

A software toolkit for web-based virtual environments based on a shared database

Boris van Schooten, Anton Nijholt, Betsy van Dijk



Parlevink Group

Dept of Human Media Interaction

University of Twente

Netherlands

Here I will tell you something about the software toolkit that I developed as part of my Ph.D. work.

I am Boris van Schooten, I work at the university of Twente in the Netherlands. Our research group is called Parlevink, with Anton Nijholt being the head of the group.

Parlevink is now part of the newly renamed Human Media Interaction department.

A software toolkit for web-based virtual environments based on a shared database

Contents

- Introduction
- Rationale
- The toolkit explained
- Examples
- Conclusions

First we give an introduction to the subject matter, answering questions like:

What are web based ves? And what is a shared database?

Then we give a rationale of our toolkit, followed by an explanation of its details.

We conclude with a couple of examples and concluding remarks.

Introduction

Web-based Virtual Environments (VEs)



The term **Web-based Virtual Environments** covers a range of UIs in which the internet, multiple users, a "shared virtual world", play a role.

There are various examples of these. Typical applications are: teaching and learning, collaboration, and entertainment.

Popular are multi-user games and meeting places. **Activeworlds** is one of the biggest.

We also have our own system, the **virtual music centre**, which is an information providing environment.

Below are some other systems, for example this is a **virtual surgery environment**, a **deepmatrix chat room**, and on the bottom right is one of the example programs written in our toolkit.

Characterisation:

- Information browsing

ex. Web pages, VR

- Graphical and 3D

ex. VRML, 3D browsers

- Multi-user (synchronous)

ex. Chat, online games, web virtual environments

- Interface agents

ex. mainly web virtual environments

We characterise the **features** of a web based VE as follows.

They are typically information browsing environments. Most of the www is web pages, but VR environments are often another kind of information browsing environment.

VEs are often though not always heavily graphical. Web browsers already have some support for 3D graphics, for example we have the VRML standard and certain 3D browsers.

Many VEs are multi-user. The Web has a lot of potential for multi-user software, though most of it is in the form of sending messages that can be read later (i.e. asynchronous groupware). What is typical of VEs though is that it is synchronous groupware. We already find various synchronous groupware on the internet, for example chat, online games, web virtual environments.

A fourth and less common characteristic is the occurrence of interface agents, that is, computer agents that present themselves as human-like. We do find them in online games and sometimes in online chat software. Sometimes they are used to present a more "traditional" web application in a more human-like way.

Rationale

Other user interface (UI) architectures:

Component and agent architectures, user interface toolkits
(Amulet, VRML, Marigold)

VETk: a shared SQL type database for passing UI information.

Advantages

- Usability
- World structure modelling
- Software engineering

In our model, we use a SQL type relational database to pass any and all information within the system. This includes GUI events and semantic information. We may compare this to other software modelling approaches and toolkits. We have various component and agent architectures, which usually concern the communication model for distributed software. While these are not UI-specific and do not use a shared database like we do, they did inspire some of the ideas of VETk.

user interface toolkits, which provide more UI and VE specific tools, such as UI event propagation models. Similar but simpler forms of structure-based modelling are found here. None uses a shared database.

[Game programming suites are also a good source of VE-specific software solutions]

Our toolkit has the following potential advantages: We may distinguish three reasons for this:

it enables specifications useful for usability

it solves software engineering issues

it provides a model for the virtual world structure.

Usability advantages

- **UI structure modelling.**

A model of the UI structure is useful usability information.

Like **ERMIA** method, uses **structure diagrams**.

Additionally: **executability**

- **Prototypability.**

So, what are the usability advantages?

Firstly, I'd like to argue that modelling the structure of a UI is useful for a usability study. This has already been illustrated by the ERMIA method.

In addition to the ERMIA method, the toolkit enables the structure diagrams to be part of the executable system.

Furthermore, the toolkit enables quick mock-ups and prototypes to be built.

World structure modelling

Software structure \Leftrightarrow VE (semantic) structure.

Readily maps to semantic structure as found in various VE systems.

Works like "ontology": world structure used by dialogue system in one of our examples.

Our model enables a direct mapping between software architecture and semantic structure.

Our software structure may be mapped to the structures used in various existing systems. Examples abound: Robert '98, Doyle '98, mVITAL, OPS5 examples.

Note that this kind of structure works like a shared ontology as found in AI systems.

In the software we wrote, we even used the database information directly to derive the dialogue context for a dialogue system. This enabled the dialogue system to see what the user is viewing and even enabled it to track whether the user was pointing at something in the virtual world.

