From Proceedings of the Fifth International Conference on Information Visualisation. London. IEEE Computer Society Press, 2001, p. 584-588. ISBN 0-7695-1195-3.

The General Variables Concept: A Simple Step from Single- to Multi-user Environment

Michal Masa, Jiri Zara
Department of Computer Science and Engineering, Czech Technical University in Prague
{xmasam, zara}@fel.cvut.cz

Abstract

When implementing multi-user cooperation systems in networked virtual environments, synchronizing local representations of shared virtual world has to be considered. It involves distributing and storing the data that represent users' activity in the virtual world. In this paper we propose a concept of general variables, which formalizes the task of distributing, journaling and locking the data in the client-server model. To verify the employability of the concept, we have implemented a Java bundle comprising a server and a client-side library. Four multi-user applications are presented to demonstrate that with the help of the bundle, a single-user application can be turned into multi-user one with minimum effort.

1. Introduction

Last year we introduced a client-server system, which utilizes multi-user VR environment for educational purposes [1]. The system sets off one of the users as a *tutor*, the rest of them become his/her *auditors*. The main goal of the system was to enable auditors to watch the virtual world through the tutor's eyes. The system was implemented using VRML (Virtual Reality Modeling Language) [2,3] and Java programming language to ensure portability to greatest extent possible.

The next development step was to extend the system to meet multi-user cooperation purposes. In contrast to previous scenario, it is crucial that any changes performed in the virtual world by any user are persistent and visible to every user. Every connected user should see the world in the same state. These requirements involve distribution of world changes among the active clients and bringing later connected users up-to-date on the current state. For this purpose we enhanced the server to support a *general variable* distribution, journaling and locking.

Since the concept of general variables is very flexible, it allows a wide range of applications to utilise the server for their specific purposes. This led us to an idea of designing a Java library, which enables turning a single-user application into a multi-user one with minimum effort. The library in conjunction with the server provides a client application with basic capabilities required for implementation of a typical multi-user system: user administration, network communication, latecomer updates, etc.

In section 2, we explain the concept of general variables and related mechanisms of variables distribution, journaling and locking. Section 3 discusses the concept implementation. The results are presented in Section 4, where applications currently exploiting the proposed approach are listed and described. Section 5 concludes the work.

2. The concept of general variables

The primary task of a cooperative multi-user system is to ensure distribution of users' activities and world modifications. The system should also bring latecomers up-to-date on the current state. As far as a client-server model is considered, the server is responsible for distributing and storing the appropriate data representing user's interaction with the world. Since the structure and the meaning of the data are application dependent, we proposed the concept of general variables.

Table 1. List of supported commands

Command	Meaning
`setValue(value)`	Sets the variable to a particular value
`set1Value(index, value)`	Sets the index-th value
`insertValue(index, value)`	Inserts a value at the index-th position
`delete(index)`	Deletes the index-th value from the variable
`clear()`	Clears the variable

From Proceedings of the Fifth International Conference on Information Visualisation. London. IEEE Computer Society Press, 2001, p. 584-588. ISBN 0-7695-1195-3.

2.1. General variable structure and distribution

General Variable consists of a name for its unique identification and a list of *commands* performed on the variable. The variable can be further partitioned and treated as an array. A typical command sets the variable to an arbitrary *value*. The flexibility of the concept is based on the fact that the value can be compounded of any number of any primitive data types. It can be a simple value as well as a heterogeneous structure. When a user attempts to interact with the world, the client application creates adequate variable and adds a specific command containing a value representing the user's action. Supported commands are listed in Table 1. The variable is then sent to the server, which is responsible for broadcasting the variable to other clients. Finally, the receiving client should decode the meaning of the variable and replay the original action locally.

Every variable sent to the server has associated information that controls the distribution of the variable – whether it should be broadcast to all active clients (implicitly except the sender) or sent just to a particular one(s). If the consistency of the virtual world is critical, all changes to the world should be performed in the same order on all clients. Since the server performs ordering of variables prior to the broadcast, echoing the variable back to the sender can accomplish the task (echoing is determined when sending the variable by setting the `ECHO` flag). The sender must then postpone the realization of the changes until all variables preceding the echoed variable are processed. As this approach reduces interactivity rate due to longer response times, another technique is mutual exclusion on accessing the variables described in article 2.3.

The similarity between the commands proposed here and those listed in [2] is not accidental. Since we assume VRML representation of the virtual world in most cases, one of the possible uses of general variables is to imitate VRML fields. In VRML, user's interaction and changes to the world will always result in VRML event(s) generation. Those events can be easily encapsulated into `setValue` command. More complex operations on VRML multiple-valued fields will exploit the remaining commands or even their sequences.

