

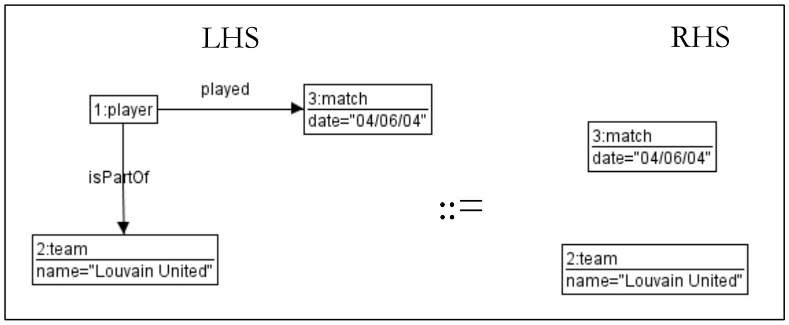
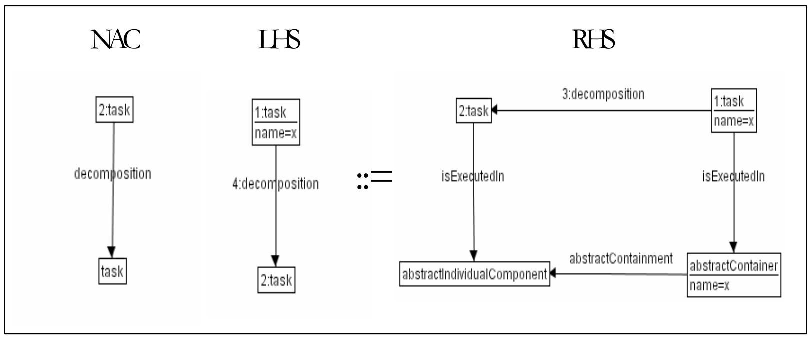


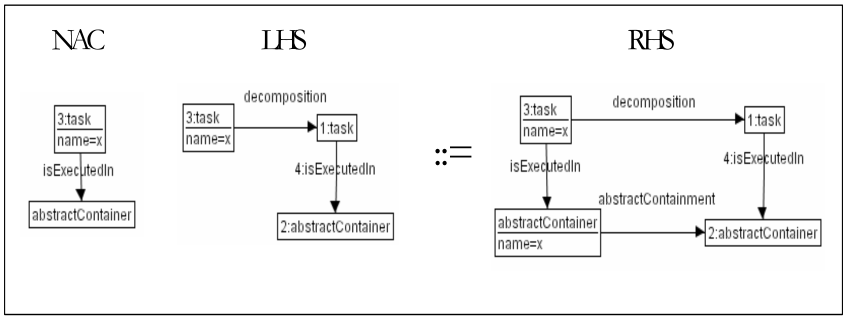
Figure C-9 Node deletion

Rule 1: For each leaf task of a task tree, **create an Abstract Individual Component**. For each task, parent of a leaf task, create an Abstract Container. Link the abstract container and the Abstract Individual Element by a containment relationship.



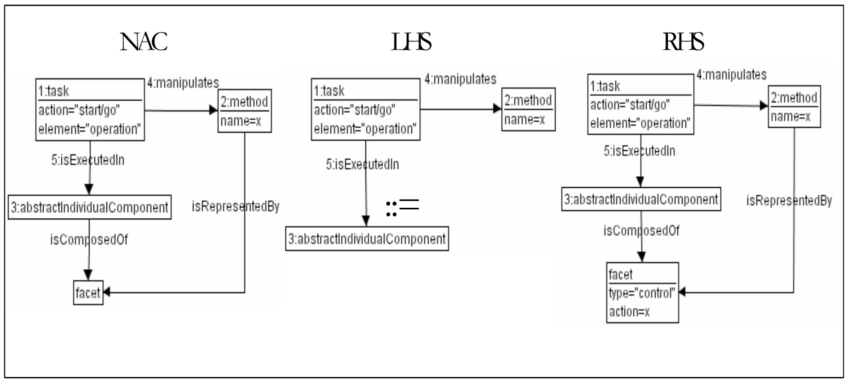
Rule 1 Creation of abstract individual components derived from task model leaves

Rule 2: **Create an Abstract Container** structure parallel to the task decomposition structure.



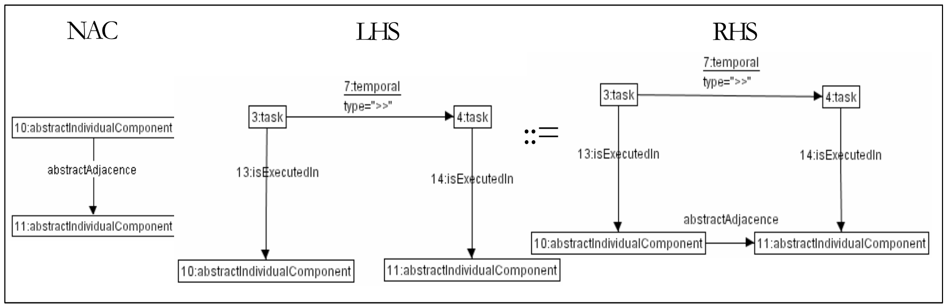
Rule 2 Creation of abstract containers derived from task model structure

Rule 3: for each **abstract individual element mapped onto a task** such that the tasks nature consists of the activation of a method and this task is mapped onto a class, assign to the abstract individual component an action facet that activates the mapped method.



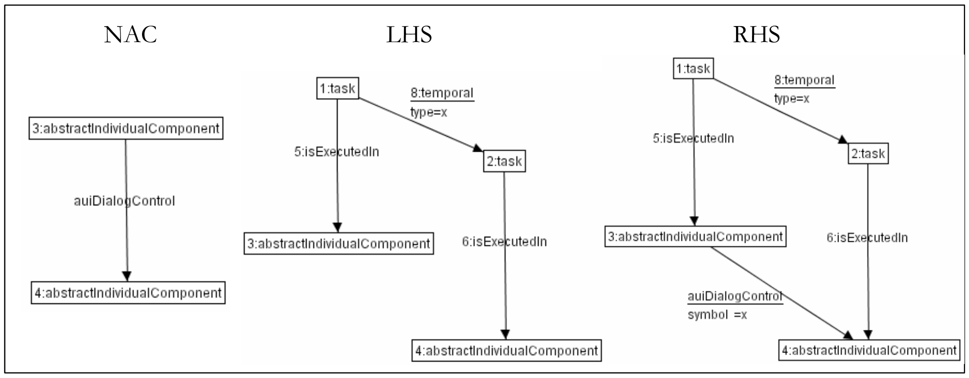
Rule 3 Creation of a facet for an abstract individual component derived from task action type

Rule 4: for **every couple of AIC mapped onto sister tasks that are sequential** “>>”, create a relationship of type “abstractAdjacence” between these AIOs.


NAC
10:abstractIndividualComponent
abstractAdjacence
11:abstractIndividualComponent
LHS
7:temporal
type=">>"
3:task
4:task
13:isExecutedIn
14:isExecutedIn
10:abstractIndividualComponent
11:abstractIndividualComponent
::=
RHS
7:temporal
type=">>"
3:task
4:task
13:isExecutedIn
14:isExecutedIn
abstractAdjacence
10:abstractIndividualComponent
11:abstractIndividualComponent

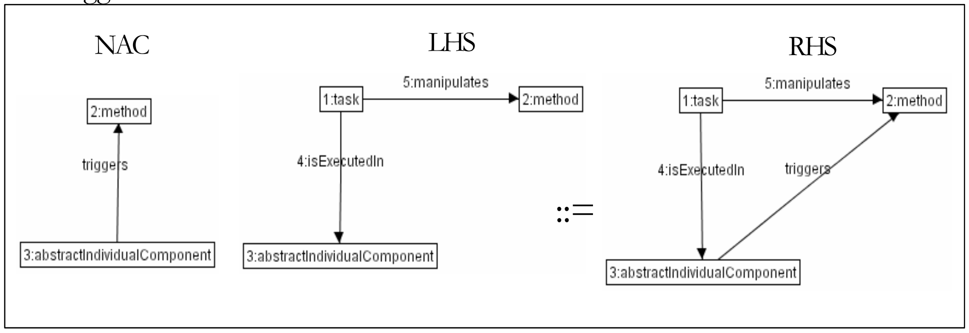
Rule 4 A sequentialisation of abstract individual component derived from task temporal relationships

Rule 5: for each couple of sister **tasks mapped onto AICs**, define a dialog control relationship between these AIC that has the same semantic as the temporal relationship.


