Virtual Object Specification for Usable Virtual Environments

Shamus P. Smith
Department of Computer Science
Durham University
Durham DH1 3LE
United Kingdom
shamus.smith@durham.ac.uk

James S. Willans
Xactium
64 Struan Road
Sheffield S7 3EJ
United Kingdom
james.willans@xactium.com

ABSTRACT
The specification of virtual world objects is an important part of virtual environment design. However, identifying the required level of behaviour and geometric decomposition for virtual objects in a particular application is difficult and error-prone. This has implications for usability as the behaviour of objects gives strong cues to potential interaction. A fundamental step in tackling usability problems is the specification of virtual world objects early in the design phase. This paper presents a method for virtual object specification using task-based scenarios.

Author Keywords
Virtual environments, design, virtual world objects, specification, scenarios, requirements.

ACM Classification Keywords
I.3.7 [Computing Methodologies]: Computer Graphics – Three-Dimensional Graphics and Realism;

H.1.2 [Information Systems]: Models and Principles – User/Machine Systems.

INTRODUCTION

One ideal of virtual environment applications is to provide an environment where the user can interact freely within a simulated 3D space. Virtual environments attempt to build on latent human knowledge by providing cues that reuse appropriate representations (Smith and Harrison, 2001). The problem is that mental representations come with other expectations as to how the environment may behave - expectations that are not necessarily met. To reproduce a real environment in every aspect would demand incredible computational and graphics performance (Kalawsky, 1993; Kessler, 2002). It is likely to be some time before a system seamlessly integrating the visual, audio, tactile, olfactory and gustatory - the senses of sight, sound, touch, smell and taste - becomes available (Bowman et al., 2005; Smith and Harrison, 2001). Therefore trade-off decisions are invariably part of the virtual environment development process (Bryson, 1995). This is particularly the case when defining the veracity, or realism, of virtual world objects.

Grinstein and Southard (1996) observe that much of the time spent on developing a virtual environment is on modelling objects. This modelling is centred on defining three closely linked attributes, a virtual world object's appearance, geometry and behaviour. For an object, the appearance is given by the visual properties such as colour or texture, the geometry is the spatial structure of the object and the behaviour describes how the object evolves or responds to user events. In the design of virtual world objects there are direct trade-offs between these three attributes. For example, computational costs to manipulate complex objects, consisting of large numbers of geometric primitives, can be reduced by rendering objects as texture maps and not as individual primitives. However, interactive behaviour may require specific geometric structures and reducing these structures constrains the flexibility for possible behaviour specification. Also realistic renderings, via texture mapping, may imply that an object has associated complex behaviour that may be difficult to implement without access to the necessary geometric primitives.

It is typical for an object's appearance to be defined via a 3D modeller, for example AC3D[1] or 3DS Max[2], and imported into the virtual environment while behaviour is coded at a low level directly into a prototype application. Unfortunately, identifying the required level of geometric decomposition for objects in a particular application is a complex and error-prone problem. Also there is limited guidance available for designing virtual environments in general (Kaur, 1997; Mills and Noyes, 1999), let alone for object definition issues. Without design guidance, there can easily be a lack of consideration between an object's specification and its role in supporting user tasks. For example, if an object's geometric granularity is prematurely completed then the modelled object may not effectively support the behaviour needed for a particular task (Willans, 2001).

In this paper the requirements of virtual objects are considered in relation to interaction needs. In particular the benefits of a scenario-based approach to highlight task requirements in a design will be discussed. This allows

OZCHI 2006, November 20-24, 2006, Sydney, Australia.

OZCHI 2006 Proceedings ISBN: 1-59593-545-2

[1] http://www.ac3d.org/ [last access 13/6/2006].

[2] http://www.discreet.com/products/3dsmax/ [last access 13/6/2006].

the pre-implementation modelling of virtual world objects. The requirements of tasks in an environment can be used in the iterative specification of virtual object behaviour, appearance and geometric decomposition.

MODELLING VIRTUAL WORLD OBJECTS

An important part of the virtual environment development process is the modelling of virtual world objects. However, although systems contain mechanisms for describing the visual or spatial elements of their contents, i.e. the position and geometries of objects, surface properties, lighting, transparency, etc., there is little coherent means for attributing dynamics or behaviour to objects or their environments (Pettifer, 1999). It is common for the behaviours of objects in virtual environments to be described in terms of geometric transformations and they are often defined on a per object basis. Another difficulty can be illustrated in virtual reality modelling language (VRML) (Parisi et al., 1997) object modelling that allows behaviours to be attached to objects but only as predefined scripts. There is minimal provision for direct user input to object behaviour.

