Thin Client Access to a Visualization Environment*

Ioannis Fudos and Ioannis Kyriazis

Department of Computer Science, University of Ioannina,
GR45110 Ioannina, Greece, {fudos, kyriazis}@cs.uoi.gr

Abstract. In this paper we present a thin client system that provides Internet access to a modular visualization environment. The communication protocol for this system is designed so as to minimize the data exchanged among the server and the clients. An XML format is introduced for communicating visualization related information. Users of the system may collaborate to perform complex visualization operations and share files. The results of each operation are cached so that they may be used by the same user in a later session or by other collaborating users. Experimental performance results demonstrate the efficiency of our system when compared with commercial general purpose solutions.

1 Introduction

Graphics visualization is a demanding computational task. To process and render a complex scene of 3D objects, computationally powerful platforms are required. We have designed a client-server system that provides remote access to a visualization environment through Internet via a web browser. The system is suitable for groups of collaborating users that need to perform complex visualization related computations without having physical access to the same machine. A number of research and commercial systems have dealt with similar problems in the area of biology for MacroMolecular Modeling [6], meteorological and oceanographical purposes [9,1,2], and for general purpose world wide web applications[4,5].

In this paper we present the following technical contributions:

- a reduced XML-based communication protocol for exchanging data visualization information.
- a file caching scheme for intermediate results, which increases the performance of the system and allows user collaboration.
- real time experiments that demonstrate the efficiency of our system over commercial solutions.

* Part of this work was funded by a Greek Ministry of Education EPEAEK-HERACLETUS Grant. We would like to thank Prof. Vaclav Skala and his group for making the command line version of MVE available for this project. Also, we would like to thank Prof. Leonidas Palios for useful suggestions on early stages in the design of this system.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 258–263, 2004.

As a concrete example of the above we have implemented a thin client access to MVE, a modular visualization environment [11].

The rest of this paper is organized as follows: Sect. 2 presents a short overview of the system and describes the data exchange protocol. Section 3 presents the caching scheme which is used to increase efficiency and allow user collaboration. Section 4 presents performance results.

2 Overview of the System

The system consists of three parts: The environment, which is responsible for all the computations performed during a session, the thin client, a light weight front end to the environment that allows users to access the environment remotely, and the server, which accepts requests from clients and passes them as arguments to the environment.

Our system uses the Modular Visualization Environment [11,10] to perform computations. Its operation is based on a set of independent modules, which are responsible for loading, modifying and rendering 3D graphics objects. Several modules connected to each other can form a scheme, which can be saved or executed. Each module is designed according to a common interface, regardless of the function of this module. Each module has a set of inputs, which are objects on which operations will be performed, a set of outputs which are the resulting objects, and some parameters that define its behavior during the execution of the operations. Thus modules can be represented by a data structure that describes its inputs, outputs, and parameters. When a user has created a scheme, the environment may store or execute this scheme. Execution of the scheme may be performed on a single machine or on a distributed platform.

We have used XML as many standards for interoperable graphics and visualization data are being developed in XML [8]. Also, there are portable parsers for XML that allow for porting the server part of our system easily to any platform. The client part is a plain java applet and is thus portable to any platform.