NAC
LHS
RHS
3:abstractIndividualComponent
auiDialogControl
4:abstractIndividualComponent
1:task
8:temporal
type=x
5:isExecutedIn
2:task
3:abstractIndividualComponent
6:isExecutedIn
4:abstractIndividualComponent
1:task
8:temporal
type=x
5:isExecutedIn
2:task
3:abstractIndividualComponent
6:isExecutedIn
auiDialogControl
symbol =x
4:abstractIndividualComponent

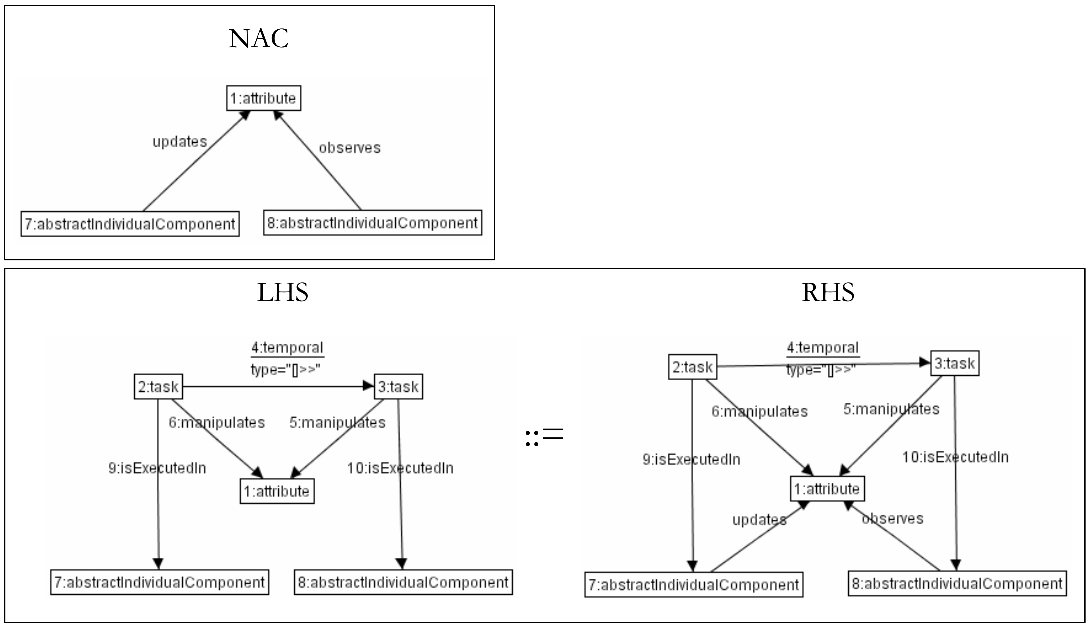
Rule 5 Abstract Dialog Derivation from Task Model

Rule 6: for each **task that manipulates a method,** the AIC that represents this task triggers the method.



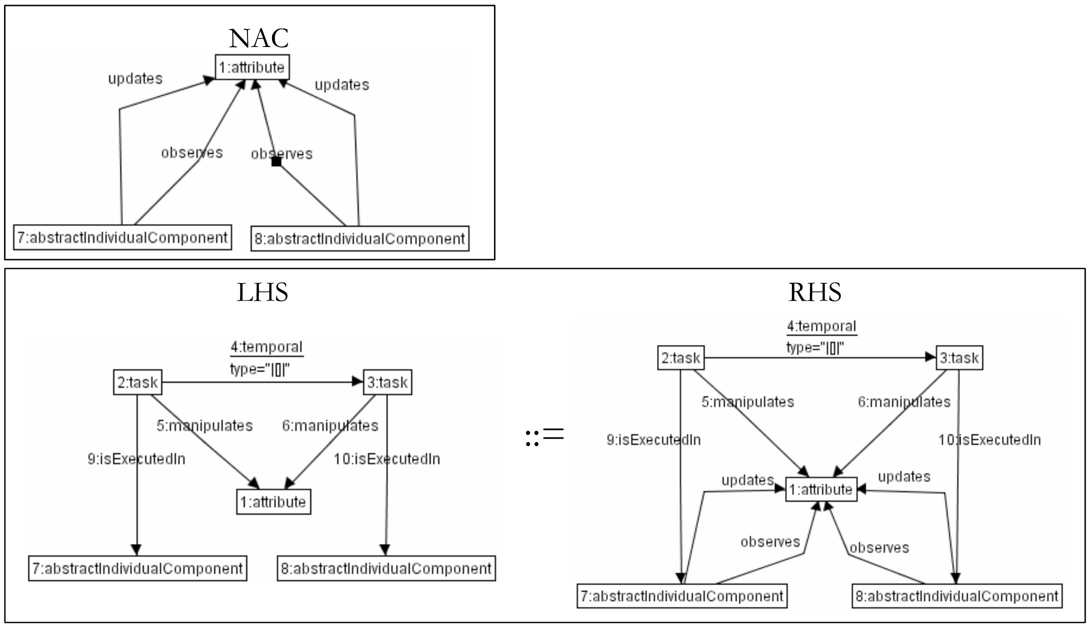
Rule 6 Deriving triggering relationships from task domain mappings

Rule 7: if two sister tasks manipulate a same attribute and are temporally constrained with a "sequence with information passing" relationship, each of these tasks being mapped onto an AIC, then the AIC that is mapped with the first task updates the attribute manipulated by the tasks. The second AIC observes this attribute.



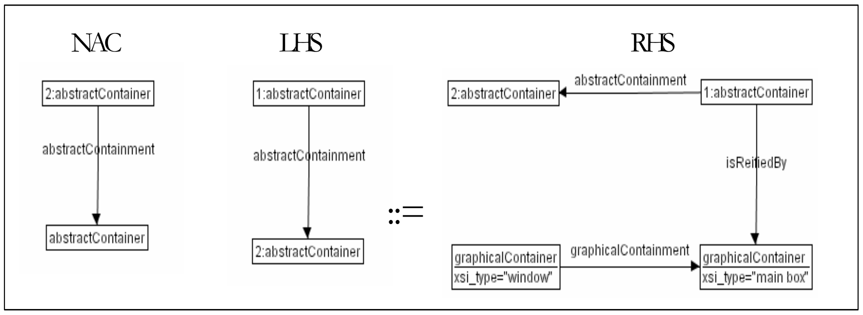
Rule 7 Derivation of Updates and observes structure on the base of a task relationship of sequential information passing

Rule 8: if two sister tasks manipulate a same attribute and are temporally constrained with a “concurrent information passing” relationship, and each of these tasks is mapped onto an AIC, then both AIC observe and update the attribute that is manipulated by the tasks.



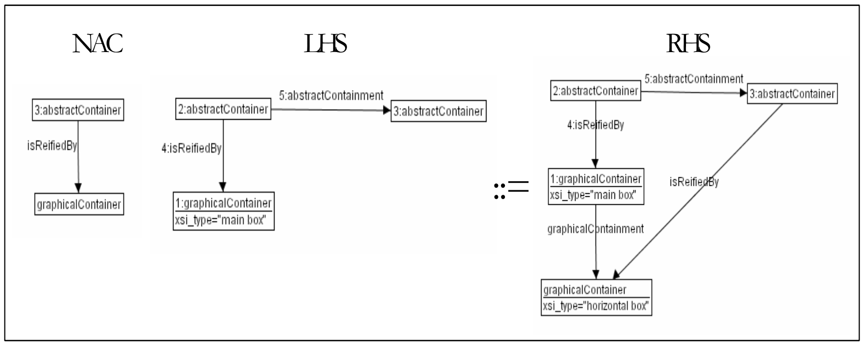
Rule 8 Derivation of Updates and Observes structure on the basis of a task relationship of concurrent information passing

Rule 9: Each **abstract container at level “leaf-l” is transformed into a window**. Note that an abstract container is always reified into a, so called, box at the concrete level. This box is then embedded into a window.



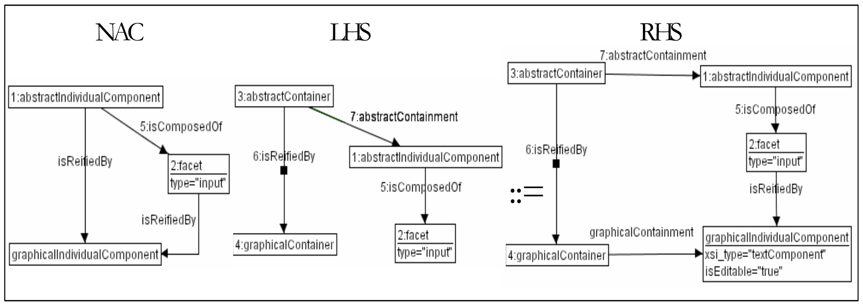
Rule 9 A creation of windows derived from containment relationships at the abstract level

Rule 10**:** each abstract container contained into an abstract container that was reified into a window is transformed into an horizontal box and embedded into the window.



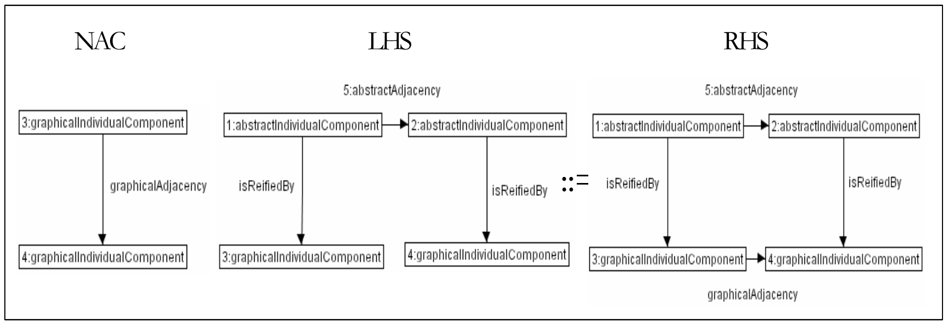
Rule 10 A generation of window structure derived from containment relationship at the abstract level

Rule 11: each input facet of an abstract individual component is reified by a graphical individual component (a type of concrete individual component) of type "editable text component" (i.e., a text box).