Tools for building virtual environments, for example toolkits, e.g. DIVE[3] and MAVERIK[4], and languages, e.g. Java3D[5] and VRML (Parisi et al., 1997), commonly import an object's geometry. However, this leaves the dynamics of specifying object behaviours to the application programmer. Such systems may *enable* rich behaviour for their virtual environments but do not *support* it (Pettifer, 1999). The application programmer either defines behaviour directly in a core language, e.g. "C" for MAVERIK, or some script based language, e.g. Java3D, VRML (Parisi et al., 1997) or Kallmann and Thalmann's (1998) *Smart Objects*. In each case, the behaviour definition is prototyped well into the implementation phase of development. However, the specification of object behaviour early in the design phase is a fundamental step in tackling usability problems in virtual environment development (Willans, 2001).

An initial step in the development of a virtual environment is to identify the objects that are required. A domain/task model may play an important role in this (pg511, Dix et al., 2004). An initial state of the world can be built using imported components from a 3D modeller. However, one of the aims of many virtual environments is to provide an environment with a rich level of interaction between the user and the virtual world objects. In principle, many objects will have behaviours that allow the user to interact with them, for example opening, closing, turning, sliding, etc. Unfortunately, 3D modellers are commonly independent of development environments and only provide the appearance, not the behavioural attributes, of objects. Therefore objects are only available as single grouped objects. Although objects may have a photo-realistic appearance, for example by utilising a texture with images of several components, they are considered by the development environment as one object with an associated bounding volume.

Suppose a designer wished to build a virtual kitchen to evaluate possible kitchen designs in terms of space constraints when cooking. This could be as an aid for structuring new home kitchen designs or for designing kitchens in confined spaces, for example in a submarine (Kaur, 1997). Initially, the designer could identify the basic objects that are required, e.g. cupboard units, ovens, benches, drawer units, hotplates etc. If real world examples of these objects are required to be simulated, a graphical designer could build these units in a 3D modeller and the designer could import and place these objects in various configurations for different virtual kitchen designs. However, one of the requirements of testing the kitchen space constraints is that when the cupboard doors are open it is still possible to easily move about the kitchen. Thus, the cupboards are required to have open/close behaviour. Unfortunately, the current virtual world objects are one unit, i.e. build in the 3D modeller. Before any open/close behaviour can be attached to the object's appearance, the objects have to be separated into the cupboard unit and the cupboard door. Similar issues could be identified with many of the kitchen units, e.g. sinks with taps, hotplates with dials, windows with latches etc. Although the example may seem trivial, the problem is scalable to a wide range of domains, for example the component decomposition of a virtual terrace apartment block with object behaviour requirements for doors, windows, lifts and mailboxes.

The example above identifies two main issues with the traditional approach to virtual environment development. Firstly, the consideration of an object's behaviour is required *before* 3D modelling of its geometry and appearance are completed. The granularity of an object's geometry is linked to the proposed behaviour of that object. However, defining the geometry and behaviour of objects at a high level of realism may be unnecessary (Isdale et al., 2002). Bowman et al. (pg333, 2005) observe that 3D user interfaces should deliver only as much realism as is needed for a particular application. If an object's behaviour is not needed for the tasks required in an environment then a detailed description may be redundant. Thus, secondly, the task requirements of the user *define* the behavioural requirements of any object. Of course, with luck, the environments' designer may have a thorough knowledge of what was wanted and may be able to design their units "right" first time. However, relying on the skill of a designer can be dangerous and open the analysis and design to bias (pg311, Leveson 1995).

---

[3] Distributed Interactive Virtual Environment (DIVE), SICS, Swedish Institute of Computer Science, Sweden, http://www.sics.se/dive/ [last access 16/06/2006].

[4] GNU MAVERIK, Manchester Virtual Environment Interface Kernel, AIG Group, University of Manchester, UK, http://aig.cs.man.ac.uk/maverik/ [last access 16/06/2006].

[5] Java3D. https://java3d.dev.java.net/ [last access 16/06/2006].

ELICITING VIRTUAL OBJECT REQUIREMENTS USING SCENARIOS

The use of scenarios[6] is common in the design of interactive systems as they describe activities or tasks in a narrative that allows exploration and discussion of contexts, needs and requirements (pg223, Preece et al., 2002) and force the designer to consider the appropriateness of the design (Dix et al. 2004). In addition, scenarios provide traceability through the design phases of a system (Benyon and Macaulay, 2002). Although scenario use is accepted practice for testing detailed requirements, the role scenarios play in helping to understand informal requirements and to specify them is the first place is less well understood (Potts et al., 1994). However a scenario-based approach can be used to tease out such initial requirements. Such an approach consists of developing scenarios of user tasks to elicit requirements information about the domain under study.

