A SOFTWARE TOOLKIT FOR WEB-BASED VIRTUAL ENVIRONMENTS BASED ON A SHARED DATABASE

B.W. van Schooten
University of Twente, Faculty of EWI, TKI group
P.O. box 217, 7500 AE, Enschede, Netherlands
schooten@cs.utwente.nl

A. Nijholt
University of Twente, Faculty of EWI, TKI group
P.O. box 217, 7500 AE, Enschede, Netherlands
anijholt@cs.utwente.nl

E.M.A.G. van Dijk
University of Twente, Faculty of EWI, TKI group
P.O. box 217, 7500 AE, Enschede, Netherlands
bvdijk@cs.utwente.nl

ABSTRACT

We propose a software toolkit for developing complex web-based user interfaces, incorporating such things as multi-user facilities, virtual environments (VEs), and interface agents. The toolkit is based on a novel software architecture that combines ideas from multi-agent platforms and user interface (UI) architectures. It provides a distributed shared database with publish-subscribe facilities. This enables UI components to observe the state and activities of any other components in the system easily. The system runs in a web-based environment. The toolkit is comprised of several programming and other specification languages, providing a complete suite of systems design languages. We illustrate the toolkit by means of a couple of examples.

KEYWORDS

Virtual Environments, Software Architectures, Entity-Relationship model, Specification Languages.

1. INTRODUCTION

We present a novel software toolkit for the development of complex web user interfaces (UIs) by building on top of the regular web model. The UIs we focus on are *virtual environments*, which include multi-user UIs, interface agents, and highly interactive graphical (such as 3D) UIs. The software can be used with regular web browsers, using Java applets embedded in regular web pages representing the UI components.

It comprises of a set of specification and programming languages (which we call a *specification technique*) which provide a complete development suite.

One of the main novelties of our software toolkit is the concept of using a relational database for modelling the *structure and data of the UI*. This is in contrast with the common use of a database for modelling the *information domain*. This means that each UI component is an entity in the data structure, and all UI data, such as mouse coordinates, mouse clicks, checkbox selections, and button presses, is communicated through the database.

In order to accomplish this, we use a specialized database, with fast database join operations, publish-subscribe facilities, and low latency. The rationale for this will be explained in the next paragraph.

In multi-user UIs, it is well known that users have to see what the others are doing in order to cooperate appropriately. This is expressed in the well established WISIWYS (What I See Is What You See) principle

(Stefik et al., 1987). The users are typically able to track each other's behaviour through one or more graphical and textual `viewports'. A similar principle may be formulated for creating an effective interface agent that complements a traditional (G)UI. Consider Lieberman's Scriptagent (Lieberman, 1998), where a conversational agent complements a GUI by observing user behaviour and operating the GUI proactively when necessary. Such an interface agent needs to have access to the data structure and operations of the UI in a manner easily readable for a software agent. A similar issue is found in agents inhabiting virtual worlds, which need to have an appropriate level of awareness of the world they inhabit, with the world being analogous to what is regularly the UI. For example in (Robert et al., 1998), objects in a virtual world are tagged with values which enable agents to extract some semantics from the objects. In their terminology, the agents are thus *grounded* in the world. A similar scheme is found in (Doyle and Hayes-Roth, 1998), in which virtual world objects are annotated with information useable by interface agents. In fact, using symbolic, relational representations are often found in multi-agent systems to describe the agents' world structure. They usually take the form of Prolog-like tuples. We find them for example in mVITAL (Anastassakis et al., 2001) and in OPS-5 (Cooper and Wogrin, 1988). Our model tackles this mutual awareness issue by focusing on the UI and the (human and computer) agents interacting with it as a shared data structure, which is then modelled using traditional data structuring techniques. The software components in the application can easily publish relevant data, and read and subscribe to the data they need.

The database model is also a powerful model to tackle the standard UI architectural problems. It enables UI component structure to be specified in an abstract way that can nevertheless be used directly in an implemented system. Additionally, we use entity-relationship diagrams (ERDs), which are generated from a formal specification of the application's data structure. ERDs are readable by non software engineers, which make them potentially useful for evaluating usability aspects. An example of the (informal) use of ERDs for UI structure for usability purposes is found in the *Entity Relationship Modelling of Information Artifacts* (ERMIA) (Green and Benyon, 1996) method. Unlike ERMIA, our model provides executable specifications to go with the ERDs.