The server communicates with MVE by means of exchanging XML files. When a new client connects to the server, the server will request from MVE the list of available modules. As a response MVE will produce an XML file called modules.xml, where all currently available modules are described. An example of an XML file that contains the list of modules is shown in Fig. 1 (left). When a client wishes to execute a scheme, the server will provide a file to the visualization environment with an XML description of the scheme (scheme.xml). An example of a simple scheme described in an XML file is shown in Fig. 1 (right). At startup, the client establishes a connection to the server and requests the currently available list of modules. After receiving the modules list, the user can develop a scheme. When ready, the client sends the scheme to the server for execution. The scheme is executed, and the results are sent to the client for visualization. Besides executing the scheme, the client may request to save the scheme on the server's shared folders, or to load a scheme from the server. During a session, a client has to communicate with the server only to post a request for

```
<?xml version="1.0" encoding="iso-8859-1"
                    standalone="no"?>
<!DOCTYPE MODULES SYSTEM "modules.dtd">
<MODULES TIME="Tue Jun 26 18:48:54.471 2001">
   <MODULE NAME="Decimation" TYPE="2"
           DLL_NAME="Modules\ Decim_module.dll"
           DESCR="Triangle Mesh Decimation Module">
      <PARAMETERS>
         <PARAM NAME="Reduction" TYPE="NUM"
                DESCR="Percent of reduction"
                MANDATORY="FALSE"/>
      </PARAMETERS>
      <INPUT TYPE="2" DESCR=""/>
      <OUTPUT TYPE="2" DESCR="Output : Reduced list of
              vertices, reduced list of triangles"/>
   </MODULE>
   <MODULE NAME="TriangleSaver" TYPE="3"
           DLL_NAME="Modules\ Triangle_Modules.dll"
           DESCR="Save Triangle Data">
      <PARAMETERS>
         <PARAM NAME="FileName" TYPE="STR"
                DESCR="STL File to save triangles"
                MANDATORY="TRUE"/>
         <PARAM NAME="Binary" TYPE="BOOL"
                DESCR="Binary file ?"
                MANDATORY="FALSE"/>
      </PARAMETERS>
      <INPUT TYPE="2" DESCR="Triangles to Save"/>
   </MODULE>
   <MODULE NAME="IsoExtractor" TYPE="2"
           DLL_NAME="Modules\ Volume_Modules.dll"
           DESCR="Compute iso-surface from Volumetric
                  data (IsoExtractor)">
      <PARAMETERS>
        Not available
      </PARAMETERS>
      <INPUT TYPE="5" DESCR="Volumetric Data Input"/>
      <OUTPUT TYPE="2" DESCR="Triangle Data Output"/>
   </MODULE>
</MODULES>
```

```
<?xml version="1.0" encoding="iso-8859-1"
                    standalone="no"?>
<!DOCTYPE SCHEME SYSTEM "scheme.dtd">
<SCHEME>
   <OPTIONS PARALLEL="FALSE"/>
   <MODULES>
      <MODULE ID="TriLoad1"
              MOD_TYPE="TriangleLoader"
              PARAMETERS="Cow.tri"/>
      <MODULE ID="Dec1" MOD_TYPE="Decimation"
              PARAMETERS="80"/>
      <MODULE ID="TriSave1"
              MOD_TYPE="TriangleSaver"
              PARAMETERS="Cow2.stl;FALSE"/>
      <MODULE ID="TriSave2"
              MOD_TYPE="TriangleSaver"
              PARAMETERS="Cow3.stl;FALSE"/>
   </MODULES>
   <CONNECTIONS>
      <CONNECTION FROM="TriLoad1" OUTPUT="0"
                  TO="Dec1" INPUT="0"/>
      <CONNECTION FROM="TriLoad1" OUTPUT="0"
                  TO="TriSave2" INPUT="0"/>
      <CONNECTION FROM="Dec1" OUTPUT="0"
                  TO="TriSave1" INPUT="0"/>
   </CONNECTIONS>
</SCHEME>
```

Fig. 1. (left) The modules.xml file; (right) the scheme.xml file

executing, saving or loading a scheme. No communication with the server is required while developing a scheme. As shown in Fig. 2 (left), the information communicated among the server and the clients is minimized, as the messages exchanged are few and short in length. This makes our system appropriate for clients running on machines with slow network connections.

The server is the part of the system that connects the clients to the MVE, and provides them with appropriate information whenever requested, whether this involves calling the environment or not. As the server is multithreaded it can serve many clients at the same time. A client may request from the server to send the list of currently available modules, to execute a scheme and render the results, to save a scheme, or to load a previously saved scheme. The server also includes a login manager, which is responsible for user authentication and identification, and handles the ownership and permissions of the schemes. This way, a client may share a scheme with other users or groups, and set access rights for its schemes. Besides the login manager, the server includes a cache manager, which is responsible for caching the results of an execution, hashing the schemes to organize the cached results, and searching the cached results to retrieve a previously executed scheme. The client is designed as lightweight as possible, and is able to execute on any java-enabled web-browser. No computations are performed on the client machine, only some basic interaction with the server is carried out. The execution of an operation is performed by the server. The client performs only the visualization of the results. For user's convenience, the GUI of the client is similar to the MVE. Fig. 2 shows messages exchanged during a session, and a snapshot of the Internet client.