Developing scenarios is a useful exercise as "it exposes the range of possible system interactions and reveals system facilities which may be required" (Sommerville and Sawyer, 1997). Also scenarios provide a rich source of material for the construction of behaviour specifications (Wieringa, 1996). By developing scenarios, the focus is on the concrete behaviours of the components in the virtual environment, of both the user and any world objects, and as such can capture the "real" requirements needed for the design (Rolland et al., 1999).

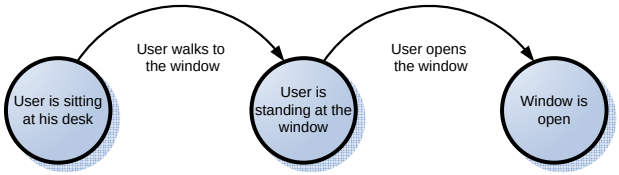

Figure 1. An example of a scenario description.

Since scenarios are informal descriptions of events, many representations can be used to realise their model. In this paper, visual state-transition descriptions are used because they are easy to use, i.e. construct and read, and lend themselves naturally to the description of scenarios. The states of the formalism describe a static representation of the environment and the transitions between the states describe the behaviour that occurs in order to transform the environment from one state to the next. An example scenario description can be seen in Figure 1.

Requirements analysis

Although scenarios document user requirements, these requirements are clouded by a clutter of non-relevant details - as far as the designers are concerned - as well as being spread across multiple representations. Interpreting the implications of scenarios directly to a design is complex and error-prone. To make the transition from requirements to design more reliable, the requirements that are important to a designer must be extracted and structured into a usable form (Willans, 2001).

For virtual environment designs, scenarios can be analysed to determine how their content influences the requirements of the virtual world objects i.e. object appearance, decomposition, behaviour and interactions with user behaviour. This process is informal, but the results of the analysis, the requirements of the environment, can be structured using a *requirements tree*. Prior to analysis there are two nodes in this requirements tree: the *environment* which is the root node, and the *user* which is a branch of the root node. The relation between a parent node and a child node in the tree should be read as *consists of*. Thus, initially the environment *consists of* a user. This is shown in Figure 2 (a)[7]. Two types of nodes can be added to the tree during scenario analysis. Firstly, object nodes that describe physical objects in the environment and secondly, behavioural nodes[8] that describe the behaviour of objects or the user interaction. A behavioural node must be a child of an object or the user. For each object added to the tree it is necessary to define its role within the environment. This can be one of three abstractions:

- *Background* objects are not critical to the scenario.
- *Contextual* objects are part of the scenario but not the focus.
- *Task* objects are central to the scenario.

During the process of analysing a scenario it may be the case that an object is initially described as one abstraction but, as the scenario unfolds, is *upgraded* to another. For instance, the *desk* object in Figure 1 is clearly *contextual* to that scenario, but another scenario in the same design may require that it becomes part of the main *task*.

The process of analysing the scenarios, to build the requirements tree, involves starting at the beginning of each scenario and, for each state and transition, considering how the description influences the requirements. This process is illustrated in the context of Figure 1. The first state describes that the *user is sitting at his desk*. The implication of this is that a *chair* and a *desk* object should be added to the tree as shown in Figure 2 (b). Both the added objects are described as contextual since they are not the focus of the scenario, i.e. opening a window.

The transition to the second state adds the requirement to the tree that the user has a behaviour enabling them to *walk to the window*. In addition, there should be a *window* in the environment that at this stage is contextual to the

[6] A scenario is an "informal narrative description" (Carroll, 2000).

[7] A requirements tree has a similar appearance to a *task description hierarchy* (pg521, Dix et al. 2004) but provides a combined representation of objects and required behaviours.

[8] Behavioural nodes are labelled *(be)* in Figure 2.