Software engineering solutions

- UI event / update architecture
 - *Structuring method*
 - *Easy support for database tables (sets of tuples)*
 - *Publish/subscribe facility*
 - *Dynamical creation and deletion of UI components*
- Distributed software model
 - network communication is transparent*

Within the domain of software engineering, our VEs pose a number of common issues.

One is the UI event problem: how do we propagate events and ensure that all views within the UI remain updated? For this problem we provide several facilities:

Firstly a structuring method. A support for working with DB tables / sets of tuples, which is useful for even the simplest UI components. That is combined with a publish-subscribe facility so that updates can be tracked automatically. The fact that the existence of components can be checked by database queries implies easy support for the administration concerning the dynamical creation and deletion of UI components.

A second is the distributed software problem: how do we ensure that our parallel software remains tractable? For this we offer a network-transparent communication model.

The VETk toolkit

Software component is called **agent**. Each agent runs in **parallel** and has a number of **inputs** and **outputs**.

Each agent is "**glued**" into system by "glue" scripts, enabling **separation** of software components and system architecture.

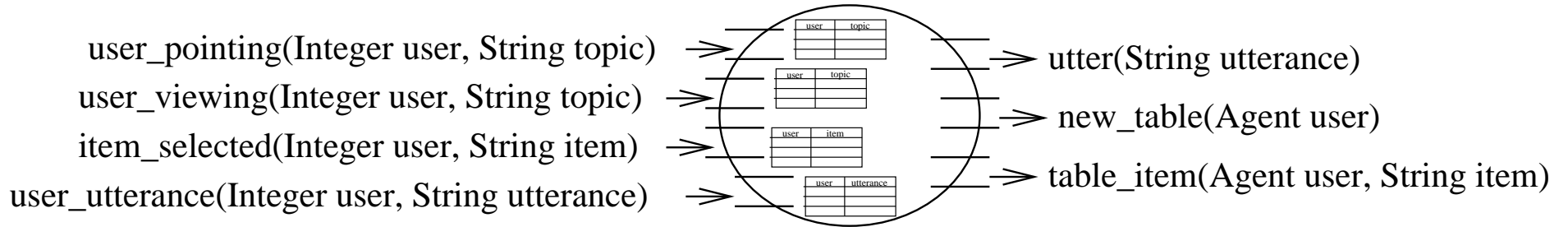
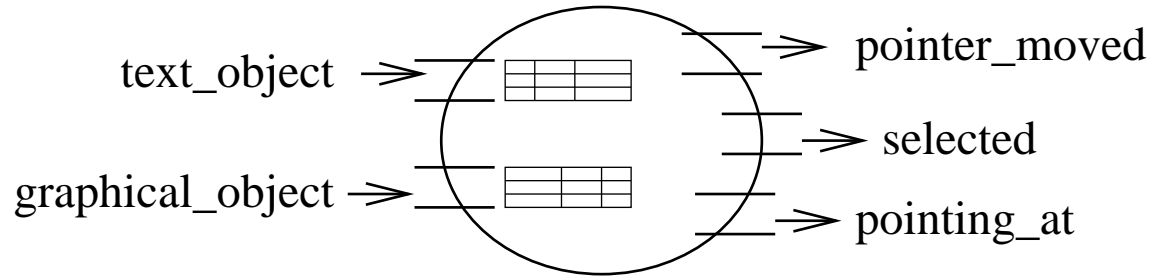
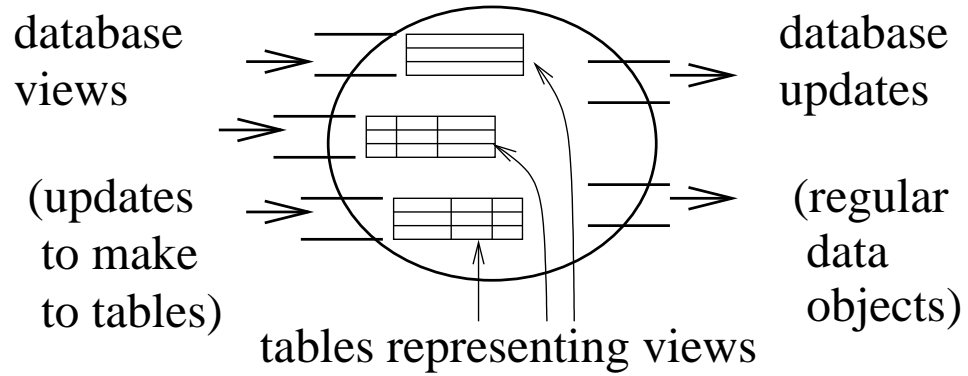
The glue scripts contain **translation** of inputs and outputs to the **shared database**.