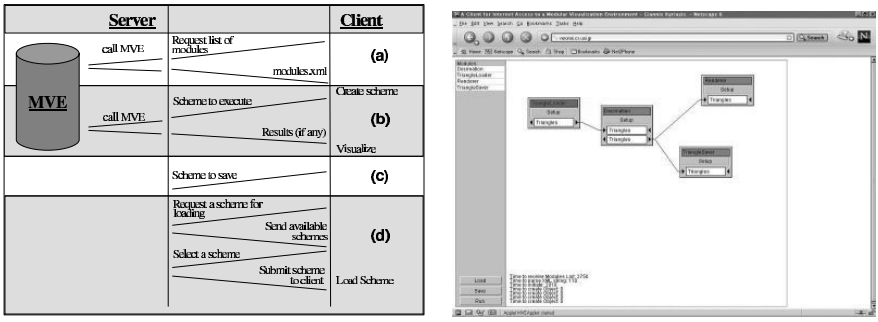

Fig. 2. (left) The messages exchanged during a session: (a) request the list of Modules, (b) execute a Scheme, (c) save a Scheme and (d) load a Scheme; (right) the Internet Client

3 File Sharing, Caching, and User Collaboration

When a client stores or loads a scheme, this scheme is actually stored on the server's site. These folders may contain other users' schemes. This way, users may collaborate by sharing their schemes. They also may form groups to share their schemes only with the members of the same group. A login process is used to identify and authenticate the client at startup. Then, when a scheme is stored, it has an owner. Owners may choose to share their scheme with members of their group, or to make it available to everybody. Like in a unix file system (UFS), the files have an owner and a group, and the owner sets the mode of the file for himself/herself, his/her group and others (read, write, execute for each such category).

To reduce the workload of the server further, we cache results of a scheme execution, so if a scheme has already been executed in a previous session, even from a different user, it will not have to be executed again. Caching only the final result of an execution would be useful only if the scheme for execution matches the previously executed scheme exactly. Even if one parameter had a different value, the cached results would be useless, as they would produce different results. This is why we cache the intermediate results as well, so that even partially matching schemes may use these results. Since the result of a module execution is the same when the input and the parameters are identical, even if the rest of the scheme differs we cache the result of each module seperately. To locate a cached result, we use a hash table. The hash function [7,12] uses the {module, input, parameter} set as input, and returns an entry in the hash table for the output file. Figure 3 shows the structure of the hash function and the hash table.

The module ID, along with its parameters and inputs are hashed as a whole, and the hash output is stored along with the output of the execution. If there is more than one output, each of them is stored separately, as they may be used as different inputs for some other module. To distinguish among the different

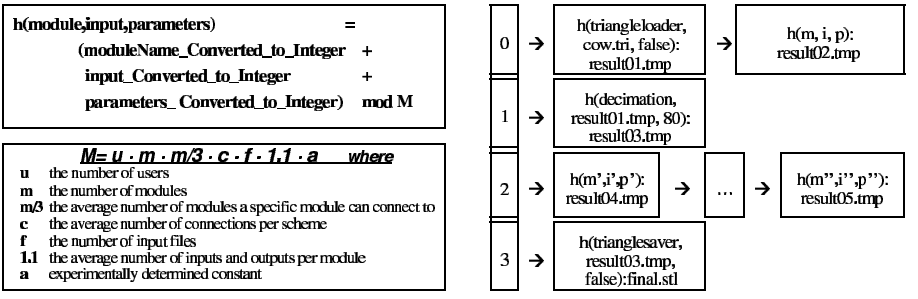

Fig. 3. Details on the hash function and the hash table

outputs of the same hash bucket, we store the {module, input, parameter} set as well. Since the input of each module was the output result of another module execution, there is a result file for this input already, so we can use this file instead of the input. The length of the hash table depends on the number of users, the number of different modules, the average number of connections per scheme, and the number of input files. If the table becomes very large, we remove the least used entries. If M is the length of the hash table, we should not allow more than 3M entries to be cached, as this increases the time to search a cached result. We have determined experimentally an efficient size which is illustrated in Fig. 3. In order to locate the cached results in the database, the server must first hash the clients scheme, to find the hash entries that may contain the cached results. The modules that participate in a scheme are hashed from the first to the last, and a hash entry is returned for each output. It is the server who searches for cached results, as it is the one that has the necessary information available. The client just sends the scheme to the server. The search is done backwards so that we can find a matching result as soon as possible. If we find a matching output, we use it as input to execute the rest of the scheme. The new results of the execution are cached as well.