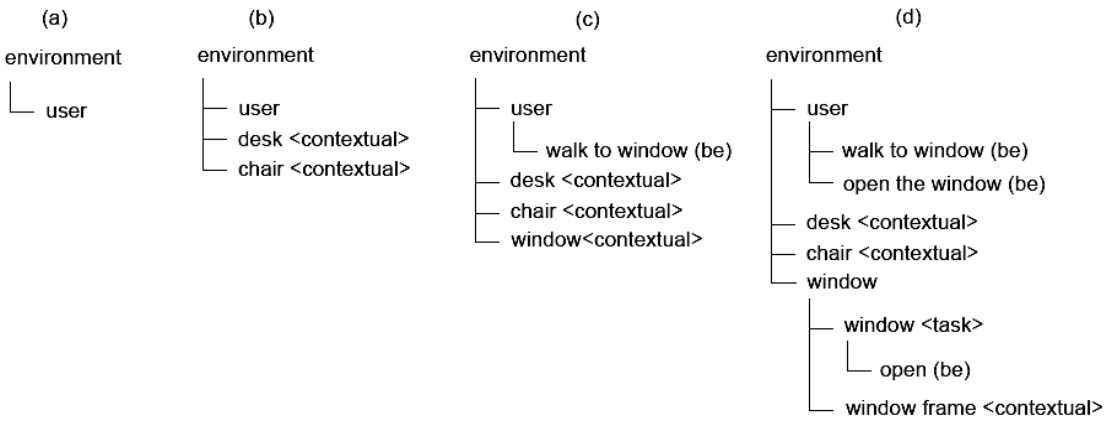

Figure 2. The evolution of the requirements tree as the example scenario is analysed.

scenario. The second state itself places no new requirements on the tree which is shown in Figure 2 (c)

The transition to the third state adds the requirements to the tree that the user has a behaviour which facilitates opening of the window. The necessity that the window should open decomposes its representation into a *window* and a *window frame*. The *window frame* is described as *contextual*, but the *window* itself is described as *task* since this is critical to the scenario. The *window* is also assigned the requirement that it supports the *open* behaviour. The final state places no new requirements on the tree, thus the complete requirements tree for this scenario is shown in Figure 2 (d).

Interpreting the requirements

As can be seen from Figure 2 (d), even for a single simple scenario the resulting requirements tree is a rich description of the requirements for an environment. In effect the process of transferring the knowledge from scenarios to the tree removes the ordering stipulated within the individual scenarios and collects together information pertaining to common phenomena, i.e. the behaviour of the user. Therefore an environment supporting the requirements of the tree, while facilitating the scenarios, is not constrained by the scenarios. For instance, given the requirements in Figure 2 (d), the user does not have to first sit at their desk before opening the window.

In terms of an *object's appearance*, the requirements tree defines the relative importance of each object to an environment (i.e. *background*, *contextual* or *task*). The importance of an object can be used to determine which objects should be rendered at a high-level of detail and those that should be rendered at a lower-level of detail. For instance in Figure 2 (d), most of the objects in the scenario are *contextual* to the task. If resources were scarce, the *contextual* objects could be rendered with fewer details.

The *decomposition of an object* is well defined in the requirements tree. For instance, in order to support the requirements that the *window* should open, the *window* object should be decomposed into a *window* and *window frame*. Similarly it can be seen from the tree that the *desk* and the *chair* can be modelled as single objects since the scenario does not place any contrary demands.

Although the behaviours of the *objects* and *user* are less concretely defined than the other aspects of the environment, the requirements tree gives a clear indication of which objects should support which behaviours and those that should be supported by the user's interactions.