Structural specifications in terms of E/R models may bridge the gap between first design stages and a full implementation of a component structure, by making clear the pattern that the structure should conform to. By specifying relational multiplicities, E/R models are suitable for modelling the patterns of dynamical creation and deletion of components.

Using the database view facility, the components are also naturally able to deal with dynamically-changing sets of values as input.

In section 2, we will describe our toolkit in detail. In section 3, we will give examples that illustrate the use of the toolkit.

2. THE MODEL

In this section, we describe our toolkit, which is called VETk (Virtual Environment Toolkit). In VETk, software is described as a number of interacting software components, called *agents*. Note that the VETk agent is a communication modelling concept, and does not concern the implementation of the behaviour of an agent, which may be arbitrary. A VETk agent has as its basis concurrent database access and subscription facilities, which forms a good basis for the implementation of UI update propagation schemes and environmental awareness.

VETk is related to other UI architectures, which also providing structuring and information propagation models for UIs. Well known are the Seeheim-like models, such as MVC and PAC, but there are many others, including groupware models such as PAC* (Coutaz, 1997) and COCA (Li and Muntz, 1998). Of these, the VETk model is closest to dataflow-type models, such as Amulet (Myers et al., 1997) and VRML (Carey and Bell, 1997). These have a concept of *node* (a kind of software component), with each node encompassing a number of *attributes* or *fields*, which are variables that can be read or changed. A change of an attribute can trigger activity of other nodes, which is achieved through a mechanism analogous to publish-subscribe. VETk adds to this concurrency, database-type access facilities, and hence, easier support for multiple agents and arbitrary and dynamically changing structures.

A VETk agent has a number of input and output points. Data arriving at an input may trigger agent behaviour and arbitrary output. An agent may also act spontaneously without input, or channel input coming

from a user. Agents interact through a shared database. Their inputs are coupled to the database by defining database views (aka subscriptions). Their outputs are translated to database operations by means of the execution of scripts. An agent's scripts are defined when the agent is created, and define its embedding in the environment.

Since the agents' inputs correspond to database views, they represent *sets* of values. The agent can process these sets at will, and changes to these sets are notified automatically. Such a fundamental support for sets as input is useful for various basic UI components, as well as for modeling interface agent knowledge. Examples are UI components that deal with sets of elements, such as listboxes (producing a clickable list of values), canvases (displaying a set of objects), multi-user components, such as multi-user scrollbars (displaying a scrollbar for each user), and interface agents that track users' behaviour. In fact, the idea of coupling database views directly to UI views is well used in Web interfaces (Moulding, 2001). In these cases however, the database contains only (relatively static) domain data, rather than UI data.

Figure 1 illustrates the `database' oriented nature of a VETk agent. An agent has a number of views (which correspond to database tables, or lists of tuples) and output points (with each output point corresponding to a tuple with specific size and element types).

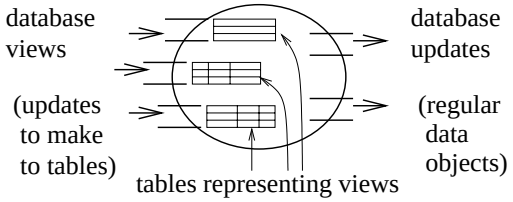


Figure 1. VETk's `database oriented' agent.

Figure 2 shows an example agent, which is a dialogue agent that is able to track multiple users' actions in their user interfaces, and is able to generate tables with information. This dialogue agent happens to be one of the agents in the example in section 3.2. Its main mode of communication is by sending and receiving language utterances (*user_utterance* and *utter*). It can create tables by means of *new_table* and *table_item*. Additionally, it tracks user behaviour: users pointing to objects, and viewing information, indicating certain topics are salient (*user_pointing* and *user_viewing*), and users selecting an item from the agent's own table (*item_selected*).

