VChemLab: A Virtual Chemistry Laboratory. The Storage, Retrieval, and Display of Chemical Information Using Standard Internet Tools

Henry S. Rzepa and Alan P. Tonge

Department of Chemistry, Imperial College, London SW7 2AY

Received February 20, 1998

The design and implementation of two interactive Internet-based applications for the storage, retrieval, and display of molecular structures, spectra, and physicochemical properties is described. The first, suitable for small-scale laboratory or teaching applications, uses a JavaScript-controlled structure to load data from discrete files located on a server or a CD-ROM into a local Web browser. The second, developed for scaling to larger commercial-sized applications, uses a Java-based Distributed Object Computing architecture to allow a local client applet to have persistent access to a remote object database, in which information for the different chemical classes has been stored. Both browser applications have a dynamically created page structure and use readily available Plugins and Java applets for the active display of three-dimensional molecular structures and their experimental spectra.

INTRODUCTION

The World Wide Web (WWW) has developed rapidly in the past 5 years as an accessible medium for the distribution of information and entertainment to those who have access to the global Internet network. Using free or low-cost browser software such as Netscape Navigator or Microsoft Internet Explorer, which have been developed to run on a variety of desktop computer platforms, users can retrieve and display documents from remote computer servers or from local storage devices such as CD-ROM, which have been laid out in standardized HyperText Markup Language (HTML) and hyperlinked to other marked-up documents. HTML allows the inclusion within the document of graphical images and other 'multimedia' features such as sound and video files may also be incorporated, although these require additional software to be replayed. In addition to displaying text and images, the potential of static HTML documents has been further enhanced by a number of recent developments that allow user-controlled active content:

Plugins. Small compiled applications that can parse specific file types defined by the MIME type filename extension and display the contents within the HTML document.[1,2] Machine dependent and must be installed locally by the user.

Java Applets. Small applications written in the Java programming language and downloaded from a remote server as machine-independent compiled 'byte code' when requested by the local client. This byte-code must be interpreted by a local Java interpreter to create the executable machine code. They are device independent, and tend therefore not to be optimized for system performance.

JavaScript. An event-driven scripting language,[3] incorporated within an HTML document, which allows dynamic user control of standard HTML page elements and interframe communication.

There has been a rapid expansion of chemical applications that use Web technology,[4] and standards have been introduced to describe the chemical information used.[1,2] The impact of the Web on computational chemistry software developers has already been significant. Within the last 18 months, a number of developers have begun to target the corporate 'Intranet' market by providing modeling tools − structure drawing, query handling, data presentation & analysis − with a consistent Web-based interface that can be easily learned by 'web-aware' researchers and chemists.[4−6] Such tools, e.g. Discovery.Net from Tripos, WebLab from Molecular Simulations, ChemScape Chime from MDL, InteractiveLab from ACD Labs, have applet- or plugin-based applications that run locally on standard browsers and are connected to networked computational or company database servers. Similarly, Web-based interfaces have also been adopted by groups designing Laboratory Information Systems (LIMS); for example, an in-house spectral database & structure-query system at Kodak,[7] and the U.S. National Institute of Standards and Technology has instituted the NIST Chemistry WebBook as an on-line resource of published infrared and mass spectral data.[8] The rationale behind these developments is twofold. First, they provide platform-independent applications, running on Mac, PC, or Unix operating systems, which can be downloaded via an Internet link to multiple users from a central source, allowing rapid and centralized software maintenance.[9] Second, it is hoped that the provision of standardized Web applications will both reduce the need for user retraining with new software developments and also promote better communication and information exchange within research groups.

