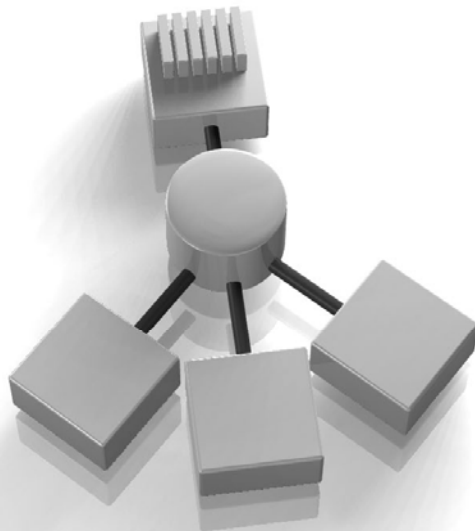


**SIXTH FRAMEWORK PROGRAMME
PRIORITY IST-2002-2.3.1.8
Networked Audiovisual Systems**



**Uni-verse project
Deliverable D 6.2
Alternative Verse Server
July 30, 2006**



**STREP project
Project acronym: Uni-Verse
Project full title: A Distributed Interactive Audio-Visual Virtual Reality System
Proposal/Contract no.: 002228
Distribution: Public**

1	INTRODUCTION	3
2	SYSTEM DESIGN	3
3	INTERNAL COMMUNICATION	5
4	NETWORK CONNECTION	5
5	STORAGE	6
6	USER AND PROJECT ADMINISTRATION	6
6.1	USER ADMINISTRATION.....	6
6.2	PROJECT ADMINISTRATION.....	7
7	RESULTS	8
8	REFERENCES	9

1 Introduction

A major task in work package 6 was the design and implementation of an Alternative Verse Server. The Verse server is the central data connection and storage instance of the Uni-Verse system. In the first version of the specification it was planned to integrate the reduction module into the server and make the server capable of connecting to other servers to enable proxy functionality. After recommendations of the end users in the Uni-Verse project and a reallocation of financial resources at Fraunhofer it was approved by the PCC that the server should contain a user administration and project handling system instead of the reduction and the proxy functionality. Therefore the implementation plan and the specification were improved and accepted as D2.4. Following the new specification the server was redesigned to meet these requirements. The server was developed using C++ using QT [1] and STLport [2] libraries.