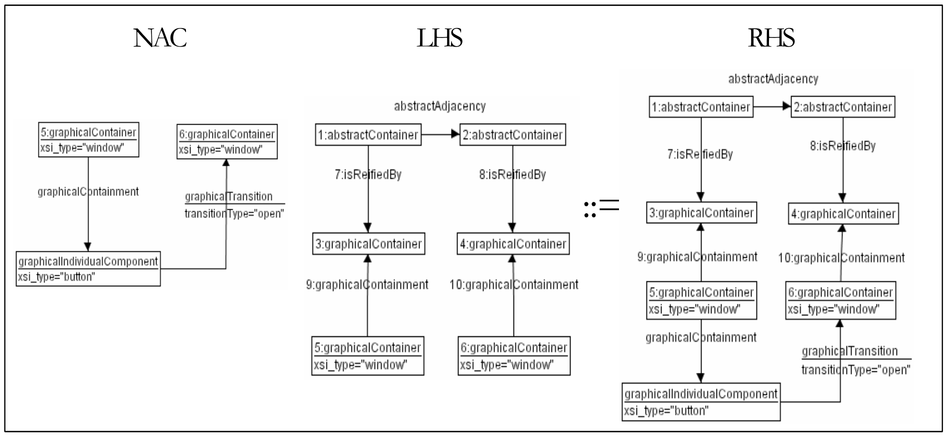
Rule 11 Creation of an editable text component (i.e., an input field) derived from facets type of abstract components

Rule 12: for each couple of abstract individual components related by an "abstractAjacency" relationship and reified into concrete individual components, generate a "concreteAdjacency" relationship between the concrete individual components.


NAC
LHS
RHS
3:graphicalIndividualComponent
graphicalAdjacency
4:graphicalIndividualComponent
5:abstractAdjacency
1:abstractIndividualComponent
2:abstractIndividualComponent
isReifiedBy
3:graphicalIndividualComponent
4:graphicalIndividualComponent
::=
graphicalAdjacency

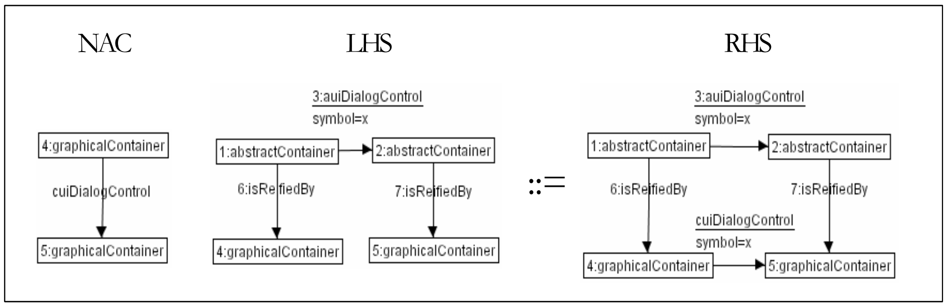
Rule 12 A placement of graphical individual components derived from spatio-temporal relationships at the abstract level

Rule 13: for each container related to another container belonging to different windows, and their respective abstract container being related by a "is before relationship", generate a navigation button in source container pointing to the window of target container.



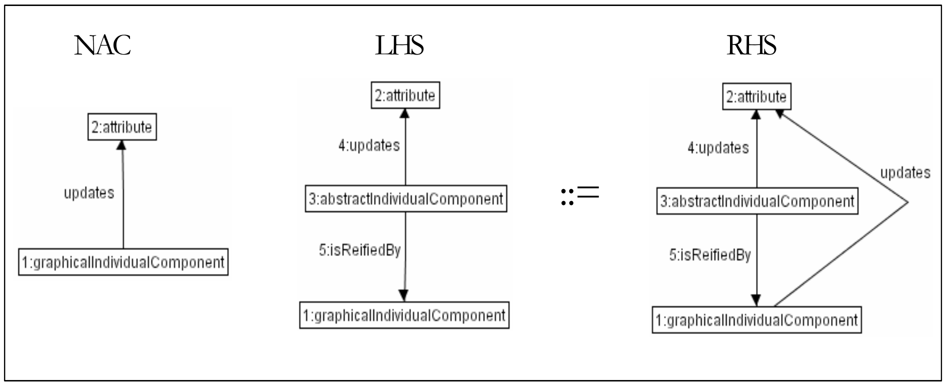
Rule 13 A window navigation definition derived from container adjacency relationships

Rule 14: for each couple of abstract container with a dialog control relationship, transpose this relationship to the couple of concrete containers that reify them.



Rule 14 Derivation of the concrete dialog from abstract dialog

Rule 15: for each AIC updating a domain concept, if a CIC reifies this AIC then the CIC updates this same domain concept.


NAC
LHS
RHS
2:attribute
updates
1:graphicalIndividualComponent
2:attribute
4:updates
3:abstractIndividualComponent
5:isReifiedBy
1:graphicalIndividualComponent
::=
2:attribute
4:updates
updates
3:abstractIndividualComponent
5:isReifiedBy
1:graphicalIndividualComponent

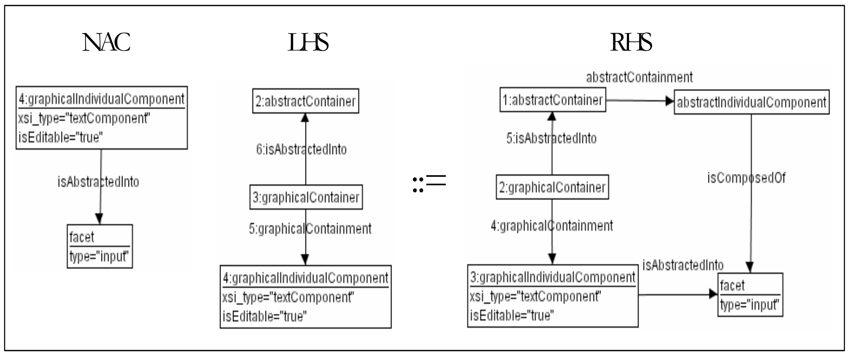
Rule 15 Transposition of update relationship

2. From Concrete User Interface to Code Rules

Step T3 consists in code generation from a CUI. Code generation techniques for UIs is a very well known topic. [Czar00] presents a state-of-the art of model to code techniques (e.g., visitor-based approach and template based approach). Scientific results for this transformation have been shown in systems issued from research like: Janus [Balze95], Trident [Boda95], Modi-D [Puer97] or from commercial world e.g., Genova [Geno04] or Oliva Nova [Moli02]. The present work does not particularly contribute to this area although several tools have been developed to provide code generation support from the concrete user interface level.

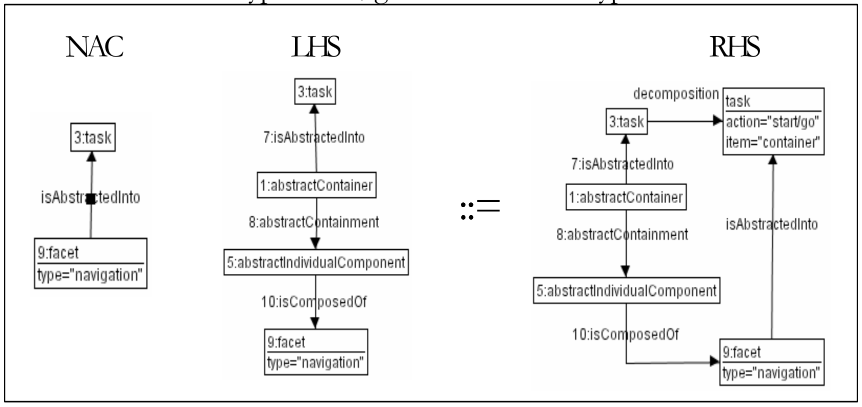
3. Reverse Engineering Rules

Rule 16**:** for each editable graphical individual component, create an abstract individual component equipped with an input facet.



Rule 16 Creation of a facet at the abstract level derived from a type analysis of graphical individual components

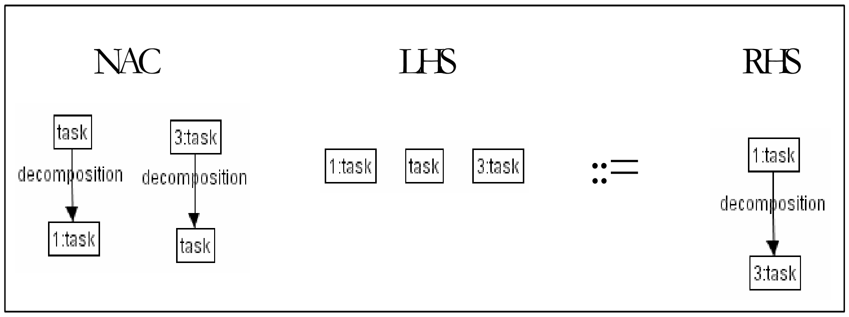
Rule 17: for each abstract individual component equipped with a navigation facet create a task with action type “start/go” on an item of type “element”.



Rule 17 Definition of task action types derived from an analysis of facets at the abstract level

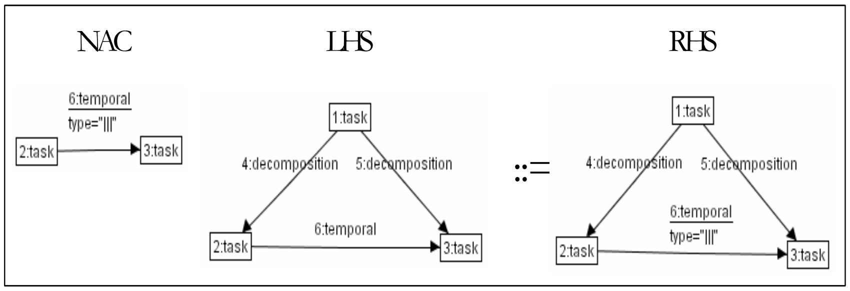
4. Adaptation to context change

Rule 18: (1) erases each intermediary task (i.e., non-leaf and non-root tasks). (2) attaches every leaf task to the root.