DISCUSSION

1. Standalone JavaScript Application. The initial proposal for VChemLab was to provide a single Web-based information resource that could be used in conjunction with practical laboratory teaching courses here at Imperial College. Information for such courses − molecular structures, physicochemical data, reference spectra, safety & toxicological information and practical details of synthetic procedures −

10.1021/ci9803280 CCC: $15.00
Published on Web 09/04/1998

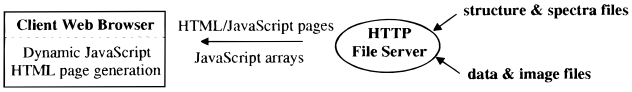

Figure 1. VChemLab client-server architecture.

```
function Molecule (ID, Molname, Filename, Project, Formula,
         h1nmr, c13nmr, ir, uv, ms, xray,
         mp, bp, ri, alpha_d,
         safety0, safety1, safety2, safety3) {

  this.ID    = ID;
  this.Molname = Molname;      this.Filename = Filename;
  this.Project = Project;      this.Formula  = Formula;
  this.h1nmr = h1nmr;          this.c13nmr = c13nmr;
  this.ir    = ir;             this.uv     = uv;
  this.ms    = ms;             this.xray   = xray;
  this.mp    = mp;             this.bp     = bp;
  this.ri    = ri;             this.alpha_d = alpha_d;
  this.safety0 = safety0;      this.safety1 = safety1;
  this.safety2 = safety2;      this.safety3 = safety3;

  return this;
}

var Mol = new Array(100);
var numMols = Mol.length;

Mol[ 0]  = new Molecule (0, "benzene",
            "benzen", "solvent", "C6H6",
            0,0,0,1,0, 5.5,80.0,1.501,0.0,
            "LD50 : 52g/kg (rat oral)", "Suspected carcinogen",
            "Leukemogen", "Fire hazard");
Mol[ 1]  = new Molecule (1, "styrene oxide",
            "styrene_oxide", "NMR", "C8H8O",
            1,0,1,0,1, -35.6,194.1,1.53,-22,5,
            "Moderately toxic", "Animal carcinogen &amp;amp; Teratogen",
            "Skin &amp;amp; eye irritant", "");
```

Figure 2. Construction of *JavaScript* molecule data array object (file *JavaScriptArray.js*).

Table 1. File Types Used by VChemLab

data	file type
3D structure coordinates	MDL molfile[10]/Brookhaven PDB (formatted ascii)
IR,NMR,UV,MS spectra	JCAMP-DX[11−13] (formatted ascii)
images	GIF (binary)
properties (physicochemical,safety,biological)	JavaScript arrays (string,integer & fp variables)

is often poorly available, scattered around laboratories and libraries in manuals that are easily damaged or mislaid. The development of VChemLab would provide chemistry students, who are generally highly computer literate, with an accessible and intuitive computer-based source of such data, which could be readily updated to include new information or subsequent changes to the course content. VChemLab was therefore first developed as a standard client-server operation with no control or limitation placed over user access. Initial work concentrated on developing the structural organization, and an early decision was made to employ JavaScript[3] in conjunction with the standard HTML Web page documents to permit the dynamic control of the displayed information through client-side searching of JavaScript data arrays (Figure 1).

Information used within VChemLab may be divided into four types and this is initially stored in files under the operating system directory structure on the server (Table 1). The JavaScript molecule arrays are used for storing basic information; variables such as chemical name, project area, molecular weights, melting points, and safety data are stored within the arrays as strings, integers, or floating point variables. Additionally, all entered structures must by default have a two-dimensional (2D) structure representation (GIF image) and a set of three-dimensional (3D) coordinates; the presence or absence of an experimental IR or NMR spectrum or a X-ray crystal structure is indicated from a set of logical switches. All arrays are held in a single JavaScript file as shown in Figure 2.

The hierarchical HTML page structure of VChemLab is shown in Figure 3, indicating how all information flow and

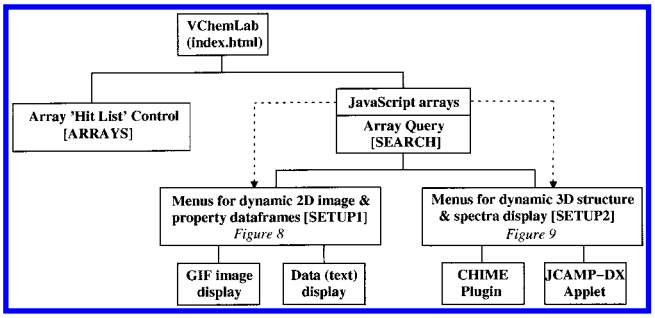

Figure 3. HTML Page hierarchy for VChemLab.

```
function checkProject(chosenProject) {

 for (ii=0; ii<numMols; ii++) {
   if (chosenProject == Mol[ ii].Project) {
      parent.frames[ 'ARRAYS'].document.MolOnOff.elements[ ii].value = 1;
   }
 }
}
```