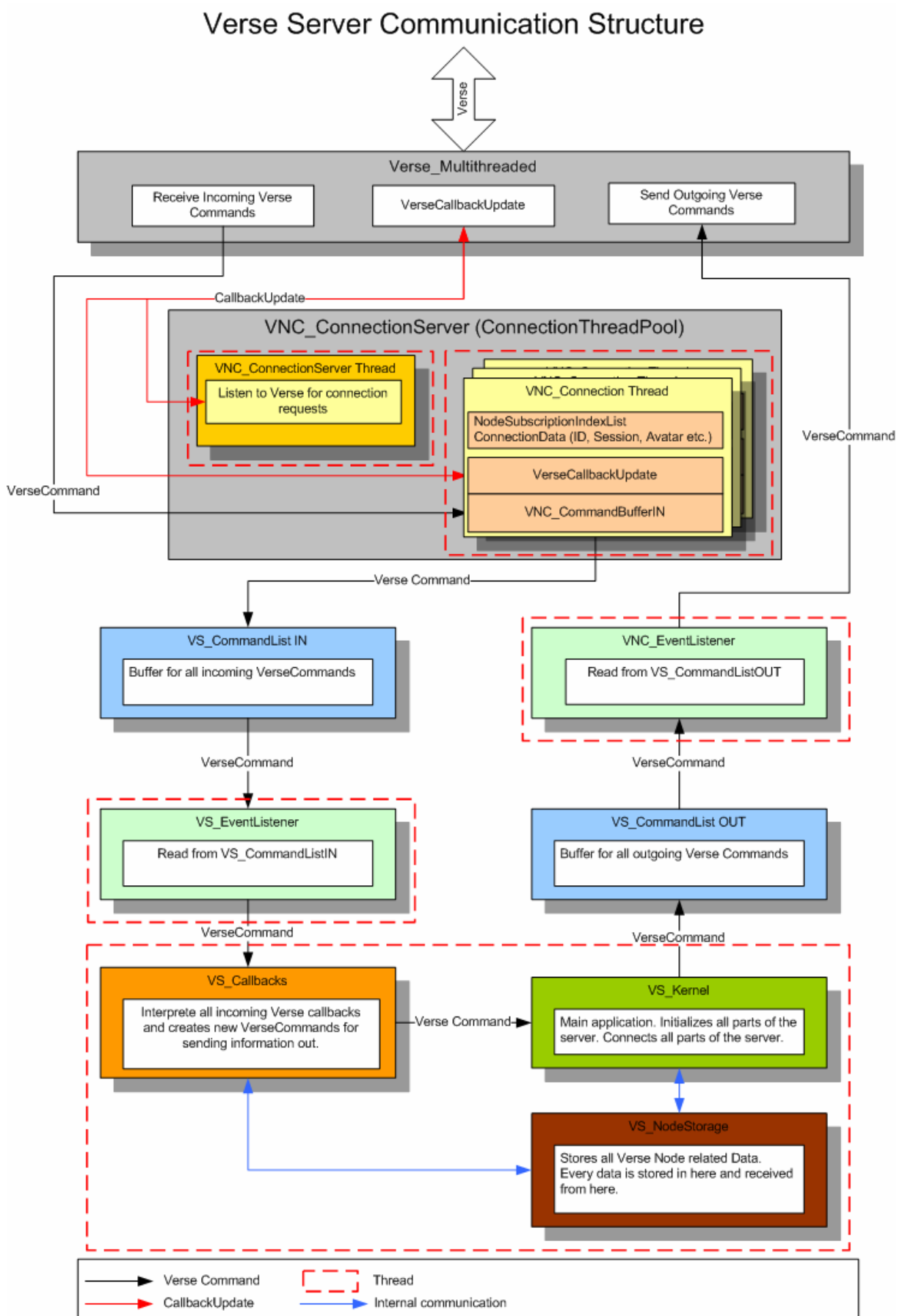
2 System design

Modern server technology predominantly uses asynchronous processing for external and internal communication. Therefore, the server was designed to make the read and write processes on the server work independently. Multiple threads organize the reading of the Verse network queue, while one thread is responsible for sending out Verse commands. The scheme in Figure 1 shows the complete server design. The server is divided into several libraries where each library contains a functional part of the server. This modular design allows the developer to exchange functionality in the future for improvements, or to use the server for other purposes.

The Verse_Multithreaded wrapper allows the server to access Verse using multiple threads. The connection server listens to new connection attempts via the Verse protocol and handles the connections. Every client connection runs internally as a thread that asks the Verse interface for updates. Thus, it is possible to asynchronously read and write Verse data.

The data of the incoming Verse callbacks is immediately stored in internal Verse commands. From that point the complete server communication works with these internal Verse commands. Each Verse command is first sent from the network layer to the current connection's buffer. There, the command is test against the current user rights. If approved the command is sent into the pipeline for interpretation and storage. The interpretation of the Verse commands is done in VS_Callbacks (see Figure 1).

The storage is organized node-based. Every node class contains all the data that a specific node can store. The callback interpreters for each command decide into which nodes the information has to be stored and to which connections the incoming commands have to respond. If a callback interpreter decides to respond by sending commands, they are stored in a command list for outgoing commands. This command list sends an event "list is not empty" to the send thread. The send thread reads out the list with the Verse commands and sends them to the connections described in the subscriber list of the command.



3 Internal communication

The internal communication uses commands and events. Arriving commands from the Verse protocol are transformed into internal Verse commands that are sent through the server's pipeline.

Verse Command

As described the complete internal communication works with Verse commands. A Verse command contains general information (which connection sent the command, what type of command is it, etc.) and specific information (the Verse data). Using these commands the server constantly knows who sent which command and to whom it has to be sent out. All specific Verse commands extend the basic internal Verse command.