4 Performance Evaluation

We tested our prototype system and evaluated its performance under various client, server, and network configurations.

In the first experiment, we compared the response time of our client-server system with a popular commercial tool that provides access to the desktop of a remote platform [3]. We measured the time it takes for the environment to start, and the times to load, save and execute a specific scheme. As shown in Fig. 4 (left), the response time for our client is relatively small, compared to the commercial tool. Our client performs well even over slow network connections, as the messages exchanged between the client and the server are few and short.

In the second experiment, we measured the performance of our client in various configurations concerning the state of the server and the sites where the

	56 kbps modem		100 Mbps LAN	
	Commercial	Our Client	Commercial	Our Client
Time to start	13 sec	2130 ms	2 sec	1650 ms
Time to execute	6 sec	975 ms	4 sec	845 ms
Save Scheme	2 sec	10 ms	About 1 sec	10 ms
Load Scheme	2 sec	280 ms	About 1 sec	130 ms

	LAN Client	Dialup Client		
Time to:	Normal	Normal	Network Traffic	Workload on server
receive Modules List	1750	2210	4030	6373
parse XML string	60	80	100	100
Startup	1910	2932	4216	8201
create module	0	0	0	0
save scheme	0	0	50	0
exec scheme	270	350	585	1010
receive file's list	0	130	370	3405
load large scheme	60	160	380	780

Fig. 4. Results of the experiments

server and the client run, such as increased network traffic, low system resources, and many clients connected on the server. The response times measured are the times to save, load, and execute a scheme, as well as to receive the list of available modules, and to startup the environment. As shown in Fig. 4 (right), the time to receive the list of modules is the main reason of delaying the initiation of the client. The time to receive the modules is relatively long, because it takes the server considerable time to generate the list.

References

1. Ferret, data visualization and analysis. http://ferret.wrc.noaa.gov/Ferret/.
2. Geovista center, collaborative visualization. http://www.geovista.psu.edu/research/ collaborativevisualization/.
3. Symantec pcanywhere. http://www.symantec.com/pcanywhere/Consumer/.
4. L. Beca, G. Cheng, G.C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, and K. Walczak. Tango, a collaborative environment for the world wide web. http://trurl.npac.syr.edu/tango/papers/tangowp.html.
5. L. Beca, G. Cheng, G.C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, and K. Walczak. Web technologies for collaborative visualization and simulatio. http://trurl.npac.syr.edu/tango/papers/tango_siam.html.
6. M. Bhandarkar, G. Budescu, W.F. Humphrey, J.A. Izaguirre, S. Izrailev, L.V. Kalt, D. Kosztin, F. Molnar, J.C. Phillips, and K. Schulten. Biocore: A collaboratory for structural biology. In *Proceedings of the SCS International Conference on Web-Based Modeling and Simulation*, pages 242–251, 1999.
7. J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal on Computing Systems and Science*, 18(2),79:143–154.
8. J. Ferraiolo, F. Jun, and D. Jackson. Scalable vector graphics. Technical Report TR-11, SVG, 2002.
9. S. Franklin, J. Davison, and D.E. Harrison. Web visualization and extraction of gridded climate data with the ferret program. http://www.pmel.noaa.gov/ferret/ferret_climate_server.html.
10. M. Rousal and V. Skala. Modular visualization environment - mve. In *Proceedings of International Conference ECI 2000*, pages 245–250, 2000.
11. V. Skala. The mve and complete programming documentation and user's manual. http://herakles.zcu.cz.
12. R. Sprugnoli. Perfect hashing functions: A single probe retrieving methods for static sets. *CACM*, 20(11),77:841–850.