Rule 18 Flattening of a task tree structure

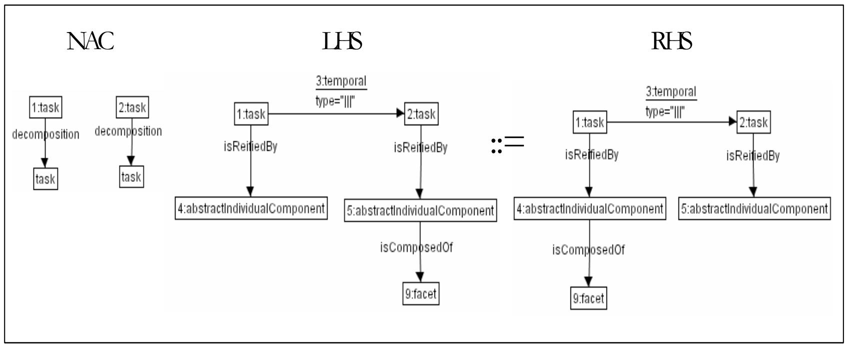
Rule 19**:** for each sister tasks change their temporal relationship into concurrent.



Rule 19 Transforming all temporal relationship to concurrent

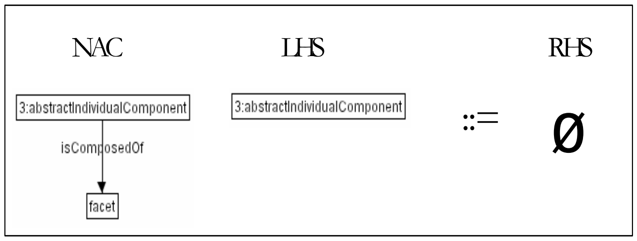
Step: From Abstract User Interface to Abstract User Interface

Rule 20: for each pair of abstract individual component mapped onto concurrent tasks, transfer all facets of the abstract individual component that is mapped onto the task target of the concurrency relationship, to the other abstract individual component.



Rule 20 A merging of facets of abstract individual components

Rule 21**:** erase all abstract individual components that have no facets left.



Rule 21 Erasing abstract individual components with no facets left

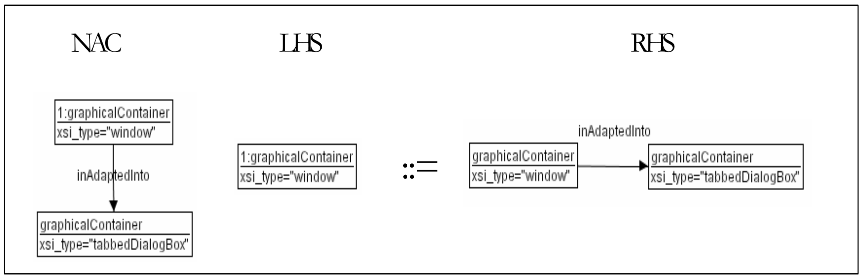
Step: From Concrete User Interface to Concrete User Interface

Adaptation at the concrete level is illustrated by several development sub-steps: container type modification (called concrete container re-formation), modification of the types of concrete individual components (called concrete individual components re-selection), layout modification (layout re-shuffling), or navigation re-definition. Examples for these first three adaptation types are given hereafter.

5. Sub-step: Concrete container re-formation

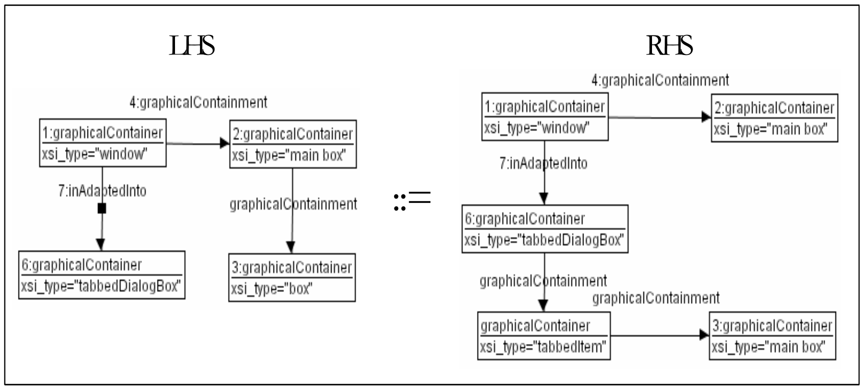
Concrete container Re-Formation may cover situations like container type transformation (e.g., a window is transformed into a tabbed dialog box), container system modification (e.g., a system of windows is merged into a single window).

Rule 22: each window is selected and mapped onto a newly created tabbed dialog box.



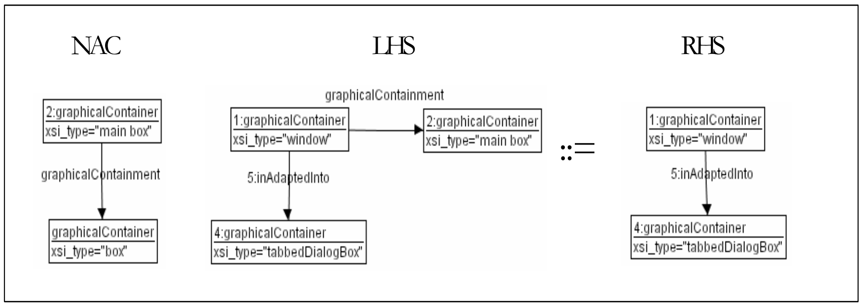
Rule 22 Initializing of the adaptation process by creating graphical component to adapt into

Rule 23: transfers every first level box of the window to adapt it into a tabbed item composing a tabbed dialog box.



Rule 23 Creation of a tabbed item and transfer of the content of the adapted window

Rule 254**:** cleans up the specification of remaining empty main boxes.

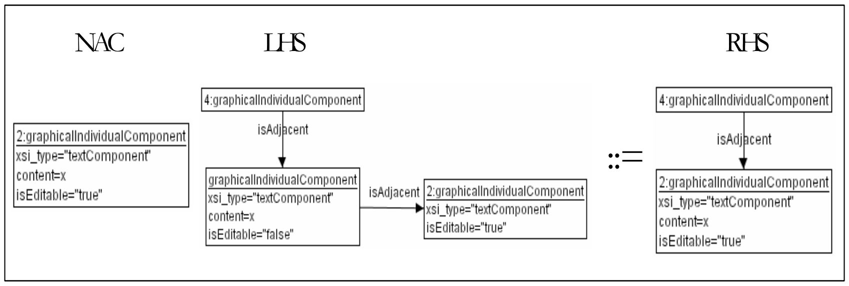


Rule 24 Deletion of unnecessary containers

6. Sub-step: Concrete individual component re-selection

Re-selection transformations adapt individual component into other individual components.

Rule 25: for each couple of adjacent editable text component and non-editable text component. Erase the editable text component and transfer its content into the non-editable text component (unless a content has already been transferred).

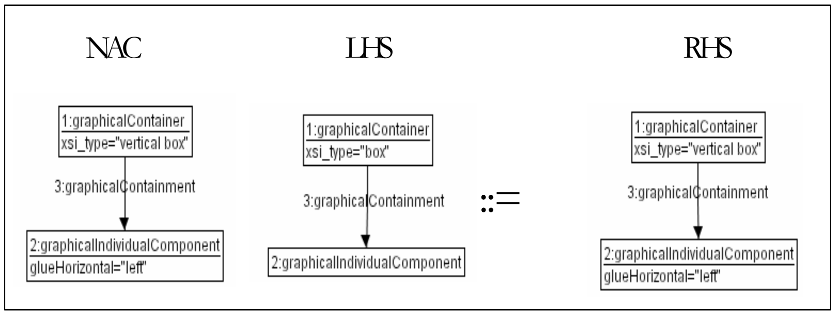


Rule 25 Merging of a non-editable text component (e.g., a label) and an editable text component (e.g., an input field) into one single editable text component

7. Sub-step: Layout re-shuffling

A layout at the concrete level is specified with horizontal and vertical boxes. An elements contained into a box may be glued to an edge of this box.

Rule 26: each box is transformed into a vertical box and every individual component is glued to left.



Rule 26 Squeezing of a layout structure to display vertically

Annex D. UsiXML Ontology for User Interface Specification

The first description of Limbourg [Limb04c] refers to the separation of concerns, inspired by Dijkstra [Dijk76]:

"*This principle states that the different aspects of a problem should be isolated from one to each other. Separation of concerns allows studying fractions of a matter in an independent manner while modularizing this matter. A concern gathers properties relevant to one perspective that can be maintained on an artifact*"

In [Limb04b] they introduce UsiXML language to handle the concepts defined in their ontology of UIs. This language is structured according to four basic levels of abstractions defined by the Cameleon reference framework [Calv03] (see Figure D-1).

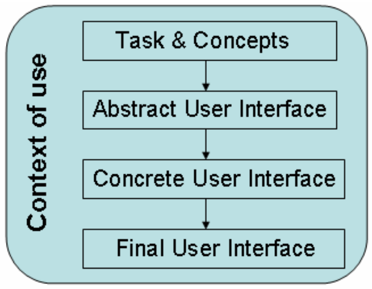


Figure D-1 The Cameleon reference framework for multi-target UIs.

The description of four levels of their approach is described as follows:

- At the top level is the ***Task & Concepts*** level that describes the various interactive tasks to be carried out by the end user and the domain objects that are manipulated by these tasks. These objects are considered as instances of classes representing the concepts.
- An ***Abstract UI*** (AUI) provides a UI definition that is independent of any modality of interaction (e.g., graphical interaction, vocal interaction, etc.).
- As an AUI does not refer to any particular modality, we do not know yet how this abstract description will be concretized: graphical, vocal or multimodal. This is achieved in the next level.
- The ***Concrete UI*** (CUI) concretizes an AUI for a given context of use into *Concrete Interaction Objects* (CIOs) so as to define layout and/or interface navigation of 2D graphical widgets and/or vocal widgets. Any CUI is composed of CIOs, which realize an abstraction of widgets sets found in popular graphical and vocal toolkits. A CIO is defined as an entity that users can perceive and/or manipulate (e.g., push button, text field, check box, vocal output, vocal input, vocal menu). The CUI abstracts a Final UI in a definition that is independent of programming toolkit peculiarities.
- The ***Final UI*** (FUI) is the operational UI, i.e. any UI running on a particular computing platform either by interpretation (e.g. through a Web browser) or by execution (e.g., after the compilation of code in an interactive development environment).