Toolkit consists of **multiple languages**, each for its own purpose.

Which brings us to a short explanation of the toolkit itself.

VETk is a component-based system, which means the software is a collection of interaction modules, called "software components". The component is called "agent".

[read the sheet]



Here is an illustration of the VETk agent.

The VETk agent is defined by a number of input and output points. The arrival of input may trigger arbitrary behaviour and output. Special about this system is that input consists of updates to database tables, and output triggers the execution of a script enabling arbitrary processing of the output.

I gave two examples here: the top example is a "canvas" agent, the bottom example is a multimodal dialogue system.

The canvas takes as input a set of objects to display, here a set of text objects and a set of graphical objects. It outputs mouse events: coordinates, move_over object events and mouse clicks on objects.

The dialogue system has as its input a set of objects that the user is viewing, the user's utterance (text, in our case), and item selections in a table. The dialogue system outputs utterances, and it may create tables of answers.

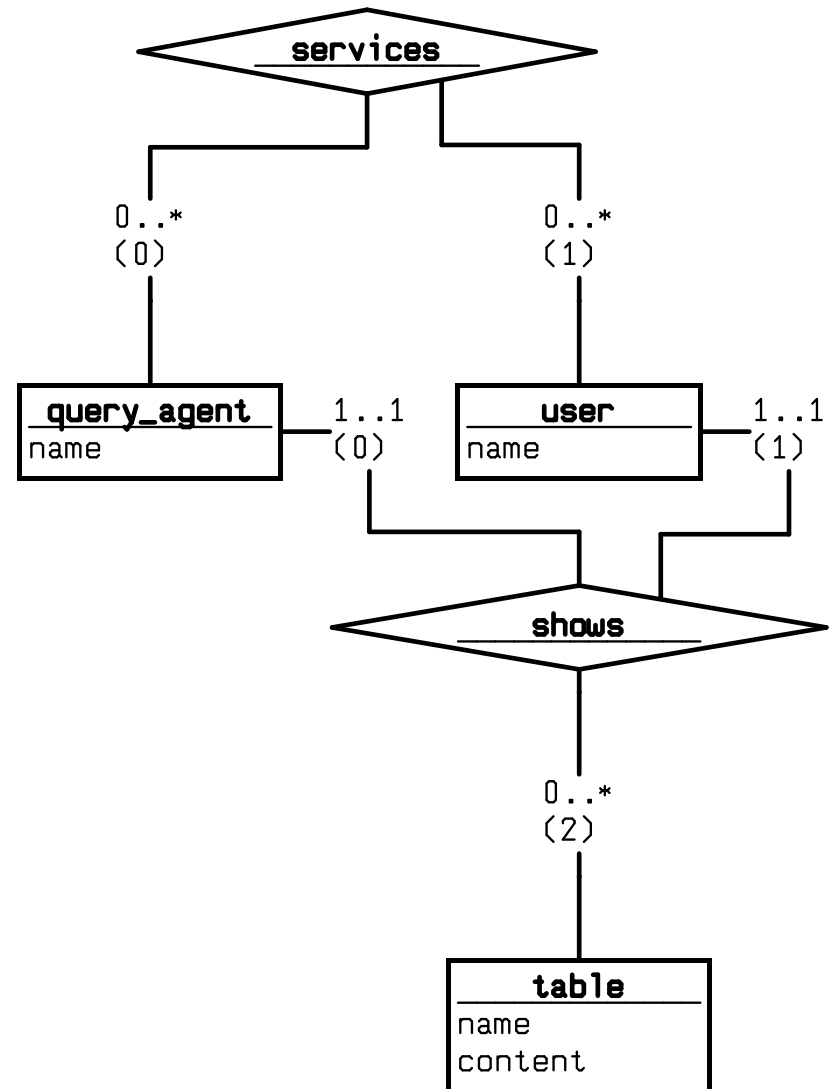
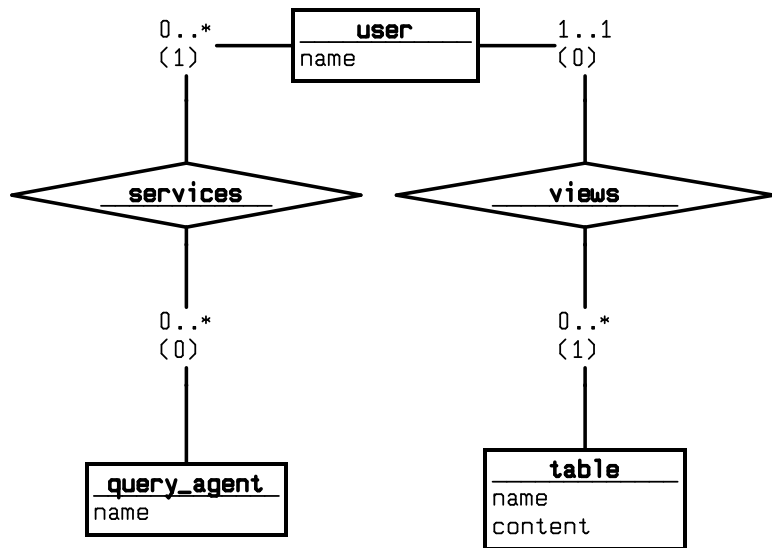
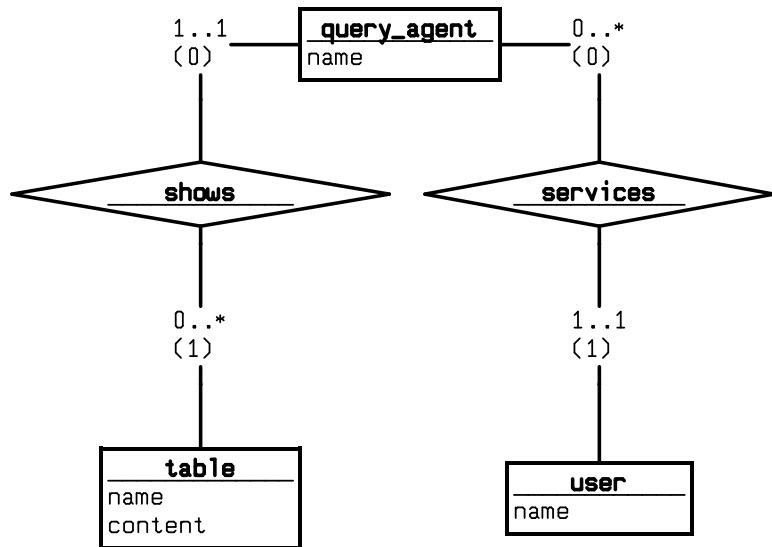
System property constraints		
ERD of user interface	VDC (VETk Data Constraints)	non-expert
State automata of UI objects	VDC	non-expert
logic constraints on any data in database	VDC	expert
Glue languages		
Visual layout	HTML (with embedded VETkScript)	non-expert
Main glue language	VETkScript	expert
Individual agent behaviour		
Agents' interface	VCL (VETk Component Language)	expert
Agent behaviour	Java language and environment	expert