However, this low-level approach is definitely not the only way of utilizing general variables. Thanks to their flexibility, they can represent properties of any abstract structure in the virtual world. The interpretation of the variable is up to the application, which has to adapt the concept to meet its specific needs.

2.2. General variable journaling

Latecomers to the system (clients connected later) should find the virtual world in its current state. However, when considering VRML again, client typically starts by loading original world, which is unaffected by any changes made to it later. To solve this problem, the server maintains a journal of variables. Implicitly, every variable distributed through the server is stored in the journal prior to further broadcasting. The process of storing involves the following steps.

First, the journal is searched for the existence of a variable. If it is not present yet, an empty variable with the same name is added to the journal. Then the list of commands contained in the processed variable is appended to the list contained in the journal variable. However, in some cases it is not necessary to cumulate all the commands performed on a variable, it is sufficient to store only the last command instead (providing it is a `setValue` or `set1Value` command). A variable representing the position of some virtual object is a good example – only the last position of the object is significant, prior changes can be safely forgotten. There are also situations, when the variable should not be stored in the journal at all. A variable holding a chat text could be such an example, unless we want to store the chat history. To differentiate among particular cases, a *storage flag* is attached to every command. Possible values of the flag and their meanings are listed in Table 2.

Special methods are being investigated to prevent the list of commands from having accumulated illimitably. The idea is that the list of commands can be modified (shorten), while it still preserves the original information. For example, when appending `clear()` command to the list, all previous commands can be removed, since they are not relevant any more.

2.3. Hierarchical locking of general variables

A user performing complex modifications of the virtual world may require exclusive access to affected part of the world to prevent other users from interfering with him/her. In the general variable concept, this is provided by a variable locking mechanism. A client can lock the whole variable or its specific index. Locks are exclusive,

Table 2. List of possible values of the Storage flag

Storage flag value (symbolic constant)	Meaning
`STORE_NEW`	Store the command by appending it to the end of the list
`STORE_LAST`	Store the command by overwriting last command in the list
`STORE_NOT`	Do not store the command

From Proceedings of the Fifth International Conference on Information Visualisation. London. IEEE Computer Society Press, 2001, p. 584-588. ISBN 0-7695-1195-3.

preventing clients from locking the variables that have already been locked by other clients. The client is responsible for unlocking the variable(s), after the user has finished the world modifications. It is up to the application to be aware of possible locks and to check the locks prior to enabling a user to modify the world. Since we assume the same application running on all clients, it should not be difficult to implement it.

The nature of a virtual world is essentially hierarchical. Virtual objects are usually compounded of smaller parts – another virtual objects. If a user attempts to modify an object exclusively, he/she might require preventing the others from modifying even sub-objects concurrently. And vice versa. For example a user changing overall size of a virtual room would interfere with another user trying to precisely position a table relatively to the room wall. In order the application would not have to check the locks of all sub-objects, we proposed a hierarchical locking based on variable names.

The principle of hierarchical locking is as follows. If a variable named `idA` is locked, then variable named `idB` cannot be locked as far as one of the following equations is satisfied for some `suffix`:

```
idA = idB+suffix        (1),
idB = idA+suffix        (2).
```

The `suffix` represents an arbitrary string (even empty one) and operator '+' denotes string concatenation.

To illustrate the use of hierarchical locking, consider a virtual world containing a room with a table and a chair. Suppose an application chooses variable names according to Table 3 to express hierarchical organization of the scene. Consider a situation in which the variable "`Room.Table`" is locked. Table 3 shows which variables can be locked next.

On the other hand, if the second user in the example above was just changing the color of the table, he/she would not interfere with the first user at all. The application should determine how and in which cases the hierarchical locking can be exploited and investigate appropriate variable names accordingly.

3. The concept implementation

To verify the efficiency of the general variable concept, we have implemented it using Java programming language. The implementation consists of two parts: the server and client side. The server side supplies the general variables distribution and journaling, updates of latecomers and user management. It is also capable of serving several different virtual worlds at the same time by separating general variables of each world.

The client side is designed as a reusable Java library that mediates between a host application and the server. The library provides the application with a set of functions for general variable creation, distribution and locking. On the other hand, the application may define a set of callback routines, which the library calls in response to the messages received from the server. Callbacks occur whenever a general variable has been received; variable locking operation has proceeded, either successfully or unsuccessfully; a user has connected to or disconnected from the system.