Figure 4. JavaScript code in SEARCH frame for switching hidden element values in the adjacent ARRAYS frame.

display is controlled via the associated JavaScript object hierarchy that has been built into the HTML pages. When the top search level of VChemLab is accessed, then the whole of the JavaScript molecule data array is downloaded from the server to the client and is made available to the SEARCH function frame. This function allows the user to set up queries based on simple string searches, so that a subset of the arrays may be created based, for example, on project area, chemical name, or some toxicological property. Those entries satisfying the search criteria are flagged, and this information is then passed over to a set of hidden form elements declared in HTML in the adjacent ARRAYS frame:

```
<FORM NAME="MolOnOff">
  <INPUT TYPE="HIDDEN" NAME="Mol0" VALUE=0>
  <INPUT TYPE="HIDDEN" NAME="Mol1" VALUE=0>
  <INPUT TYPE="HIDDEN" NAME="Mol2" VALUE=0>
  .....
</FORM>
```

switching the value of these elements from 0 to 1 if a hit is found (Figure 4).

These form elements must be kept isolated from the SEARCH frame in order to prevent the hidden data element 'hit list' from being re-initialized each time the SEARCH frame is reloaded as we pass between the Property and Structure-Spectra SETUP display frames. Unlike compiled languages, such as Fortran or Java, we cannot create *common* data arrays in JavaScript to hold information. Data variables are local to the page in which they are created and can only be accessed via the JavaScript object heirarchy, which has a strict parent−child relationship based on the heirarchical frame structure shown in Figure 3. Hence, to pass between the Property and Structure-Spectra framesets, we must always go via the same route, otherwise we would lose both the variable information location and values required to dynamically create the select menu lists in the SETUP frames with JavaScript (Figure 5).

Structure and spectra display frames are grouped together (at some future date we would like to make direct associations between spectral frequencies and structural elements) and their files downloaded to the client browser from specified directory locations on the server when the associated menu item is selected. The files are displayed using the appropriate plugin or applet, MDL ChemScape Chime[14]

```
<SELECT NAME="structure" onChange="LoadFrame('CHIMEFRAME');">

<SCRIPT LANGUAGE="JavaScript" SRC="JavaScriptArray.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript">

        /*    This loads drop-down menu     */

  var numMols = Mol.length;
  var pointer = new Array(numMols);
  var Kount = 0;

  for (jj = 0; jj < numMols; jj++) {

   if (parent.parent.ARRAYS.document.MolOnOff.elements[ jj].value == 1) {

    pointer[ Kount] = jj;
    Kount = Kount + 1;
    if (jj == 0) {
     document.writeln ("<OPTION VALUE=" + Mol[ jj][ 'FilName'] + " SELECTED>"
        + Mol[ jj][ 'Molname'].toUpperCase() );
     } else {
     document.writeln ("<OPTION VALUE=" + Mol[ jj][ 'FilName'] + ">"
        + Mol[ jj][ 'Molname'].toUpperCase() );
    }
   }
  }

</SCRIPT>
</SELECT>
```

Figure 5. JavaScript and HTML code for dynamically creating HTML tags for a select menu list.

```
var idx = parent.SETUP.document.molecule.structure.selectedIndex
var file = parent.SETUP.document.molecule.structure.options[ idx].value;

PDBFilNam = "COORDS/" + file + ".pdb";          //structure file URL
document.write("<BR><EMBED SRC=" + PDBFilNam +
      " WIDTH=175 HEIGHT=175 spiny=30 bgcolor=#00006b" +
      " startspin=true display3D=ball&stick" +
      " script='select elemno=6; colour atom [ 0,160,0] ;" +
      " select *.Cl; spacefill 135;" +
      " select hydrogen; spacefill 90; select *; zoom 140;'>");
```