As depicted in Figure D-1, the ***Context of use*** surrounds the different levels. This context of use describes all the entities that may influence how the user's task is carrying out with the future UI. It takes into account three relevant aspects, each aspect having its own associated attributes contained in a separate model: *user type* (e.g., experience with device and/or system, task motivation), *computing platform type* (e.g., desktop, PocketPC, PDA, GSM), and *physical environment type* (e.g., lighting level, stress level, noise level). These attributes initiate transformations that are applicable depending on the current context of use.

Finally, in order to map different elements belonging to the models described above, UsiXML provides the designer with a set of pre-defined relationships called *mappings*.

1. Task Model

A task model describes the various tasks to be carried out by a user in interaction with an interactive system. The task model used in this methodology is similar as the proposed in [USIX06], which is an extended version of ConcurTaskTree (CTT) [Pate97], selected as it represents user's tasks along with their logical and temporal ordering.

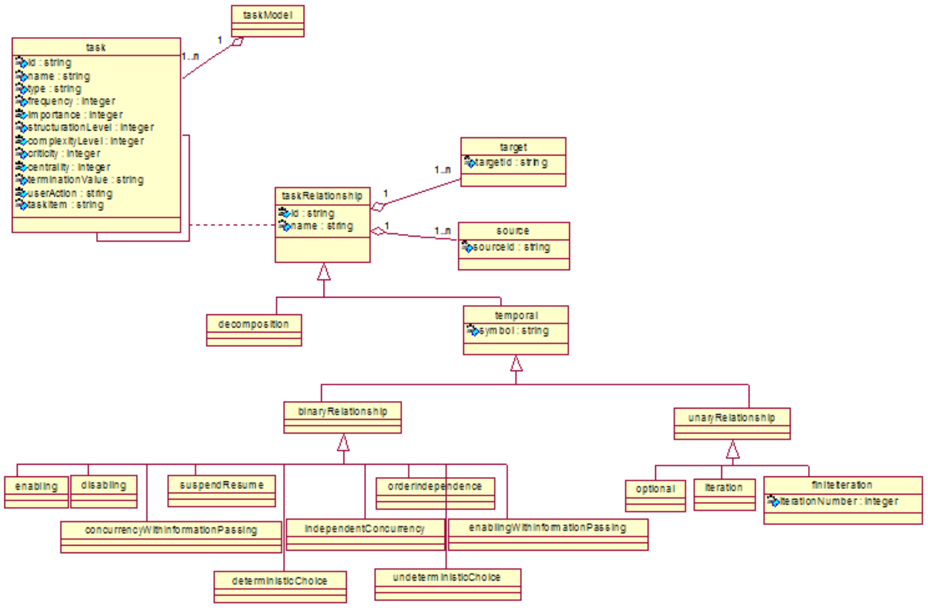


Figure D-2 Conceptual view on the task model

The proposed task model is composed of *tasks* and *task relationships*, see Figure D-2. A task *frequency* attribute is an assessment of the relative frequency of execution of a task. Task frequency is evaluated on a scale from 1 to 5. A task *importance* attribute assesses the relative importance of a task with respect to main user's goals. Task importance is evaluated on a scale from 1 to 5. A value of 1 means that a task has a low importance, 5 means that a task is very important. Tasks have been described with a *name*, and a *type*. Task *type,* similarly as propose by [Pate97], may be:

- *User tasks* are notably useful to predict a task execution time, as is the user responsible on executing them. A user task refers to a cognitive action like taking a decision, or acquiring information.
- An *interactive task* involves an active interaction of the user with the system (e.g., selecting a value, browsing a collection of items).
- A *system task* is an action that is performed by the system (e.g., check a credit card number, display a banner).
- An *abstract task* is an intermediary construct allowing a grouping of tasks of different types.

Task relationships are:

- Enabling (T1 has to be finished in order to initiate T2)
- Non-deterministic choice (Once one task is finished the other cannot be accomplished anymore)
- Deterministic Choice (Once one task is initiated, the other cannot be accomplished anymore)
- Parallelism (T1 is interleaved with T2).
- Sequential independence (Is equivalent to (T1>>T2) OR (T2 >>T1))
- Deactivation (T2 may interrupt T1 before the termination of T1;. T1 cannot be resumed after T2 has terminated.)
- Suspend/Resume (T2 may interrupt T1 before the termination of T1. Once T2 is finished, T1 may be resumed.)
- Enabling with information passing. Task T1 has to be finished in order to initiate task T2 and T2 is synchronized with T1 on some piece of data.
- Parallelism with information passing. Task T1 is interleaved with task T2 while they synchronize on some data.
- Task Iteration (*|n). Task T can be iterated an infinite number of times *, or n times
- Optional tasks. Task T is optional.

Several additional constraints may be formulated on the consistency of a task model:

- There exists a maximum of one binary (i.e., temporal or decomposition) relationship between two tasks.
- If a task is decomposed into another task then this last task must have a brother task.

- There is only one root task. This means that there is only one element with no decomposition relationship pointing to it.

2. Domain Model

A *domain model* describes the real-world concepts, and their interactions as understood by users and the operations that are possible on these concepts [DSou99]. We selected UML class diagrams as the basis of expression for our domain model. We considered UML class diagrams as Extended Entity Relationship model (EER) [Teor86]. The main reason for this choice is that UML has become a lingua franca in the domain of software engineering and is widely used in industrial practice [Limb04c].

We rely on [USIX06] domain meta-model, shown in Figure D-3. This UML class diagram shows the features added to the initial UML standard in order to better tackle the problem of transformational development of UIs. For instance, the domain n of values attached to attributes is described with a richer precision in order to allow widget selection (e.g., enumerated domains can be described extensively).

From [USIX06] Domain model concepts are:

- *domainClass*. Classes describe the characteristics (attributes and methods.) of a set of objects sharing a set of common properties.
- *Attribute*. Attributes enable a description of a particular feature of a class.
- The *type* of an attribute refers to common data types found in most programming language i.e., Boolean, char, string, integer, float. The type attribute may also make reference to an object type, such as the vector required to denote a 3D color.
- The *cardinality* of an attribute indicates the number of values an attribute may be associated with. The cardinality can be specified by providing two integers: a minimal cardinality and a maximal cardinality.

An original typology allows characterizing a type of *domain* for an attribute. Indeed, attributeDomainCharacterization takes the value of: interval, continuous interval, discrete interval, linear interval, circular interval, set[n] (where n is the number of possible values in an attribute domain). When used in combination with a task model, this typology helps to map domain attributes to a type of interaction object by which it will be rendered [Limb04c]. For instance, a "choose element" task on an attribute with a circular interval enables the derivation of a (multi-state) toggle button.

Methods (in this context) are presences which are called either by objects of the domain or by user interface components. Methods manipulate object's attributes. Methods are, here, described with their signature i.e., with their name, type, and parameters.

Objects are instances of a class. An object is composed of attribute instances which may have values and define the state of an object.

domain class relationships describe various types of relationships between classes. They can be classified in three types: *generalization*, *aggregation*, *usage, materialization, instanciation* and *ad hoc*. Class relationships are described with several attributes enabling the specification of role names and cardinalities.

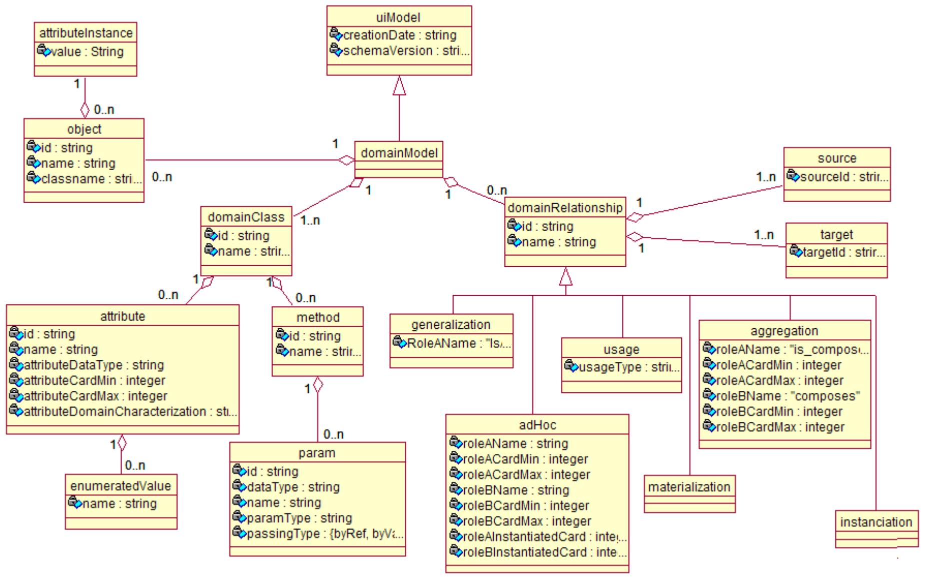


Figure D-3 Conceptual model for a domain model

3. Abstract User Interface Model

From [Limb04c] *Abstract User Interface (AUI) model* is defined as "*a user interface model that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is as independent as possible from modalities and computing platform specificities"*.

We rely to a similar AUI (Figure 4-17). The AUI is populated by *Abstract Interaction Objects (*AIO*)* and *abstract user interface relationships.* These concepts constitute a vocabulary that is independent of the modality and the computing resources for which a system is targeted at.