Pipeline

The command pipeline consists of two main buffers connecting senders and readers on both sides. The first buffer is the command list for incoming Verse commands. Each connection writes its commands into that buffer. An event listener on the storage side reads out this command list and distributes the commands to the concurrent command interpreters. The communication from the storage to the Verse send thread uses the outgoing command list as a buffer. The send thread contains another event listener that reads out this command list.

The event listeners on both sides always read out the complete command list content instead of only single commands. This strategy lowers communication effort and speeds up the overall processing. On the outgoing side, this strategy has the effect that a larger group of commands can be sent out to the Verse protocol from the same thread at the same time.

Multiple threads

The usage of multiple threads – at least one for each connection and the send thread – requires the internal server modules to be designed thread-safe. The communication between the different modules is done using the described internal Verse commands, since using these commands allows a message-based communication that is asynchronous and thread-safe.

4 Network connection

The server connects to the Verse R5 protocol and accepts and interprets all commands coming from the protocol. The network module of the Verse server contains a library that acts as wrapper for the Verse interface, a connection server that stores and manages the connection-related data and structures and the connection threads for each connection. The VerseMultithreaded wrapper grants multiple threads parallel access to the Verse protocol. Here, the Verse callbacks are transformed into internal Verse commands and assigned directly to the connections. The connection itself will then test the command against the current user rights and send the command to the command pipeline. The Verse callback update is currently wrapped using a mutex.

5 Storage

The storage makes use of STL structures like vector or hash_map. These are proven in practice, safe and fast. The storage contains the possibility to save data into different projects. For each project the storage generates a separate complete Verse storage structure in the server. The projects can be accessed by e.g. using tag information and method calls.

Since the Verse data structure is based on nodes, all data is stored node-oriented. Each node contains all data related to it regarding to the Verse specification. The complete specified node structure is defined in the server storage. Every Verse parameter and structure can be saved there for each node type. Each storage access function provides an exit code that determines if the storage was successful or not.

6 User and project administration

Verse originally does not support commands that enable the creation or the changing of projects. Therefore, an administration tool for users and projects was implemented. This tool implements an additional TCP-based simple network connection outside Verse to connect to the Alternative Verse Server. Using this connection all project and user related commands are sent to the server. The AdminInterface of the server interprets these commands and activates the concurrent actions on the server. Because user and project management is not a Verse data-oriented transmission this additional network connection saves bandwidth for Verse itself and allows the server to be administrated remotely.

6.1 User administration

The user administration ensures that only authorized users can access the data on the Verse server. A MySQL-based administration tool was developed to register and modify users and their rights on the server. The data from the administration tool is saved into a MySQL database. The server reads out the information of the database and reacts accordingly.

Due to the use of a pipeline and message queue, each command can be parsed to determine its rights. For example, if the user has only the right to read, all commands that contain create or change operations can be filtered by the command parser in the command pipeline. The user administration is designed as a prototype. Improvements can easily be done by updating the parser for the command pipeline. This could be useful to e.g. prevent a user from sending a special Verse command or to allow another user to use only a subset of the Verse commands. To integrate this functionality, the administration tool must be improved to give the users more detailed rights than just read or write properties.

Figure 2 shows the developed interface for the user and project administration of the Alternative Verse Server.