Figure 6. JavaScript code for dynamically creating HTML tags in the *Chime* plugin display frame.

```
<SCRIPT LANGUAGE="JavaScript" SRC="JavaScriptArray.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript">

   var idx = parent.SETUP.document.molecule.structure.selectedIndex
   var selected = parent.SETUP.pointer[ idx];

   document.write('<BR><FONT COLOR=red><CENTER><B>');
   document.write(Mol[ selected][ 'Molname'].toUpperCase() +
         '</B></FONT><BR><BR>');
   document.write('<FONT COLOR=#00008B>');

   if(option == "physical") {

     document.write('Melting Point : ' + Mol[ selected].mp + '<BR>');
     document.write('Boiling Point : ' + Mol[ selected].bp + '<BR>');
     document.write('Refractive Index : ' + Mol[ selected].ri + '<BR>');
     document.write('Rotation : ' + Mol[ selected].alpha_d + '<BR>');
     document.write('Molecular Weight : ' + Mol[ selected].mw + '<BR>');
   }

   if(option == "safety") {

     for ( ii = 0; ii <= 3; ii++) {
       var id = "safety" + ii;
       document.write('\n' + Mol[ selected][ id] + '<BR>');
     }
   }
   document.write('</FONT></CENTER>');

</SCRIPT>
```

Figure 7. JavaScript code for dynamically creating HTML tags of text display of physicochemical properties.

for the structure files or a JCAMP-DX spectral viewer applet developed in-house,[15] with the necessary HTML code generated dynamically with JavaScript (Figure 6) for the display frames as each example is chosen from a SETUP menu, hence avoiding the need for a unique page for each structure or spectrum.

The HTML for the physicochemical and safety data displays is similarly created by dynamic JavaScript generation from the appropriate JavaScript array elements (Figure 7) and sample display frames are shown in Figures 8 and 9.

2. Distributed Object Computing Architecture. Corporate Intranets, which use Internet standards and protocols as a means of communication, are being rapidly developed as a cost-effective method of sharing data resources and distributing reusable software components.[16] Pharmaceutical companies are investing heavily in data-intensive technologies, such as those associated with high-throughput screening (HTS) and combinatorial libraries,[17] and there is a growing requirement that Intranets should support multiple views into such stored information by making use of data objects, which

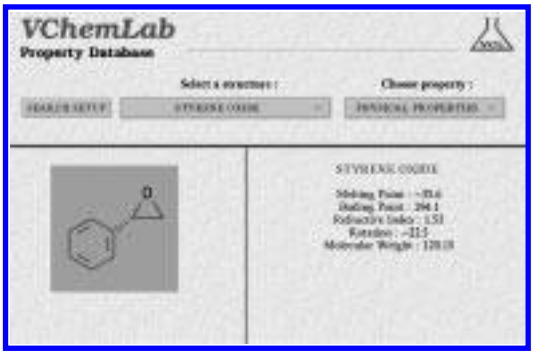

Figure 8. 2D Structure and property display frames.

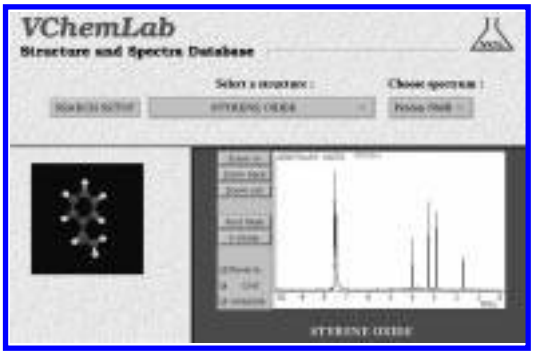

Figure 9. Interactive 3D structure and spectra display frames.

contain the same core information but allow different users to view different attributes of the object depending on their requirements. There are, therefore, a number of reasons for extending VChemLab beyond the simple client−server structure just described and developing an application that would enable users to remotely interrogate a fully integrated chemistry database, containing entries for 3D structures, spectra, and physicochemical and biological data.

Scalability. Although the current structure would probably extend to 100s of molecule entries for the downloaded JavaScript arrays, browsers are not designed for large-scale data handling applications. Hence, we need a design that would allow users to submit searches on a full database remotely to a server, before downloading a manageable subset to the browser.