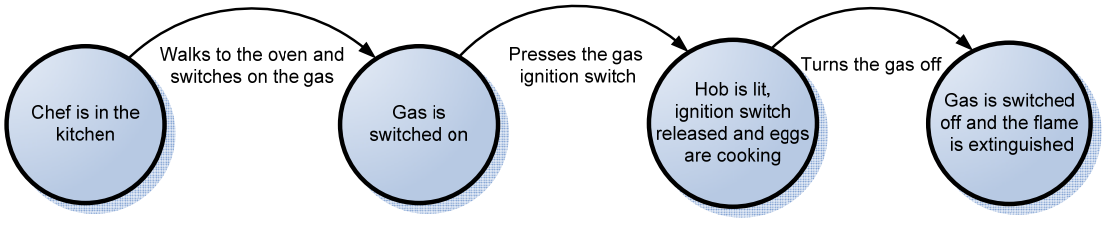

Figure 3. Using an oven to fry an egg scenario description

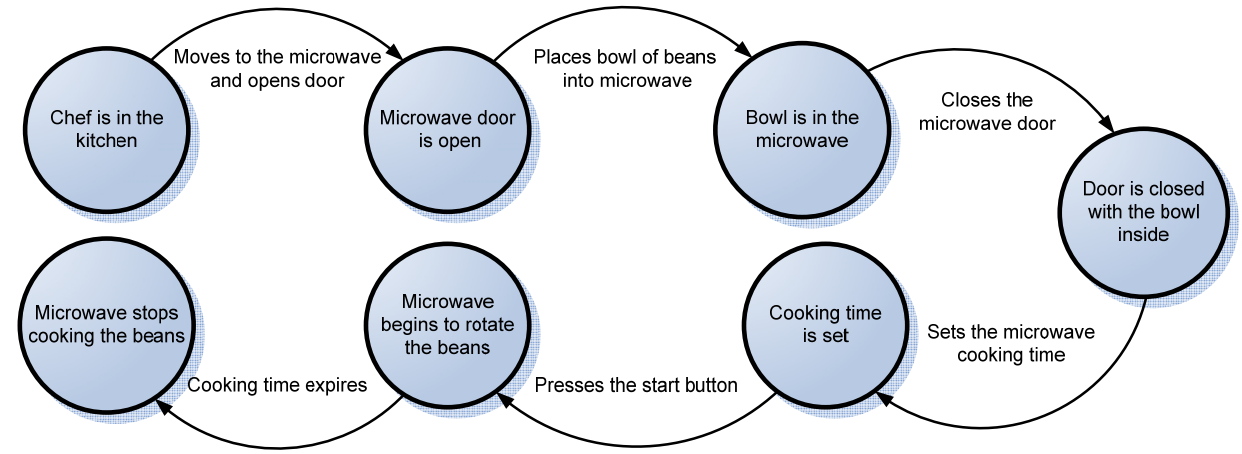

Figure 4. Using the microwave to heat beans scenario description.

CASE STUDY - TASKS IN A VIRTUAL KITCHEN

In this section the application of the approach with a larger example will be demonstrated. The problem statement addressed in this example is to "provide a virtual environment to train a chef to use typical kitchen equipment to cook a breakfast." For this problem statement two scenarios are derived dealing with (i) the frying of an egg and (ii) the heating of beans. Each scenario will extend the same requirements tree to build a requirements specification for the proposed design.

The scenario for frying an egg[9] using a gas oven (Figure 3) results in the requirements tree (Figure 5 (a)) containing an *oven* world object. The oven is decomposed into a *gas switch*, *ignition switch* and a *flame* to facilitate the behaviour associated with each of these world objects. In addition, there is an *oven unit* world object which supports no interaction. The *user* has associated behaviour which supports their interaction with the gas oven.

The scenario for cooking beans using a microwave (Figure 4) has augmented the requirements tree (Figure 5 (b)). Within this, the tree includes two additional world objects the *microwave* and the *bowl of beans*. The *microwave* is decomposed into an *on switch*, *timer* and a *door* to facilitate the behaviour associated with each of these world objects. There is also a *microwave unit* world object which has no associated behaviour. The *bowl of beans* world object is not decomposed further since it has no associated behaviour. Additional behaviour has been associated with the *user* in order to facilitate moving the bowl of beans to the microwave and the interaction with the microwave.

(a)

- environment
  - oven <task>
    - oven unit <contextual>
    - flame <task>
      - on <be>
      - off <be>
    - ignition switch <task>
      - press on <be>
      - release off <be>
    - gas switch <task>
      - turn on <be>
      - turn off <be>
  - user
    - release ignition switch <be>
    - turn gas off <be>
    - press ignition switch <be>
    - turn gas on <be>
    - move to oven <be>

(b)

- environment
  - bowl of beans <task>
    - move inside microwave <be>
  - microwave <task>
    - microwave unit <contextual>
    - on switch <task>
      - press <be>
    - timer <task>
      - set <be>
    - door <task>
      - open <be>
      - close <be>
    - food plate <contextual>
      - rotate <be>
  - oven <task>
    - oven unit <contextual>
    - flame <task>
      - on <be>
      - off <be>
    - ignition switch <task>
      - press on <be>
      - release off <be>
    - gas switch <task>
      - turn off <be>
  - user
    - set microwave timer <be>
    - place bowl in microwave <be>
    - move to microwave <be>
    - open microwave door <be>
    - close microwave door <be>
    - press microwave start button <be>
    - release ignition switch <be>
    - turn gas off <be>
    - press ignition switch <be>
    - turn gas on <be>
    - move to oven <be>

Figure 5. Including the (a) frying egg scenario and (b) microwave scenario in the requirements tree.

TOOL SUPPORT

The process of applying this approach by hand can be difficult and error-prone. Rarely do the scenario descriptions fit inside a state or transition, and it is necessary to rewrite the requirements tree for almost every step of the scenario analysis. To overcome these

[9] In this simple scenario there is an assumption that the egg is already in a pan on the hotplate. In addition it is the object's requirements are being modelled and not any associated timing issues i.e. the time needed to fry the egg.

pragmatic concerns, a tool called Primrose[10] has been developed (see Figure 6). Primrose is written in Java and has been tested for portability across UNIX and Windows based platforms.