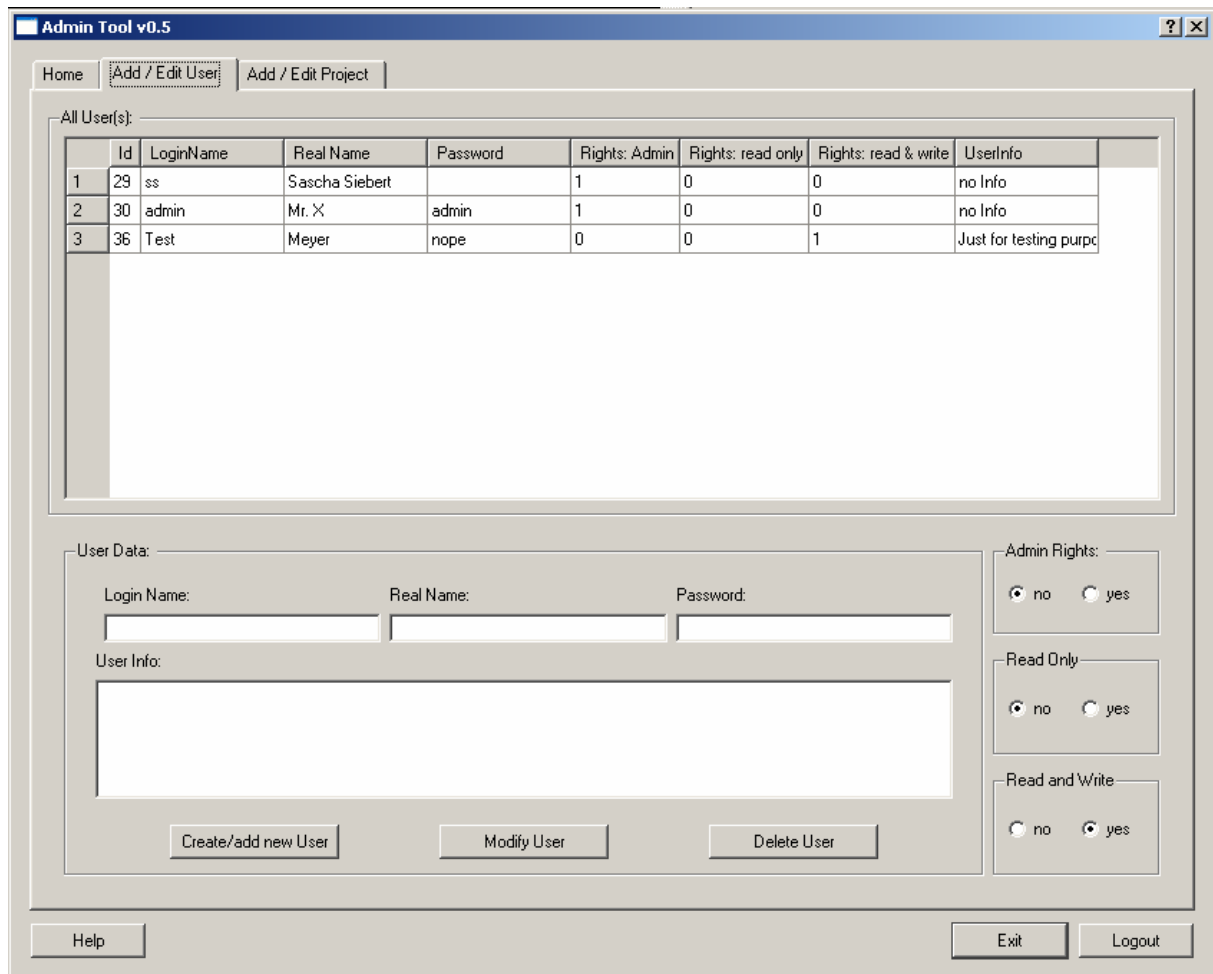


Figure 2: MySQL based User and project administration tool

6.2 Project administration

The standard Verse server only knows one large storage facility. The storage management inside the Alternative Verse Server organizes all data project-oriented. This means that every connection (i.e., every user) is assigned to a project. Each project contains exactly one storage instance for the Verse structures. The user sends all data and change attempts to this project. Multiple users can be assigned to the same project, but one user can only be assigned to one project at the same time. When starting the server a default project is defined and created. Every user is connected to the default project first. If there are more projects with data stored on the server, the user gets tags containing information on all currently active projects on the server.

As described, Verse itself does not provide commands for project-oriented communication. Therefore a set of tags and methods had to be defined on the Verse server. In Verse, every connection or user is represented by an avatar node of type object. The server creates a tag group called "Projects" in every new connected object avatar node that contains tags with the project information of the server. Each tag in this tag group defines one project. Additionally a method was defined that enables users to change projects on the server. The tags and the method are stored

inside the object avatar node and the information is sent to the client. It is now up to the client to decide whether to interpret this data or not. This mechanism assures the compatibility to Verse clients that have no interest or knowledge of the possibilities to store data in different projects. These clients will store all their data in the default project.