Searchability. A properly constructed database would allow users to conduct chemical substructure searching and store associations between different chemical and biological data types, such as those derived from HTS.

Security. We would like the ability to restrict access in certain project areas and also allow remote update of the database by designated users.

We have therefore begun to develop VChemLab for a Distributed Object Computing environment, whereby a client process operating on a local computer can have access over a Network to the methods and data handling procedures of remote objects running on a different machines. In principle, these objects would be platform, operating system, and programming language independent, although this requires adherence to a heterogeneous communications interface standard such as that defined by the Common Object Request Broker Architecture (CORBA).[18] Java is an object-oriented programming language has become increasingly popular with WWW users in the past few years, with Java-based applets allowing complete client-side browser applications to be downloaded from a remote server. With the release of the

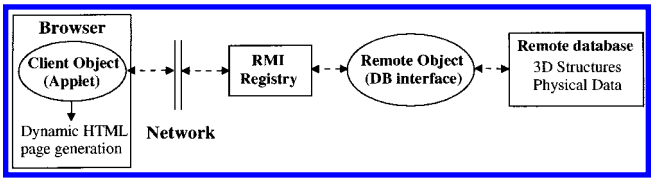

Figure 10. VChemLab distributed object computing environment.

```
import java.io.*;

public class molData implements Serializable {

     String molName; String filName; String Project;
     double mp; double bp; double ri; double alpha_d;
     public int molArray = 4; String safe[ ] = new String[ molArray];

   //        Create constructor method for class molData

     public molData (String molName2, String filName2, String Project2,
           double mp2, double bp2, double ri2, double alpha_d2,
           String safe0, String safe1, String safe2, String safe3) {
   //
   //        Associate the object variables with
   //        arguments passed to the constructor
   //
        this.molNam     = molName2;       this.filName    = filName2;
        this.mp         = mp2;            this.bp         = bp2;
        this.ri         = ri2;            this.alpha_d    = alpha_d2;
        this.safe[ 0]   = safe0;          this.safe[ 1]   = safe1;
        this.safe[ 2]   = safe2;          this.safe[ 3]   = safe3;
     }
      public String getMolName() { return molName;}
      public double getMP() { return mp;}
      public double getBP() { return bp;}
      public String getSafety(int ii) { return safe[ ii];}
   }
```

Figure 11. Representative Java class definition for the molecular data object.

```
import java.rmi.*;

public class RMIStart {

      public static void main(String args[ ]) {

//          Install the security manager. This prevents this
//          (server) object from accessing remote (client)
//                objects loaded from another host

        System.setSecurityManager(new RMISecurityManager());

        try {

//          Tell the process where it is - otherwise it will
//          use the machines local name and give a security
//          exception when you try to attach client object (applet).

            System.getProperties().put("java.rmi.server.hostname",
                  "www.this_domain_name");

            remotePSE servlet = new remotePSE ();
            Naming.rebind("serviceName", servlet);
            servlet.setMainThread(Thread.currentThread());

        } catch (Exception e) { }
      }
}
```

Figure 12. Standalone program to register the remote object 'remotePSE' with the RMI registry.

latest Java Development Kit (JDK 1.1),[19] it has become possible to develop a homogeneous object-based distributed computing architecture, written entirely in Java, known as Remote Method Invocation (RMI).[20,21] This allows a persistent network connection to be maintained between the client-based application, such as an applet, and the remote server-based object, such as a database interface (Figure 10). In this, a Java client running on one virtual machine can request the services of the remote object, running on a separate virtual machine, by making a request to the RMI registry on the same host as the remote object. The RMI registry runs as a background application on the remote host and contains references to the active remote objects, which bind to the registry using specified service names. When the registry accepts the clients request, it returns a reference to the remote object back to the client in the form of a stub. The actual exchange of parameters between client and server is controlled by the remote object's stub and skeleton, which are client- and server-side proxies of the remote object.[20]