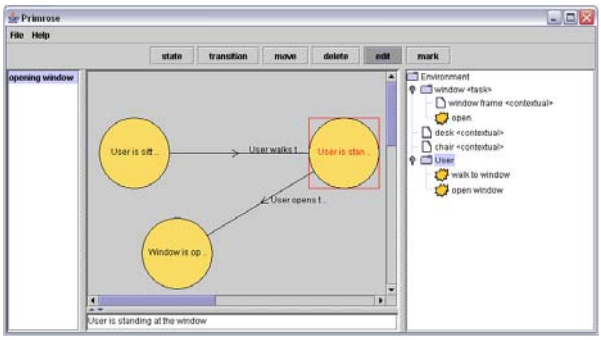

Figure 6. The Primrose tool.

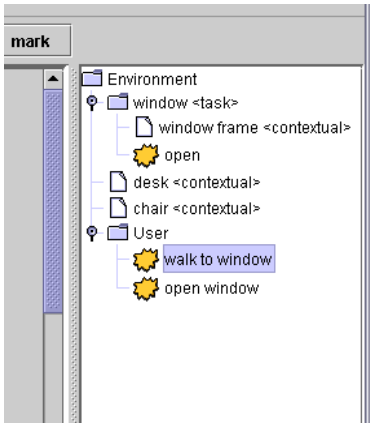

Figure 7. Primrose requirements tree.

At the left side of the tool is a list of the scenarios for the environment being designed and new scenarios can be added to this dynamically. In the centre of the tool is an area for creating the state transition description of the selected scenario. The descriptions of states and transitions are displayed in the text area below the diagram allowing long comprehensive descriptions of each step of the scenario. At the right hand side of the screen is the requirements tree that can be dynamically updated and revised during the analysis of scenarios (see Figure 7). Additionally, during scenario analysis it is possible to mark those states and transitions which have been considered to prevent duplicating work.

It is important to note that the approach presented here is not bound to one tool or particular representation. What is important is the process of defining object components so that they will support the required interaction in the target virtual environment.

TOWARDS IMPLEMENTATION

When looking at approaches that involve modelling and the refinement of models - see for example (Dubois et al., 2002; Massink et al., 1999; Willans and Harrison, 2000) - an important question is what level of detail is required in the models before a binding between the design and an implementation can take place (Smith and Duke, 2000). This is typically dependent on the constraints of the target implementation environment, for example in both hardware and software. An important consideration is the reuse of existing components. If libraries of objects, with predefined appearance and behaviour, are available extensive modelling in the design phase may be unnecessary. For example, VRML (Parisi et al., 1997) specifications may be available to provide a standard interaction for open/close behaviour of objects.

Unfortunately, this style of bottom-up design has dominated current views of virtual environment development, i.e. a technology centered design approach (Pettifer, 1999). Detailed world object specification is often overlooked in the rush to build working prototypes. However, such development strategies can be costly in the long term. The benefits of delaying the commitment to implementation are commonly championed in the software engineering literature (Sommerville, 2004). Changes in the implementation phase of software development are more costly for many reasons including the effort required to change and possible consequences to the existing design and implementation. Although designers should be aware of the possible constraints on a systems implementation, delaying the final binding to an implementation allows for a flexible approach to design and the associated benefits of design phase analysis and refinement.

One approach addressing the virtual environment design and implementation binding is proposed in (Willans, 2001). In this, the behaviour specifications of the requirements tree are refined into diagrammatic behavioural models using a graphical formalism called Flownets (Smith et al., 1999; Smith, 2006) and the requirement tree's visual requirements are built using a 3D modeller. These two types of specification are *plugged* together using a virtual environment development toolset called Marigold (Willans, 2001, Willans and Harrison, 2001) from which a prototype implementation can be generated – in this case directly using the MAVERIK toolkit.

The advantage of such an approach is that parameters of the environment can be changed in a “plug and play” fashion (Willans and Harrison, 2000) and the prototype re-evaluated. For example, new interaction devices can be tried in particular contexts in order to establish the extent to which they meet the requirements, or existing specifications can be tried to understand whether they meet the requirements demanded by the requirements tree.

A screenshot of the Marigold environment can be seen in Figure 8. This prototyping environment provides a visual notation for linking virtual world objects, both complex (*co* in Figure 8) e.g. toaster and oven and simple (*wo*) e.g. bowl and toast, with interaction techniques (*be*), e.g. select behaviour, with other virtual environment components such as interaction devices (*de*) and environment viewpoints (*vp*).

[10] The Primrose tool is publicly available at http://www.dur.ac.uk/shamus.smith/jwillans/primrose.zip.