A *modality* (also called interaction technique) can be defined more precisely, after [Niga95], as the coupling of a physical device d with an interaction language *L: <d, L>*. Our language supports, at the concrete level, two modalities: speech (i.e. *audito*ry) input and output and graphic (i.e., *graphical*) input and output. This support will be extended at the graphical level that considered 2D UI.

Abstract Interaction Object (AIO) may be of two types *Abstract Individual Components (*AIC*)* and *Abstract Containers (*AC*).*

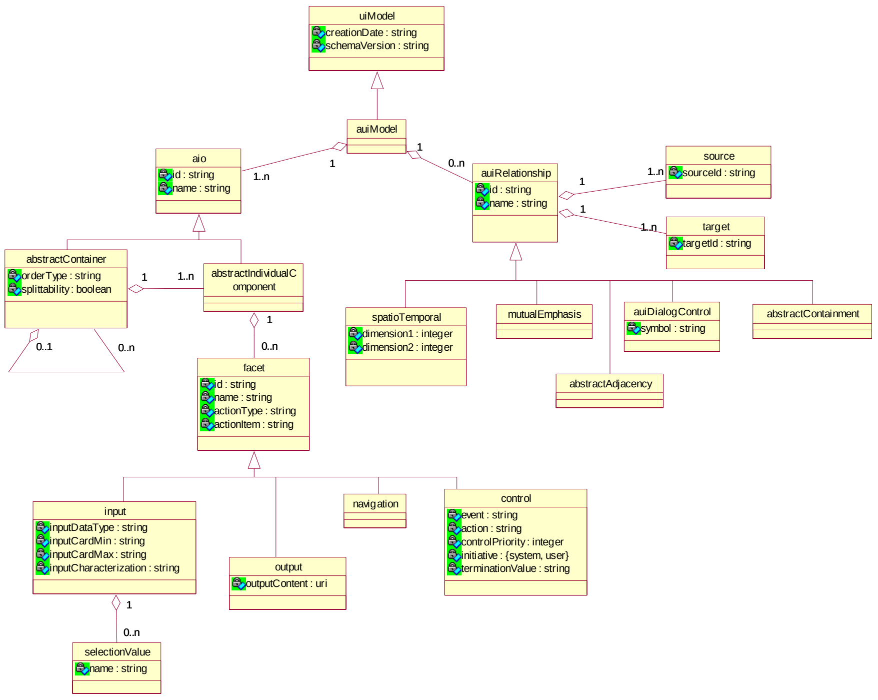


Figure D-4 Concept model for abstract user interfaces

An *Abstract Individual Component* (*AIC*) is an abstraction that allows the description of interaction objects in a way that is independent of the modality in which it will be rendered in the physical world. An AIC may be composed of multiple facets. Each facet describes a particular function an AIC may endorse in the physical world. Four main facets are identified:

An *input* facet describes the input action supported by an AIC.
An *output* facet describes what data may be presented to the user by an AIC.
A *navigation* facet describes the possible container transition a particular AIC may enable.
A *control* facet describes the links between an AIC and system functions i.e., methods from the domain model when existing.

A single AIC may assume several facets at the same time. The AIO that reifies this multi-facetted AIO will assume all those ‘functionalities’. For instance, a CIO may display an output while accepting an input from a user, ensure a transition between windows and trigger a method defined in the domain model.

An *Abstract Container* (AC) is an entity allowing a logical grouping of other abstract containers or abstract individual components. AC are said to support the execution of a set of logically/semantically connected tasks. Actually AC may be reified, at the concrete level, into one or more graphical containers like windows, dialog boxes, layout boxes or time slots in the case of auditory user interfaces. However there is no concretization of these objects for 3D UIs.

Abstract User Interface Relationships (AUI relationship) are relationships that can be drawn between abstract interaction objects of all kinds.

Five types of abstract relationships may be defined at this level:

Decomposition relationship allows specifying a hierarchical structure of abstract containers and abstract individual components.
AbstractAdjacency relationship indicates that two AIO are logically adjacent.
Spatio-temporal relationship allows a specification of a very precise layout in time or space in a way that is independent of any modality.
Dialog control relationship allows a specification of a flow of control between the abstract interaction objects.
Mutual emphasis relationship allows specifying that two components should be somehow differentiated at the concrete level. This relationship may be useful in a user interface where the probability of confusing two UI elements is high (e.g., in an airplane cockpit, a field displaying the angular speed and the absolute speed).

4. Concrete User Interface Model

The Concrete User Interface Model (CUI) represents a concretization of an AUI model. A CUI is populated by Concrete Interaction Objects and Concrete User Interface relationships between them, figure 4-18. The CUI model is a UI model allowing a specification of an appearance and behavior of a UI with elements that can be perceived by users. The actual [USIX06] specification just considers 2D UIs (Figure 4-19 and 4-20) and vocal UIs (Figure 4-21).

By definition, a CUI is modality dependent as any CUI instance refers to the interaction modalities that have been selected for this UI. In contrast to its modality dependence, a CUI remains toolkit independent as no CUI instance does refer to any physical element (i.e., toolkit elements or widget) of the computing platform. Nonetheless, a CUI description can be detailed enough to allow a complete rendering of a user interface [Limb04c].

A CUI model (Figure 4-18) is composed of *Concrete Interaction Objects* (CIO) and *cui relationships*.

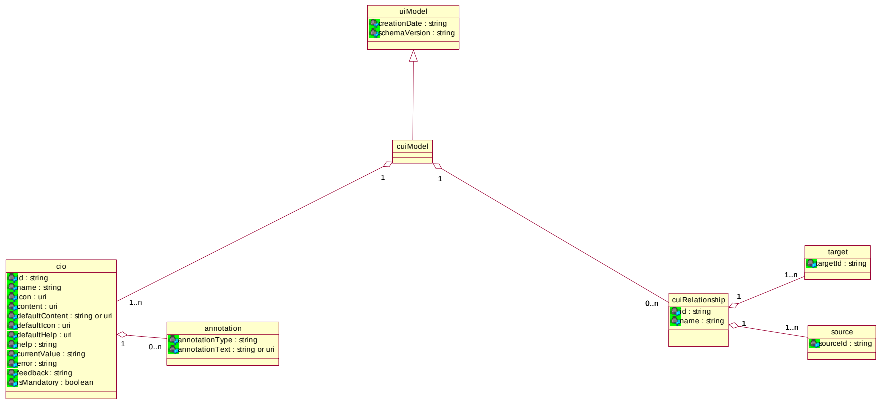


Figure D-5 Root elements of the concrete user interface model

A Concrete Interaction Object (CIO) is defined as an entity that users can perceive and/or manipulate (e.g., a push button, a list box, a check box, a sound). The actual specification realizes an abstraction of widget sets found in popular toolkits: 2D graphical (Java Swing, HTML 4.01, Flash) and auditory (earcons and VoiceXML 2.0). In other words, CIOs allows an expression of UI elements that is independent of their actual rendering. Our target is to extend this representation to 3D UIs, including languages such VRML, X3D or JAVA 3D.

Graphical and auditory CIOs are further decomposed into containers and individual components. We have just summarized the main characteristics of the actual model more information can be found in the [USIX06] documentation. More emphasis will be dedicated on the extension.

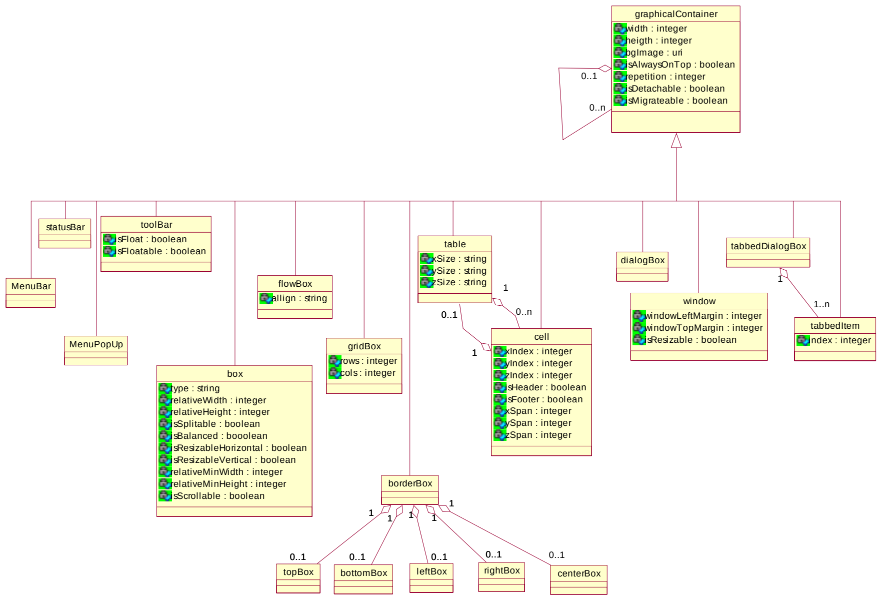


Figure D-6 2D Graphical containers

Graphical containers (GC) are detailed in Fig. 4-20 and corresponds to classical and common 2D UIs containers. Attributes used are as abstract as possible in order to respect the independence on implementation.

Graphical Individual Components *(GIC)* are detailed in Figure 4-20. Text components are differentiated in two types, for input (an input field, a password field, a multi-line input field) and output (a label, a complex textual output as a rtf file) purposes.

Vocal Concrete interaction objects are represented in Figure 4-21. *Vocal Containers* represent a logical grouping of other auditory containers or auditory individual components. *Vocal individual components* are of five types: auditory output which may consist in music, voice or a simple "earcon" (i.e., an auditory icon), auditory input which is a mere time slot allowing the user to provide an auditory input using her voice, or any other physical device able to produce sound, vocal navigation (Specifies a transition to another vocalForm), break (Interrupts the execution of the current vocalContainer) and exit (Terminates the execution of the vocal interface).