Other distributed computing architectures are possible; for example, using the Common Gateway Interface (CGI) or CORBA technology. CGI has an advantage in not requiring

```
//    This class implements the remote interface for use as
//    an RMI remote object. It supports multiple users by using
//    an array of the 'connectPSE' connection object, which
//    contains the logic for database assess and data retrieval.
//    The separate interface class defines those methods which
//    are available to the client (with no programming logic).

import java.util.*;      //for Vector
import java.rmi.*;
import COM.odi.*;        //ObjectStore PSE classes

public class remotePSE extends UnicastRemoteObject implements interfacePSE {

  private connectPSE PSEthread[ ] = new connectPSE[ 5];
  Thread mainThread;

  public remotePSE() throws RemoteException {}         // default constructor

  public void setMainThread(Thread thread) {

//          Initialize PSE with the main RMI Server thread -
//          all cooperating threads must subsequently call
//                ObjectStore.initialize(mainThread)

     mainThread = thread;
     ObjectStore.initialize(null, null);
  }

//          Create the remote threads to be used by client

  public int openDatabase() throws RemoteException {

     int connectionID;
     synchronized(PSEthread) {

       for (connectionID=0; connectionID < PSEthread.length; connectionID++) {

           if (PSEthread[ connectionID] == null) break;
       }
       if (connectionID >= PSEthread.length) {

           return -1;         //Out of connections
       }
       PSEthread[ connectionID] = new connectPSE(mainThread);
     }

     PSEthread[ connectionID].start();
     PSEthread[ connectionID].openDatabase();
     return connectionID;
  }

  public Vector getMoleculeList(int id) throws RemoteException {

     return PSEthread[ id].getMoleculeList();
  }
  public molData getData(int id, String molName) throws RemoteException {

     return PSEthread[ id].getData(molName);
  }
}
```

Figure 13. Java code for the remote object.

Java capability in the client. However, it is a stateless client−server model and there is no persistent connection to a database, so the run-time processes must be reestablished with each connection, leading to significant processing overheads on the server. CORBA allows the use of a persistent heterogeneous programming environment, although any performance loss with the RMI will hopefully disappear with the development of more efficient Java compilers. Furthermore, with the availability of object databases, as distinct from the more commonly found relational or heirarchical databases, it is now possible to store and retrieve these objects within a database in the same form as the object-oriented programming environment (Figure 11).

We have therefore developed a set of Java classes (i.e., data structures and their associated handling methods) for several types of molecular object, including physicochemical (Figure 11) and 3D structural information. These classes have been incorporated within an ObjectStore PSE object database[21] and a set of Java routines for retrieving and displaying the information from the database using the RMI in a multi-user application has been developed. Each new instance of a molecular object is created from file and added with an identifying key into a (persistence capable) Java hashtable, which is stored within the object database when complete. Methods for accessing the database are defined in a connection class that is called by the remote object running as a background process on the server. Sample code for the multi-user remote object, which manages multiple client calls by activating an array of threads, and for registering it with the RMI registry are shown in Figures 12 and 13.

```
import java.rmi.*;
import java.applet.*;
import netscape.javascript.JSObject;

public class appletRMI extends Applet {

      public void init() {

        registryName = "rmi://" + getCodeBase().getHost() + "/serviceName";
        PSEservlet   = (PSEServerInterface)Naming.lookup(registryName);
      }
      JSObject win = JSObject.getWindow(this);        //browser window
      TextArea outText;
      List pickList = new List(10,false);
      int connectID;

      public void start() {

//              Open the database for searching

            List pickList = new List(10,false);
            Vector molList = new Vector();

            connectID = PSEservlet.openDatabase();
            molList = (Vector)PSEservlet.getMoleculeList(connectID);

            Enumeration mols = molList.elements();
            String tempKey;
            while (mols.hasMoreElements()) {
                  tempKey = (String)mols.nextElement();
                  pickList.addItem(tempKey);
            }
      }
      public boolean action(Event evt, Object o) {

         if (evt.target == dataButton) {

            molName = (String)pickList.getSelectedItem();
//
//              Return database entry and write out coords
//
            PSEservlet.writeCoords(connectID, molName);
            tempMol = (molData)PSEservlet.getData(connectID, molName);

            outText.appendText("\nSafety :\n"
            for (ii = 0; ii <=3; ii++) {
                  safeMessage = tempMol.getSafety(ii);
                  outText.appendText("\n  " + safeMessage);
            }
            outText.appendText("\nPhysicochemical :\n");
            outText.appendText("\n  m.p.: " + tempMol.getMP() +
                  " Boiling Point : " + tempMol.getBP() - "\n");
//
//          Write to JavaScript browser function "newData(mp,bp)"
//
            String dataStr = "newData(\""+ tempMol.getMP() +
                  "\",\"" + tempMol.getBP() + "\")";
            win.eval(dataStr);
          }
         return true;
      }
}
```