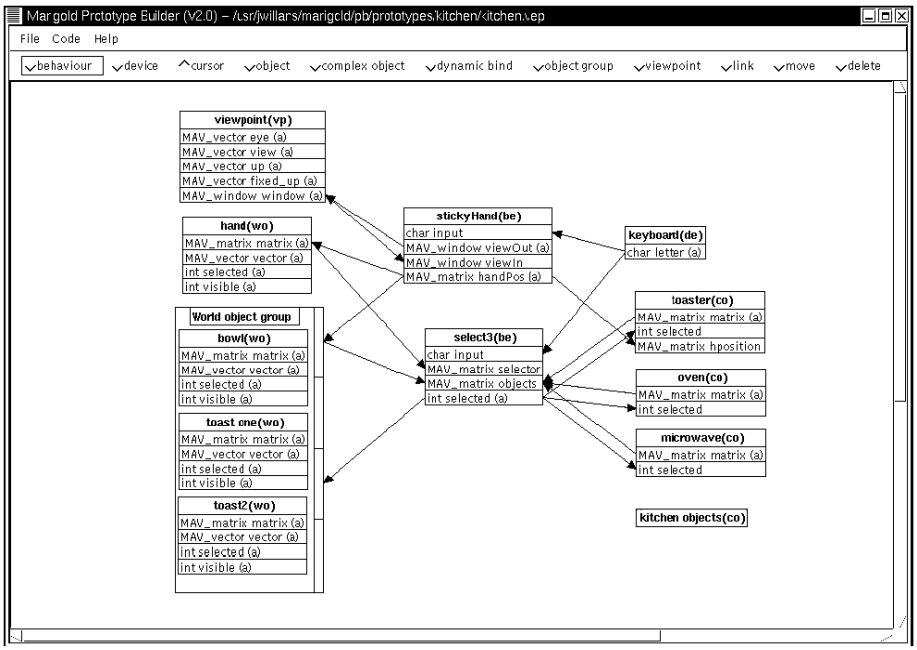

Figure 8. Marigold specification for a virtual kitchen.

The finished virtual environment is automatically generated via C code imported and executed in the MAVERIK toolkit.

A screenshot of a generated prototype environment driven by an extended requirements tree of a complete kitchen case study is shown in Figure 9. Further details on the Marigold toolset can be found in (Willans, 2001, Willans and Harrison, 2001).

Figure 9. Prototype implementation generated from a Marigold specification.

CONCLUSIONS

In this paper an approach to eliciting virtual environment object and user requirements using state transition based scenarios has been described. Scenarios have long been recognised as a good method of recording existing work practices which are easy for users to adopt (Ben et al., 1999). This is because the story-telling style of scenarios places few constraints on how they should be constructed. In the presented approach the user is describing interaction with the familiar domain of the real world. As such the scenarios have the additional desirable characteristic of abstracting from implementation factors which may hinder the elicitation. A criticism that can be applied to scenarios is that they do not necessarily expose all the requirements. Indeed this criticism can be applied to most requirements specification approaches since there is no guarantee that the specified requirements are indicative of the real requirements.

Although scenarios document the requirements of the virtual environment, these requirements are clouded by a clutter of non-relevant detail, for the designer, as well as being spread across multiple representations. Interpreting the implications of scenarios directly to a design is a complex and error-prone process. The requirements tree has been introduced as an intermediate representation that addresses this problem.

This paper has exemplified how scenarios can be mapped to the requirements tree, and argued that the coherency of the requirements tree can ease the design process. This is reinforced by the author's experience of implementing prototypes of the requirements tree using the Marigold environment.

As virtual environment technology and the environments they support become more integrated into mainstream software systems, usability based approaches will be increasingly important. For most computer systems, usability satisfaction with real users is a benchmark of success. Given the highly integrated context of virtual environments and their users, such a user-centred approach will become a required criterion if virtual environments are to meet expected usability standards.

ACKNOWLEDGMENTS

This work was supported in part by the EPSRC INQUISITIVE project (Grant GR/L53199) and the University of York. The authors are grateful to David Duke (University of Leeds) and Michael Harrison (University of Newcastle) who were both instrumental in the research reported in this paper.

REFERENCES

Ben, C., Mustapha Tawbi A. and Souveyet, C. Bridging the gap between users and requirements engineering: the scenario-based approach. Technical Report 99-07, CREWS Report Series, CRI Université Paris (1999).

Benyon, D. and Macaulay, C. Scenarios and the HCI-SE design problem. Interacting with Computers 14 (2002), 397-405.

Bowman, D., A., Kruijff, E., Poupyrev, I. and LaViola Jr, J. J. 3D User interfaces: Theory and Practise. Addison Wesley, USA (2005).

Bryson, S. Approaches to the successful design and implementation of VR applications. In R. A. Earnshaw, J. A. Vince, and H. Jones (eds), Virtual Reality Applications, Academic Press, London (1995), 9.1-9.11.