Figure 1 shows the most common architecture of the client. The host application is implemented as a Java applet communicating through EAI [3] with VRML browser, both running in a WWW browser. The dashed line between the library and EAI block denotes an extra library features, which are a limited implementation of multi-user extensions to VRML as proposed in [4] (namely `NetworkNode`) and a support of trace mode required by DILEWA. Since these features rely on VRML scene representation, they can be deactivated on request.

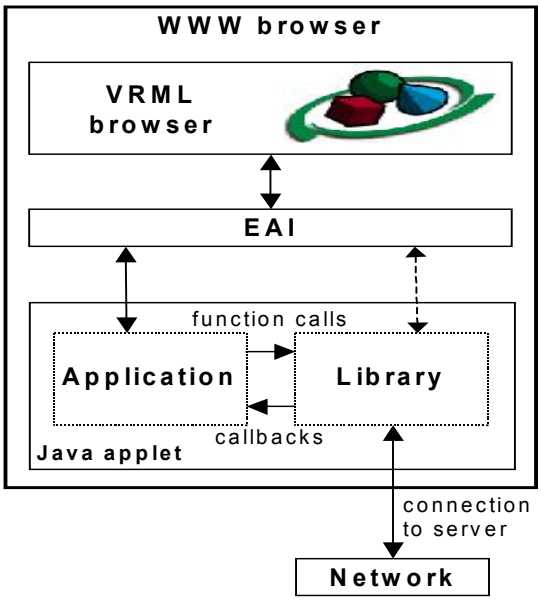

Figure 1. A common client architecture

Table 3. An example of general variable names expressing hierarchical structure of the scene

Variable name	Can be locked?	Why not? (Satisfied equation(s), `suffix`)
`Room`	No	(1), "`.Table`"
`Room.Table`	No	(1), (2), "" – empty
`Room.Chair`	Yes	
`Room.Table.Leg`	No	(2), "`.Leg`"

From Proceedings of the Fifth International Conference on Information Visualisation. London. IEEE Computer Society Press, 2001, p. 584-588. ISBN 0-7695-1195-3.

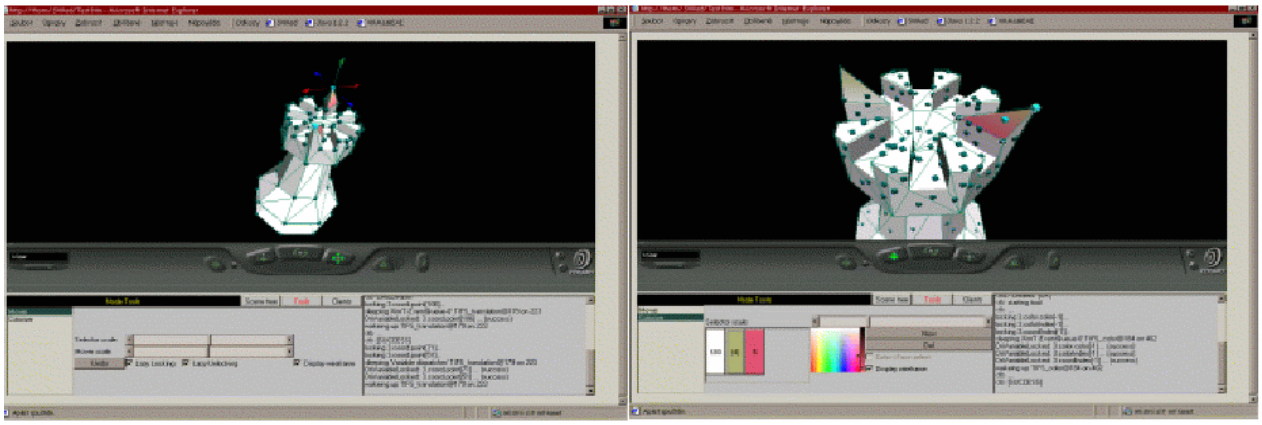

Figure 2. A concurrent editing session, where two users are editing the same model. The Transform tool has been attached to the rook model, thus enabling the users to change the positions of vertices by dragging.