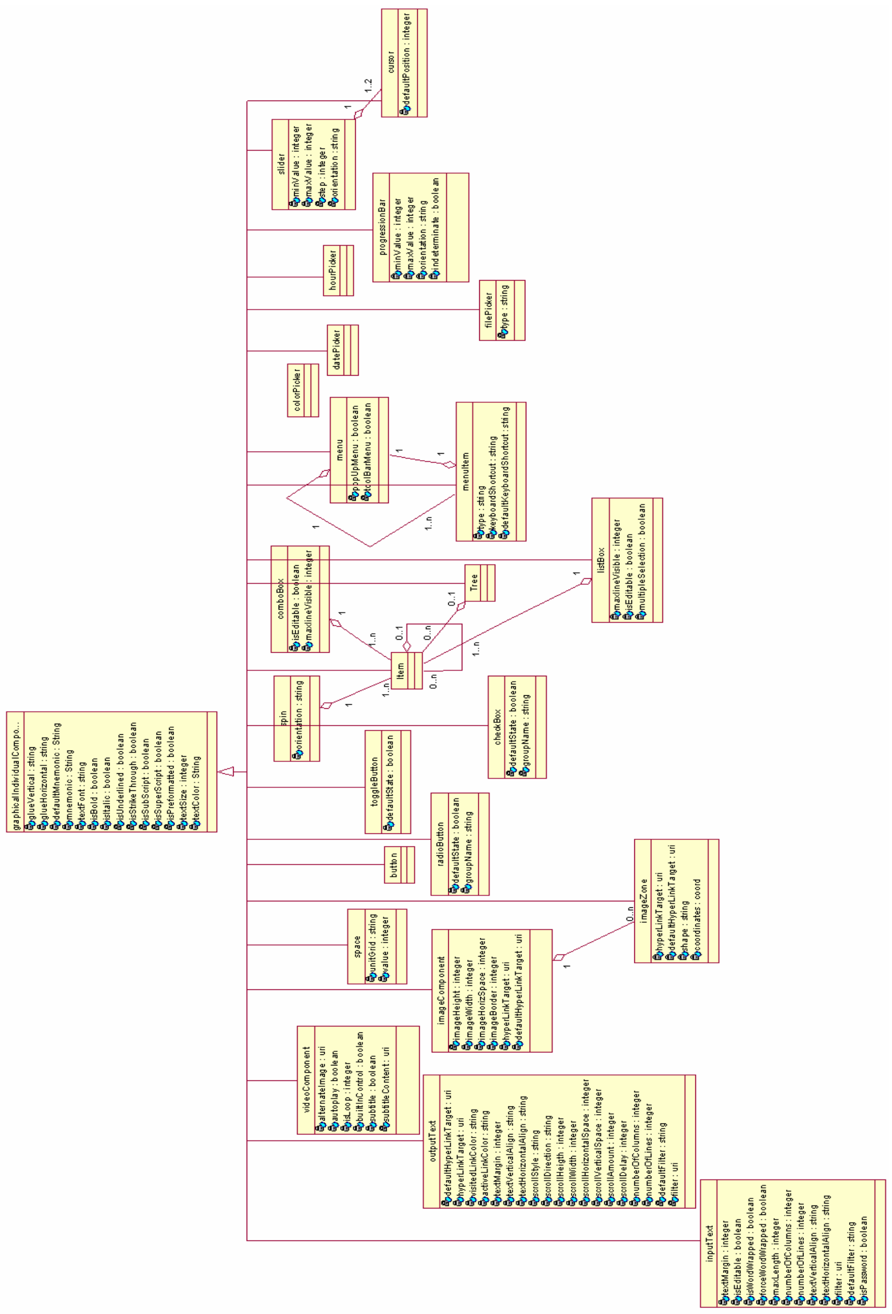


Figure D-7 Graphical Individual Components Types

CUI relationships are exposed in Fig. 4-23. Similarly to Concrete Interaction Objects they are divided into *vocal relationships* and *graphical relationships*. *Dialog control relationship* can be defined between both types of interaction objects [USIX06].

Vocal relationships are of three types: *vocal transition* that enables to specify a transition between two auditory containers; *vocalAdjacency* that indicates a time adjacency between two auditory components; and *vocalContainment* that allows adding or deleting *vocalIndividualComponets* from a *vocalContainer*.

Graphical relationships are of five types: *Graphical transition* specifies navigation links between the different containers populating the UI, *alignment* that may also be specified among any individual component belonging to the same window, *adjacency* indicates that two components are topologically adjacent, *emphasis* enables to specify that two or more graphicalIndividualComponents must be differentiated in some way (e.g., with different color attributes) and *containment* analog to the vocal containment, allows to specify that a graphicalContainer embeds one or more graphicalContainers or one or more graphicalIndividualComponents.

Dialog control allows a specification of a flow of control between the concrete interaction objects. As so a dialog control may be specified independently of a task model. LOTOS (see Appendix A) operators are used for this purpose. For instance a relationship CIC1.EnterCountry []> CIC2.EnterProvince, indicates that CIC2 cannot be initiated while CIC1 is not terminated and that CIC1 has provided a value for the data on which the two component synchronize with.

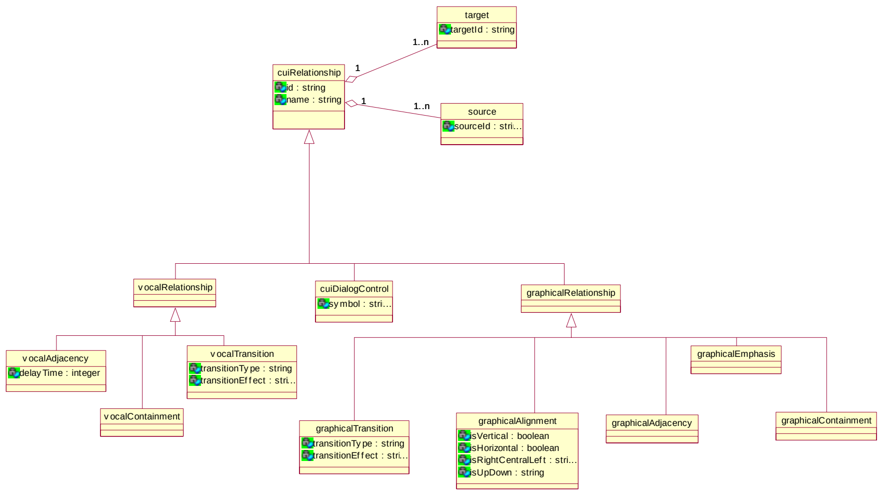


Figure D-8 Relationships ate the concrete user interface level

Any CIO may be associated with any number of *behaviors* (see Fig. 4-24). A *behavior* is the description of an event-response mechanism that results in a system state change. The specification of a behavior may be decomposed into three types of elements: an *event,* a *condition,* and an *action* [USIX06].

An *event* is a description of a run-time occurrence that triggers an action. They consist of any system event (i.e., issued from a process belonging to the domain), user interface event (i.e., issued in the context of the user interface). A limitation on the events is that they cannot make any reference to coordinates, which is imperative in 3D event handling. Events can be composed into more complex event expressions using a subset of the LOTOS operators introduced earlier. However, as it is not part of the language, the behavior description is straightforward from the actual [USIX06] specification.

A *condition* is the expression of a state that has to hold true before (pre-condition) or after (post-condition) an action is performed. A condition may be positive or negative. An *action* is a process that results in a state change in the system. An action can be of three types: a *method call*, a *transformation system*, or a *transition.*

A *method call* is a call to a method that is external to the UI. If a domain model exists, all method calls must reference a method belonging to this model. A method call is normally specified with the name of the method (under the form

Class.methodName), but other referencing techniques are not forbidden. The method call parameters can be specified by making a reference to the value of a property of an object belonging to the CUI.

A *transformation system* is the expression of any property change at the UI level. We use a mechanism to specify property changes on the UI. This mechanism is similar to the one that will be introduced in Chapter 4. To avoid too much forward reference, it can be said that a transformation system can be explained as follows: when a pattern is found in CUI specification, changes should occur on the elements matching the pattern. A transformation system might be, for instance, "when a green button is found in the specification, change the color property of this button to red" or "For all text components belonging to the main window, increase their font by a factor of 2".

A *transition*, also called *navigation,* is a description of a change in the container's visibility property of a user interface system. A transition has a source (a navigation individual component) and a target (generally a container). Depending on the type of modality, transitions may be of different types (see above in this Section).