Carroll, J. M. Introduction to the special issue on "Scenario-Based Systems Development". Interacting with Computers 13, 1 (2000), 41-42.

Dix, A., Finlay, J., Abowd, G., D. and Beale, R. Human-Computer Interaction. Pearson/Prentice Hall, Harlow, England, third edition, (2004).

Dubois, E., da Silva, P. P. and Gray, P. Notational Support for the Design of Augmented Reality Systems. In Forbrig, P., Limbourg, Q., Urban, B. and Vanderdonckt, J (eds) Interactive Systems: Design, Specification and Verification, Springer LNCS 2545 (2002), 74-88.

Grinstein, G. G. and Southard, D. A. Rapid modeling and design in virtual environments. Presence 5, 1 (1996), 146-158.

Isdale, D., Fencott, C., Heim, M. and Daly, L. Content Design for Virtual Environments. In K. M. Stanney (ed) Handbook of Virtual Environments: Design, Implementation and Applications, Lawrence Erlbaum Associates (2002), 519-532.

Kalawsky, R. S. The Science of Virtual Reality and Virtual Environments. Addison-Wesley (1993).

Kallmann, M. and Thalmann, D. Modeling objects for interaction tasks. In B. Arnaldi and G. Hegron (eds) Computer Animation and Simulation '98, Springer-Verlag Wien (1998), 73-86.

Kaur, K. Designing virtual environments for usability. In S. Howard, J. Hammond, and G. Lindgaard (eds) Human-Computer Interaction: INTERACT'97, Chapman and Hall (1997), 636-639.

Kessler, G. D. Virtual Environment Models. In K. M. Stanney (ed) Handbook of Virtual Environments: Design, Implementation and Applications, Lawrence Erlbaum Associates (2002), 255-276.

Leveson, N. G. Safeware: System Safety and Computers. Addison Wesley (1995).

Massink, M, Duke, D., and Smith, S. Towards hybrid interface specification for virtual environments. In D. J. Duke and A. Puerta (eds) Design, Specification and Verification of Interactive Systems '99, Springer-Verlag/Wien (1999), 30-51.

Mills, S and Noyes, J. Virtual reality: an overview of user-related design issues revised paper for special issue on "Virtual reality: User Issues'" in Interacting with Computers. Interacting with Computers 11 (1999), 375-386.

Parisi, A., Bell, G. and Pesce, M. VRML, the virtual reality modelling language. International Standard ISO/IEC 14772-1:1997, (1997).

Pettifer, S. An Operating Environment for Large Scale Virtual Reality. PhD thesis, University of Manchester (1999).

Potts, C., Takahashi, K. and Anton, A. Inquiry-based scenario analysis of systems requirements. Technical Report GIT-CC-94/14, Georgia Tech (1994).

Preece, J., Rogers, Y. and Sharp, H. Interaction Design, John Wiley and Sons, USA (2002).

Rolland, C., Grosz, G., and Kla, R. Experience with goal-scenario coupling in requirements engineering. In F. M. Titsworth, (ed) RE'99: 4th IEEE Int. Sym. on Requirements Engineering, IEEE (1999), 74-81.

Smith, S. P. Exploring the specification of haptic interaction. Design, Specification and Verification of Interactive Systems 2006. (Forthcoming: Springer Lecture Notes in Computer Science).

Smith, S. P. and Duke, D. J. Binding virtual environments to toolkit capabilities. Computer Graphics Forum 19, 3 (2000), C81-C89.

Smith, S. P., Duke, D. J. and Massink, M. The hybrid world of virtual environments. Computer Graphics Forum 18, 3 (1999), C297-C307.

Smith, S. P. and Harrison, M. D. Editorial: User centred design and implementation of virtual environments. Int. J. of Human-Computer Studies 55, 2 (2001), 109-114.

Sommerville, I. Software Engineering. Seventh edition, Addison Wesley (2004).

Sommerville, I. and Sawyer, P. Requirements Engineering: A good practise guide. John Wiley and Sons, England (1997).

Wieringa, R. J. Requirements engineering: Frameworks for understanding. John Wiley and Sons, England (1996).

Willans, J. S. Integrating behavioural design into the virtual environment development process. PhD thesis, University of York (2001).

Willans, J. S. and Harrison, M. D. Verifying the behaviour of virtual environment world objects. In P. Palanque and F. Paterno (eds). Interactive Systems: Design, Specification, and Verification. LNCS 1946, Springer (2000), 65-77.

Willans, J. S. and Harrison, M. D. A toolset supported approach for designing and testing virtual environment interaction techniques. Int. J. of Human-Computer Studies 55, 2 (2001), 145-165.