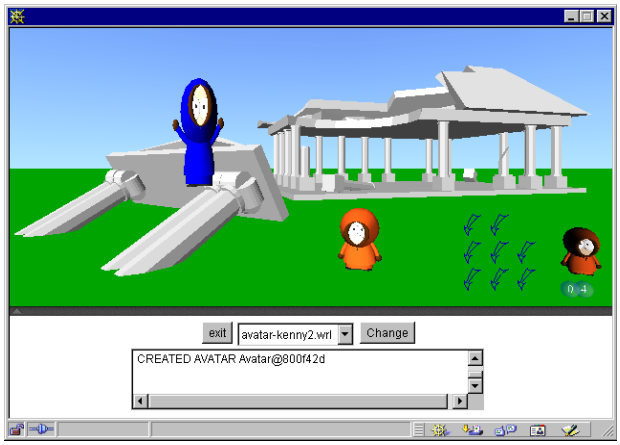

Figure 3. Avatars and chat window in Virtual Agora

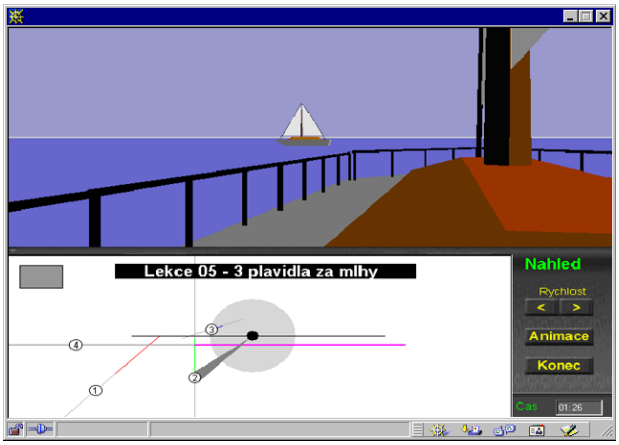

Figure 4. User interface of yacht training application

4. Sample applications

At present, our research group is developing four multi-user projects based on the library.

The first of them is a successor of the DILEWA project, which preserves original tutor/auditors idea. The DILEWA functionality is now fully supported by the library, including the trace mode and net-routes, which were implemented using `NetworkNode` mentioned above.

The next project deals with cooperative editing of VRML scene. Users can modify scene objects by attaching special geometries (*tools*) to them (Figure 2). Manipulating the controls of the tool results in changes of the object properties. There are several different tools related to particular object properties: the Material tool and the Transform tool are the basic ones. The tools are attached dynamically and no preprocessing of VRML file describing the edited scene is required. The application employs general variables for representing object properties that correspond directly to VRML fields in most cases. The library assures that changes of the scene are visible to every user, even to the latecomers. The variable locking mechanism can be exploited to prevent users from changing the same property of the same object simultaneously.

The third project has got a working name Virtual Agora. It involves existence of avatars and chat facility (in a similar manner to the Blaxxun system [5]). These are not natively supported by the library but can be easily implemented by the use of general variables (Figure 3).

The last project is an experimental system for teaching and testing yacht captains (Figure 4). The project proposes a scenario, where an instructor can control, guide and examine his/her student(s) in a shared virtual world. The implementation comprises of two distinct client applications to differentiate between specific needs of the instructor and the student. While the instructor

generates tasks and test situations, the student has to solve them. Since both are sharing the same world, the instructor can easily control the student or help him/her when the student is experiencing difficulties.

5. Conclusion

In this paper, we have presented a formalized approach to the problem of distributing, storing and locking the data in multi-user networked virtual environment based on the client-server model. The concept has been implemented as a Java bundle, consisting of a server and a client side library. Four multi-user applications, each aimed at different purpose, have used the library and proved the efficiency of the concept.

6. Acknowledgment

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under research program No. 1898/2001 – Cooperation in the virtual reality.

References

[1] M. Máša, J. Žára, "DILEWA: The Distributed Learning Environment Without Avatars", *Proceedings of 2000 IEEE International Conference on Information Visualisation*, 2000, pp. 563-567.

[2] The Virtual Reality Modeling Language International Standard ISO/IEC 14772-1:1997
http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[3] The Virtual Reality Modeling Language External Authoring Interface Committee Draft ISO/IEC 14772-2
http://www.vrml.org/WorkingGroups/vrml-eai/Specification/

[4] Living Worlds,
http://www.vrml.org/WorkingGroups/living-worlds/

[5] Blaxxun Interactive,
http://www.blaxxun.com/

[6] G. Reitmayr, "Behind the scenes of a VRML multi-user architecture", VRML99 Courses, 1999
http://www.c-lab.de/vrml99/
http://www.geometrek.com/

[7] R. Galli, Y. Luo, "Mu3D: A Causal Consistency Protocol for a Collaborative VRML Editor", *Proceedings Web3D – VRML 2000*, 2000, pp. 53-62.

[8] D. Margery, B. Arnaldi, and N. Plouzeau, "A General Framework for Cooperative Manipulation in Virtual Environments", *Proceedings of the Eurographics Workshop in Vienna*, Austria, May 31-June 1, 1999, pp. 169-178.