The toolkit provide a number of languages for different purposes. All are integrated into an executable system, either being executable or being used to verify the correctness of the executable system.

We may divide the languages into three classes: **system property constraints**, declarative functional specifications which are used to verify the correctness of the system. The ERDs are in fact system property constrains. It is also possible to specify extra logic-type constraints.

glue languages, which are used to specify architecture and integration of agents within the architecture. The main language is a scripting language **VETkScript**, which specifies instances of agents within the system, and which may be embedded in **HTML** pages.

agent behaviour specifications, which specify the individual agents.



Here is a simple example that illustrates the assets of structure diagrams for UI specification. In these diagrams, the boxes are entities, and the rhomb shapes are the relationships.

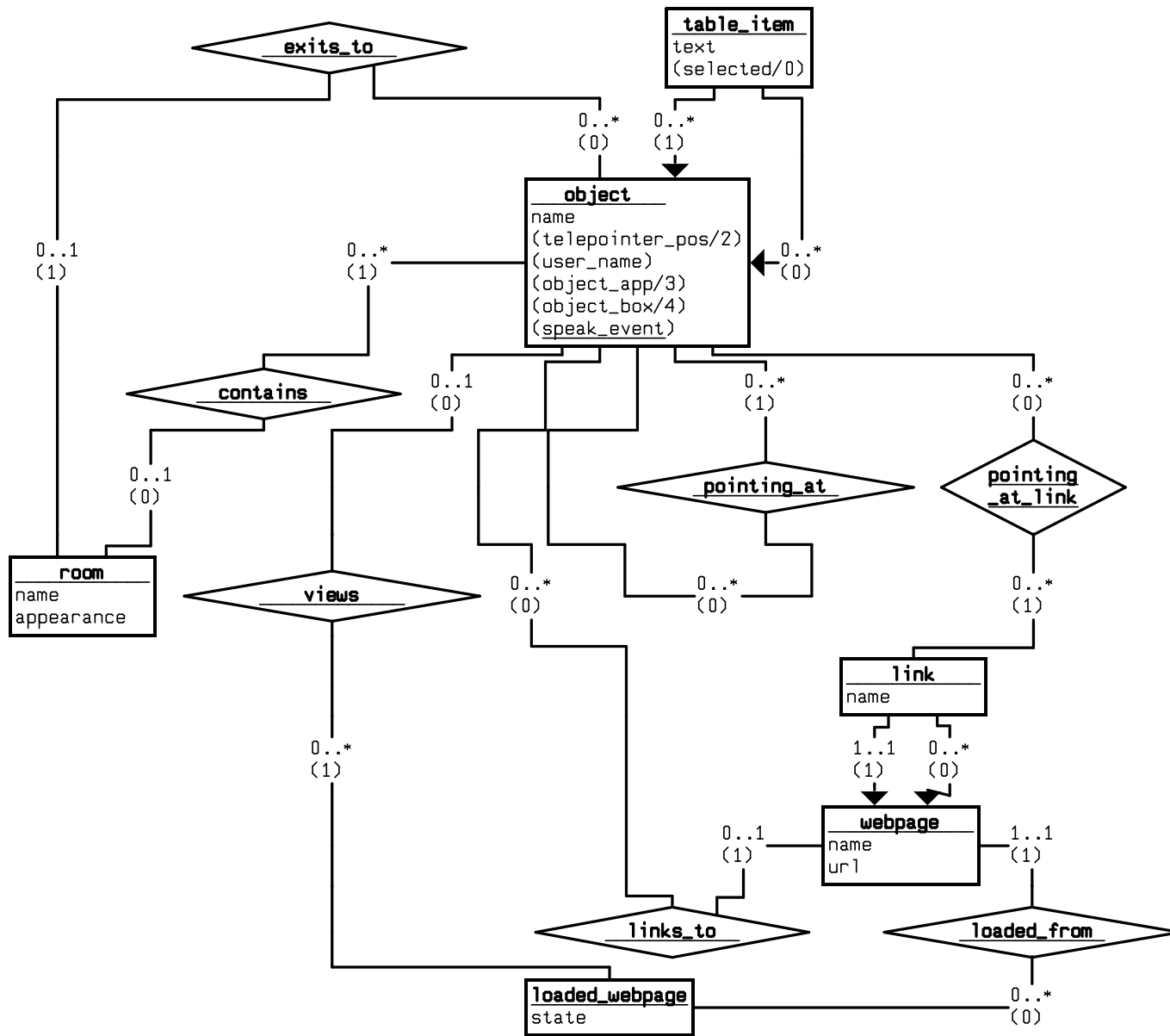
Consider a "search engine" type query system, in which we want to have one or more "intelligent" query agents and perhaps even some interaction between users of the engine.

Here we try to specify its possible structure, abstracting away from the representation, which may be as a regular search engine, or as a 3D virtual reality agent.

The question is: how does a search engine relate to users and to the search results being displayed? We have three alternatives here.

The topleft diagram shows that each agent services only one user, and that a user may be serviced by multiple agents. Each agent may show some search results to the user.

Alternatively, we may have global query agents servicing multiple users. This is the situation at the bottom left. However in this structure we cannot see which of the search results belong to which agent. To solve this problem, an additional relation coupling both agent and user to a search result is needed. This is the situation at the right.



This second example is a full-blown web-based VE. It combines web pages with a virtual world and a dialogue system.

The virtual world consists of *rooms containing objects*. Avatars are objects too. Some objects may be doors, *exiting* to other rooms. Meanwhile, each user may be viewing a number of webpages out of the set of known webpages.

This data structure makes it easy for an agent to track system dynamics, for example what web pages the users are viewing. This information is for example used by the dialogue system to enable certain linguistic references to be resolved.

Conclusions

- It is possible to build a complex system
- Specifications remain short

⇒ Successful as a proof of concept

Possible improvements:

- Error display, programming facilities, and language features
- Integration of 3D engine underway

We have only built a prototype version of the toolkit, we have put limited time in it.

Basically, we can say that with this version

it is possible to build a complex system, such as a multiuser virtual world with dialogue system,

and that the code and specifications remain concise. Simple distributed multiuser applications may be written using a single page of code, and we wrote more complex examples using only a few pages of code.

Hence we conclude that the approach is successful as a proof of concept, and may be worth developing further.

The current software could use various small improvements in for example error display and programming language usability, which is mostly straightforward programming work.

While the first version of the toolkit has no special support for 3D, a first prototype of a 3D engine with integration into the toolkit has been built.

For more information

see the VETk home page

<http://wwwhome.cs.utwente.nl/~schooten/vetk/>

You will find the software, and also the slides and notes of this presentation on this page.