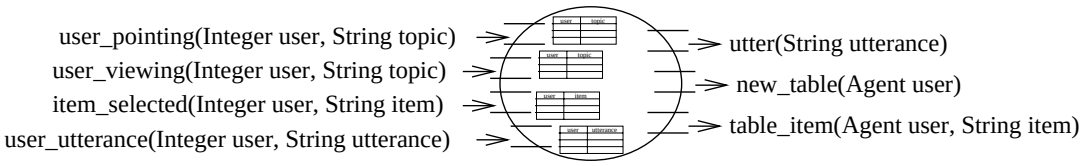


Figure 2. Example `multimodal dialogue' agent.

2.1 The specification languages

The VETk specification technique consists of several languages which serve different, complementary, purposes. While a detailed description of these falls outside the scope of this paper, we will give a short overview of them to give some idea of the specification technique as a whole. Note that most of the languages require a lengthy technical introduction in order to understand them. For this reason, we will limit ourselves to the ERD specifications in this paper. For detailed language description and examples, see (van Schooten, 2003).

We have three classes of languages: system properties constraints (specifying properties that the desired application should conform to, including ERD structure and multiplicity), glue languages (specifying how a set of agents is glued to specify an application), and agent behaviour languages. Table 1 shows the kinds of

specifications currently supported by the technique. At the left side are the types of specifications that are supported, in the middle are the languages in which they may be specified, at the right side what level of experience is required to use the specifications properly. We indicate three experience levels here: user (neither a systems designer, nor a programmer), designer but non-programmer (i.e. a domain expert, or a UI designer), and programmer (someone who is experienced in programming).

System property constraints		
ERD of user interface	VDC (VETk Data Constraints)	Designer (non-programmer)
State automata of UI objects	VDC	Designer (non-programmer)
Logic constraints on any data in database	VDC	Programmer
Glue languages		
Visual layout	HTML with embedded agents and VETkScript	Designer, Programmer (writing), User (reading)
Main glue language	VETkScript	Programmer
Individual agent behaviour		
Agent's interface	VCL (VETk Component Language)	Programmer
Agent behaviour	Java language and environment	Programmer

Table 1. VETk's specification languages

Each application typically has one VDC specification, specifying the global constraints on the application. For each UI window, there is one HTML document, which embeds the UI agents in the form of Java applets. Next to the HTML documents, there may be any number of separate VETkScripts, which may be executed whenever desired. For each type of agent, there is a VCL specification, which specifies how the agent reacts to events in its environment. Internally, each agent has full access to the Java language to specify its behaviour.

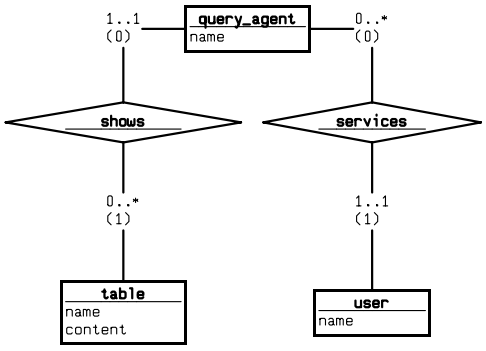
The system is based on an engine that implements the database and scripting facilities. It runs as a separate process. Agents interact with the engine after establishing a TCP/IP connection. We enable agents to be run both as stand-alone Java applications and as applets in a Web page, with each agent being a separate applet. Both applications and applets may open windows using the appropriate VETkScript commands, while applets may additionally load HTML pages (with further agents embedded in them) through a request to the browser. A separate Web server serves the relevant HTML and VETkScript files.

3. EXAMPLES

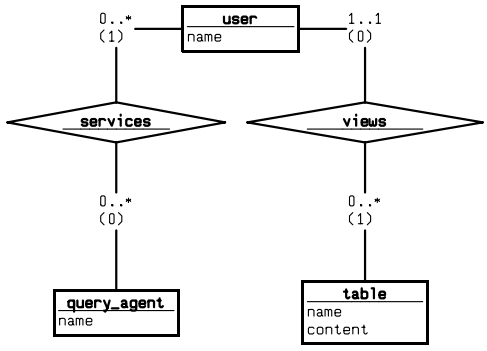
We will discuss two example applications to illustrate the usage of our model. The first example is an abstract example illustrating design alternatives of a multi-user information agent system. The second example is a multi-user virtual environment embedded in a web environment, which contains a dialogue agent.