7 Results

A functional prototype was realized for the described server design using C++, QT, and STLport. The prototype uses a thread-safe wrapper for the Verse protocol. Behind that wrapper, multiple threads can access the Verse protocol in parallel. Every connection of the Alternative Verse Server is running in an own thread to improve the performance of the server increasing the performance on computers capable of parallel computing. A flexible design concept using modules was implemented including a user interface for the server. A user and project administration tool with connection to the Alternative Verse Server was implemented.

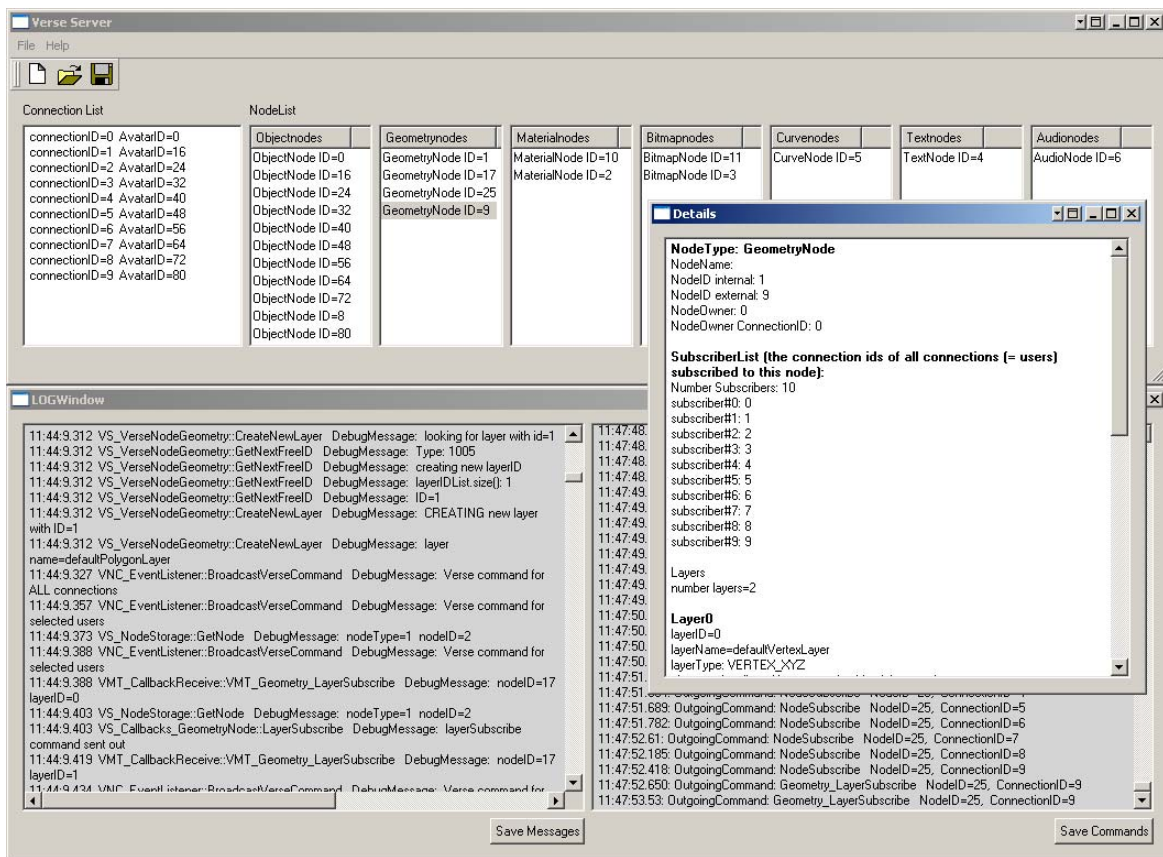


Figure 3: Alternative Verse Server's user interface

Potential future improvements

As an outlook for future work, the server performance can be improved by optimizing the communication at the Verse interface. Furthermore, the communication in the command pipeline can be optimized to gain higher performance. This should be done after long-term tests of the current prototype in an environment with a large number of concurrent users.

8 References

[1] QT by Trolltech: www.trolltech.com

[2] STLport by SGI: <http://www.sgi.com/tech/stl>