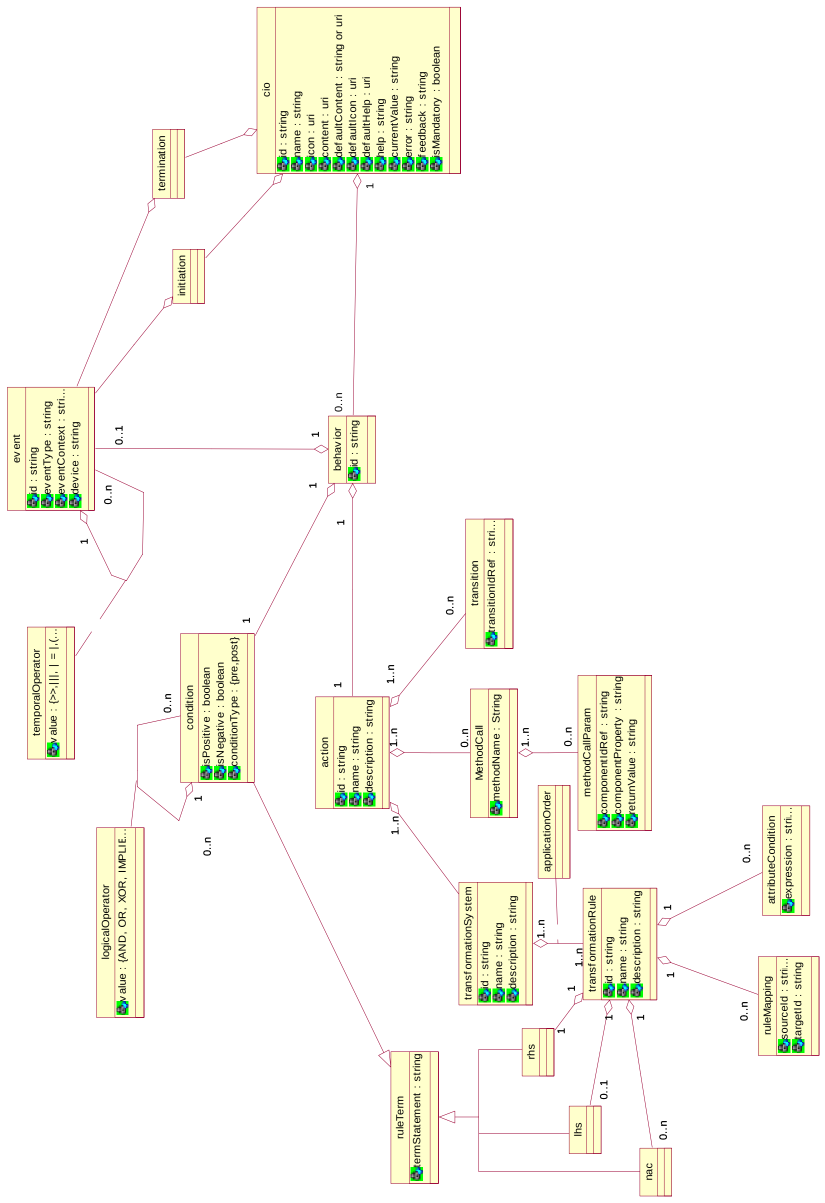


Figure D-9 Behavioral specification at the concrete level

5. Context Model

A context model (Figure 4-36) is a model describing the three aspects of a context of use in which an end user is carrying out an interactive task with a specific computing platform in a given surrounding environment [Thev01]. In [USIX06] the context model is composed of *context* and the *plasticity model set*. The *context* is defined as a triple of the form <*e, p, u*> where *e* is an element of the environments set considered for the interactive system, *p* is an element of the platforms set considered for the interactive system and *u* is an element of the users set for the interactive system. The *plasticity model set* composed of plasticity domains, which defines a sub-area in a specified context (itself included in the physical space: user, environment, platform) where a specified AIO/CIO will be represented such as a specified form. Details of the model can be found in [USIX06] documentation.

The environmental model, which is part of the context model, relies on the assumption that the user interact in the physical world. The actual describes any property of interest of the "physical" environment where the user is using the UI on the computing platform to accomplish her interactive tasks. Such attributes may be physical (e.g., lighting conditions), psychological (e.g., level of stress), and organizational (e.g., location and role definition in the organization chart).

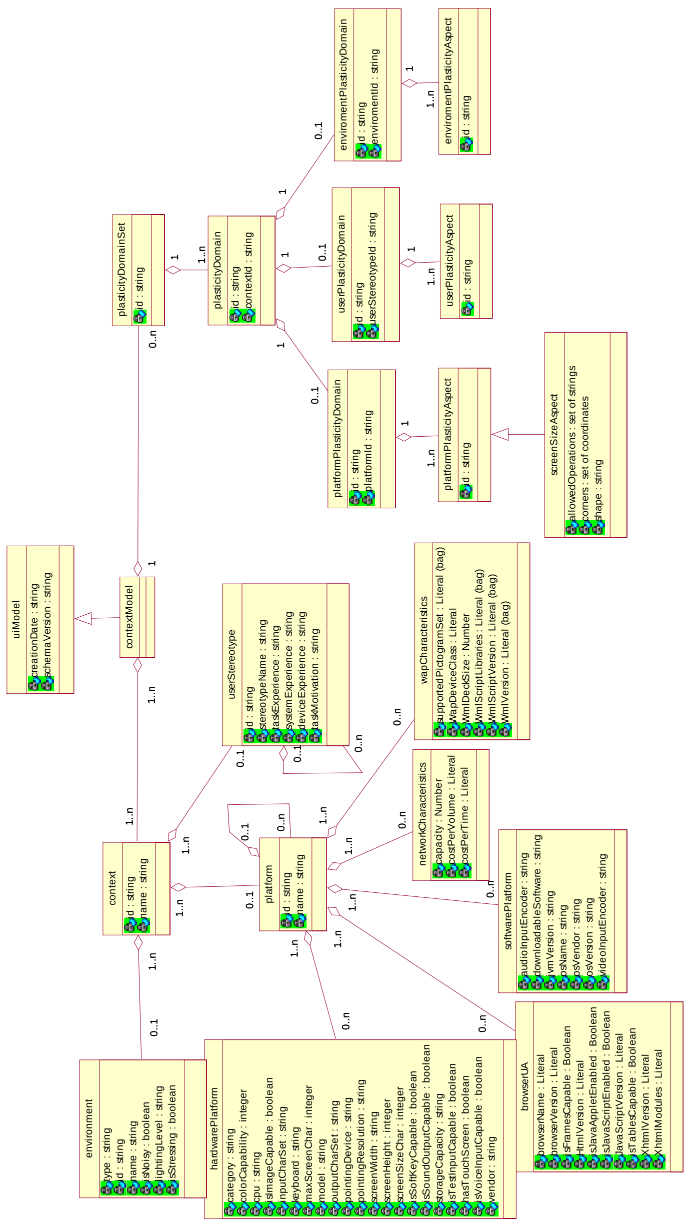


Figure D-10 Context model

6. Inter-Model Relationships

The concepts described in the ontology require a way to be interconnected. Model integration is a well-known issue in transformation driven development of UI [Puer99]. This problem was sorted with the creation of a set of pre-defined relationships allowing a mapping of elements from heterogeneous models and viewpoints (Figure 4-39). Several advantages where identify in [Limb04c] such as: the derivation of the system architecture (mappings between domain and CUI/AUI models), for traceability in the development cycle (reification, abstraction and translation), for addressing context sensitive issues (has context), for dialog control issues, for improving the preciseness of model derivation heuristics.

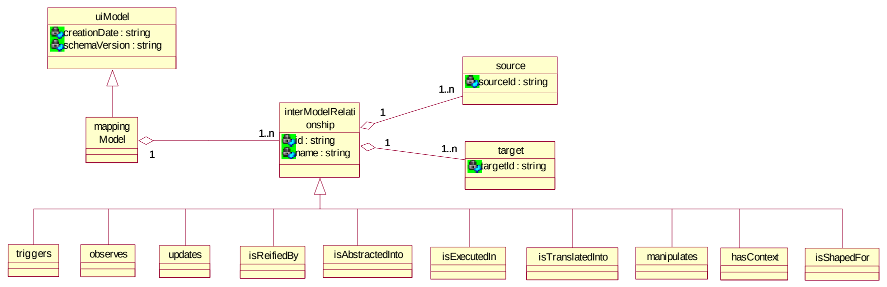


Figure D-11 Inter-Model Mappings

The *intermodel relationship* is any type of relationship established between one or many source models and one or many target models [USIX06]. A typical *interModelRelationship* is established between one source model and one target model, but it can be easily imagined that such a relationship can start from one source model to many target models, but from many source models to many target models.

An interModelRelationship is the super class of all possible relationships between models and elements of models. Consists of: one to many sources, one to many targets and the source should not necessarily be different from the target.

Several relationships [USIX06] can be defined to explicit the relationships between the domain model and the UI models (both abstract and concrete):

- *Observes* is a mapping between any UI component (at abstract or concrete level) and a domain attribute or instantiated attribute (at run time). Observes enables to specify that a UI component observes a value from the related domain concept. This mapping may be interpreted as follows: the content of a UI object must be synchronized when:
    - A mapped attribute is modified. The new state resulting from this modification should be presented on the UI (the notion of view could be of interest here).
    - A mapped method is executed. Its output parameters are displayed on the UI.
- *Updates* is a mapping between any UI component (at abstract or concrete level) and a domain attribute or instantiated attribute (at run time). Updates enable to specify that a UI component provides a value for the related domain concept.
- *Triggers* indicates a connection between a method of the domain model and a UI individual component (either at the abstract or at the concrete level)

Some other mappings are related to assure the transformations in order to achieve multi-path development of user interfaces. *Traceability mappings* are helpful for keeping a trace of the execution of the transformations. For instance it may be interesting to know which concrete object reifies which abstract object, or vice versa, which abstract object is an abstraction of which concrete object.

- *Is Executed In* maps a task to one or several AUI or CUI elements.
- *Is Reified By* indicates that a concrete object is the reification of an abstract one through a reification transformation.
- *Is Abstracted Into* indicates that an abstract object is the reification of a concrete one through an abstraction transformation.
- *Is translated Into* enables to provide a trace of the adaptation of one component in another, the transformation called translation.

Other useful mappings are:

- *Manipulates* maps a task to a domain concept. It may be an attribute, a set of attributes, a class (or an object), or a set of classes (or a set of objects). This relationship is useful when it comes to find the most appropriate interaction object to support a specific task.
- *Has Context* maps any model element to one or several contexts of use.
- *IsShapedFor* allows to associate a plasticity domain to a CUI.