3.1 Multi-user information agents system

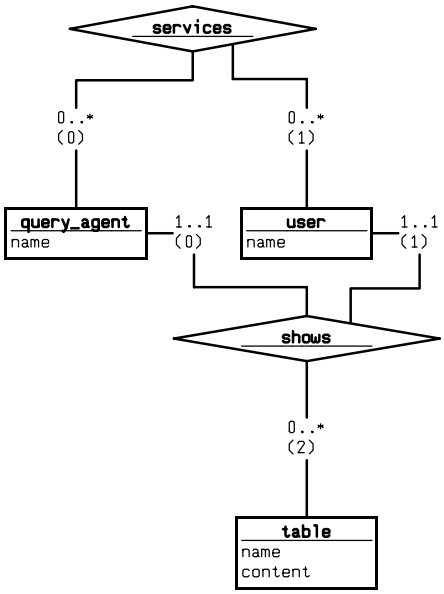
Consider an application in which multiple users may query information agents, which may present their results in the form of tables. The simplest case is a search engine like Google, with the search bar being the place where users can query the agent, and the table of search results being the agent's response. In case we have multiple users and query agents, we may have various design alternatives. We may set up the query agents to be local and private to each user, or they may be global, interacting with multiple users at once. This for example becomes relevant when the agents are actually visible to the users, such as in a virtual world with embodied agents. We may present the result tables as just a bunch of pages, or as visibly related to the agents that generated them. For example, if tables are included in the same window as the search bars, it is made visible which table belongs to which search engine. From the agents' perspective, the agents may be aware of the different users present, and may for example give information about one user to another user if this is deemed useful.



Alternative 1. Each user may interact with some private query agents, and each agent may be showing some tables.



Alternative 2. Each query agent is global, serving multiple users, just as each user may be served by multiple such agents. Each user may be viewing some tables. It cannot be made out which table belongs to which agent.



Alternative 3. Again, each query agent is global. Each table is now associated with both user and query agent through the ternary *shows* relation.

Figure 3. Conceptual design alternatives for an interactive query system.

In the above ERDs, the boxes are entities, and the rhomb shapes are relations. The arcs are annotated with the relational multiplicities (plus a number indicating the reference's parameter number, which is only useful for implementation purposes).

Visibility and reachability relations such as those described above can be expressed in an abstract manner in an ERD. The presence of a relation in the ERD makes it possible to design the system so that certain users or agents are able to find the thus related entities. How exactly a relation is expressed in the final system is still a choice of the designer.

Three design alternatives are given in the ERDs in figure 3. In alternative 1, each query agent only services one user at a time (as given by the multiplicity of the services relation), and may produce its own set of tables.

In alternative 2, each agent services multiple users simultaneously. Both the users and the agents may be made aware of the multiplicity of users being serviced by each specific agent. The tables belong to the users and cannot be traced to the agent which generated them. In alternative 3, the tables are related to both the agent that produced them and the user that views them.

3.2 A Web-based VE with dialogue agent

The example we discuss in this section is the Mini Virtual Music Centre, which is inspired by the Virtual Music Centre system (Nijholt and Hondorp, 2000), a Web-based VE. This is a full-fledged application, including a virtual world, multiple users, and a dialogue agent embodied in the world. We will give a short description of the system, a screen grab, and an ERD. Figure 4 shows the screen layout. The ERD is explained in detail in figure 5.

Mini-VMC is a 2D VE combined with a Web environment consisting of a number of HTML pages. There is a dialogue agent situated in the VE, which is based on the VMC Karin agent. It tracks the users' Web browsing behaviour and pointing at objects in the VE and hyperlinks, and answers queries about theatre performances. It may also show a list of query results, which are displayed in a table. Results may be selected by clicking on them. The VE consists of a set of rooms with objects in them, which is displayed as a background image with the objects displayed as images, placed in front of it. The objects may be users, exits which can be clicked on to move to other rooms, and objects that bring up Web pages when clicked. Karin is also an object. The user may select user name and an avatar, and may move the avatar around by clicking on the room. Karin and the users may chat by means of text.

The Mini-VMC ERD describes two information structures: the structure of Web pages being browsed (given by *link*s linking *webpage*s, and the structure of the VE world (given by *room*s, *containing object*s, some of which may *exit_to* other *room*s).

Beside these, there are some structures for describing UI state. First, one should note that avatars and interface agents are *object*s. Agents may be *pointing_at* objects and *pointing_at_link*s. They may have *loaded_webpage*s, as *loaded_from* specific *webpage* documents on the server, which they may be *view*ing. Finally, and agent may show a *table_item* to another. As this text paraphrases the diagram almost literally, the structure diagram should be quite intuitive to understand. Note though, that, in order to simplify the model, we have used entities with references to model *table_item* and *link*, rather than use extra relations to link these entities.

Figure 4. Screen layout of Mini-VMC

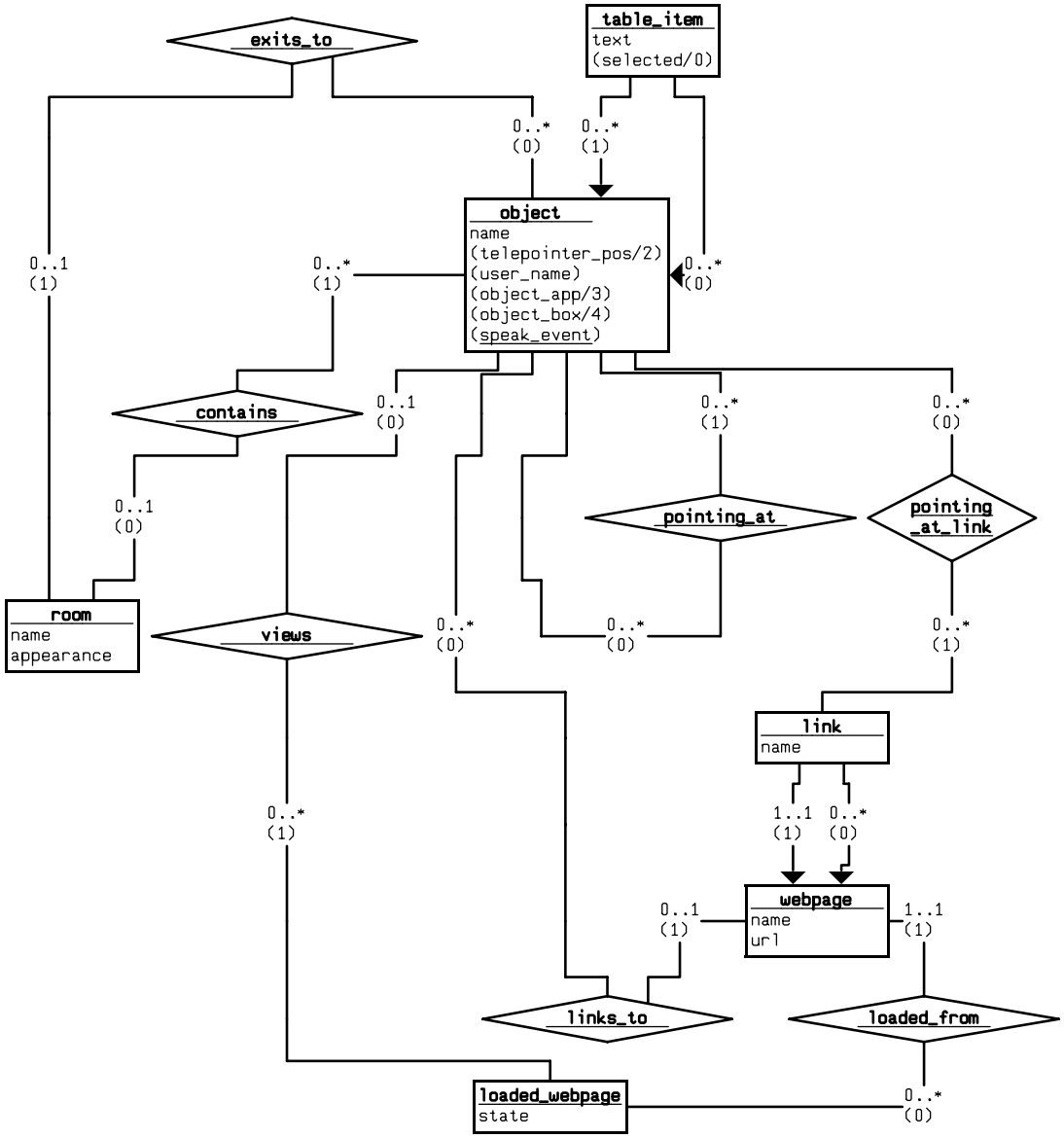


Figure 5. ERD of Mini-VMC

This ERD has some extra features not found in figure 3. Bracketed attributes indicate optional values. Bracketed and underlined attributes indicate events. The numbers following the '/' after these attributes indicate the number of arguments, i.e. '/0' means the attribute indicates just a signal without content, '/2', a 2-tuple. The default is '/1'. References from entities to entities are indicated by arcs with arrows pointing to the destination of the reference.

4. CONCLUSIONS

We presented an agent-oriented specification technique for developing web-based virtual environments with multiple users and interface agents. In this technique we implemented the idea of using structure

specifications and a relational database as an integral part of UI development. The technique provides a close coupling between abstract and early specifications and final system implementation, which makes it well suited for bridging the gap between first design and implementation.

The technique proved effective for specifying a number of example applications, of which we showed some here. While direct comparison with other methods that involve different approaches is hard, we have at least shown that this method has potential, and is worth pursuing further. The current software toolkit still has some inconveniences that may be solved without further evaluation, such as error reporting problems. But, after these are solved, the toolkit may be further evaluated by offering it to a larger user base.

REFERENCES

Anastassakis, G. et al., 2001. Virtual agent societies with the mVITAL intelligent agent system. *Lecture Notes in Computer Science*, Vol. 2190, pp 112-125.

Carey, R. and Bell, G., 1997. *The Annotated VRML 2.0 Reference Manual, 1st Edition*. Addison-Wesley.

Cooper, T.A. and Wogrin, N., 1988. *Rule-based programming with OPS5*. Morgan Kaufmann publishers.

Coutaz, J., 1997. PAC-ing the architecture of your user interface. *Proceedings of the 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems*, pp 15-32. Springer, Addison-Wesley.

Doyle, P. and Hayes-Roth, B., 1998. Agents in annotated worlds. *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pp 173-180. ACM Press.

Green, T. R. G. and Benyon, D., 1996. The skull beneath the skin: entity-relationship models of information artifacts. *International Journal of Human-Computer Studies*, Vol. 44, No. 6, pp 801-828.

Li, D. and Muntz, R., 1998. COCA: Collaborative object coordination architecture. *CSCW98: The 1998 ACM conference on computer supported cooperative work*.

Lieberman, H., 1998. Integrating user interface agents with conventional applications. *International Conference on Intelligent User Interfaces*.

Moulding, P., 2001. *PHP Black Book*. Coriolis.

Myers, B. A. et al., 1997. The Amulet environment: New models for effective user interface software development. *IEEE Transactions on Software Engineering*, Vol. 23, No.6, pp 347-365.

Nijholt, A. and Hondorp, H., 2000. Towards communicating agents and avatars in virtual worlds. *Proceedings of EUROGRAPHICS 2000*, pp 91-95.

B. van Schooten, 2003. *Development and specification of virtual environments, Ph.D. thesis*. Neslia Paniculata, Enschede, Netherlands.

Robert, A. et al., 1998. Grounding agents in EMud artificial worlds. *Proceedings of the 1st International Conference on Virtual Worlds (VW-98)*, Vol. 1434 of LNAI, pp 193-204. Springer.

Stefik, M. et al., 1987. WYSIWIS revised: Early experiences with multi-user interfaces. *ACM Transactions on Office Information Systems*, Vol. 5, No. 2, pp 147-167.