Figure 14. Sample Java code for the client Applet, showing local calls to the methods of the remote interface (italics) and the returned object (bold).

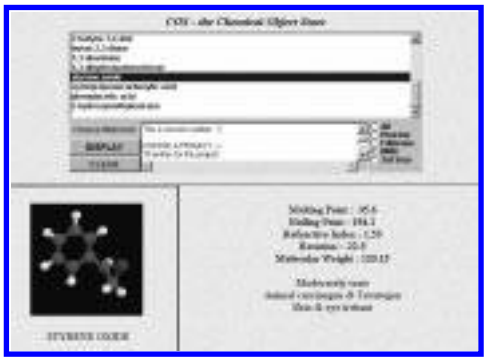

Figure 15. Browser interface to VChemLab *Distributed Object Computing* module.

When the applet is downloaded from the server to the client, it links itself to the RMI registry also running on the server and creates a persistent interface to the remote object, which is bound to the registry by its service name (*init()* method in Figure 14). By calling the remote interface methods, the applet retrieves a list of stored database objects and then displays them, or a subset, as a menu list when it starts up (*start()* method). When a particular entry is chosen (*action()* method), then the whole of that object is returned to the applet from the remote database using the RMI object serialization procedure − making available all objects' data and class handling methods (shown in bold type; *cf*. methods shown in Figure 11). The data entries may then be retrieved locally using the remote object's class methods and displayed either within the applet or passed on further to the browser window.

The 3D coordinates for the chosen entry are written out to a temporary file on the server where they can be accessed by the Chime plugin,[14] which is reloaded by a Java-JavaScript call from within the applet, allowing client-based dynamic HTML page generation (Figure 15).

IMPLEMENTATION

A working version of VChemLab, suitable for viewing by both current JavaScript-enabled major commercial browsers [Netscape Navigator 3.01 (or higher) and Internet Explorer 4.0] has been published on CD-Rom.[22]

A trial demonstrator version of VChemLab for Distributed Object Computing has been developed[23] using the Java RMI to allow downloaded client applets persistent access to a small-scale ObjectStore PSE database, containing full 3D coordinate and physicochemical data objects, located on the remote Unix HTTP server. The client applet (Figure 15) may be viewed with the latest release of the Netscape Navigator (v4.04 or higher) browser,[24] which fully implements JDK1.1 methods and also permits Java-JavaScript communication. We will report in a future article the scaleable characteristics of this system for large molecular object databases.

ACKNOWLEDGMENT

We gratefully acknowledge the Joint Information Systems Committee (UK) for a JTAP award to fund this project.

REFERENCES AND NOTES

(1) Rzepa, H. S. The Application of Chemical Multipurpose Internet Mail Extensions (Chemical MIME) Internet Standards to Electronic Mail and World-Wide Web Information Exchange. accepted for publication in *J. Chem. Inf. Comput. Sci.*

(2) Murray-Rust, P.; Rzepa, H. S.; Whittaker, B. J. The World Wide Web as a Chemical Information Tool. *Chem. Soc. Rev.* **1997,** *27*, 1−10.

(3) Kent, P. *Official Netscape JavaScript 1.2 Book*; Ventura, 1998.

(4) Krieger, J. H. Web Technology and Drug Discovery Power Fast-Moving Development Efforts in Computational Chemistry Software. *Chem. Eng. News* (16th September 1996), 30−37 (*http://pubs.acs.org/hotartcl/cenear/960916/software.html*).

(5) Borman, S. Web Explodes with Chemical Applications, *Chem. Eng. News* (16th September 1996), 38−40 (*http://pubs.acs.org/hotartcl/cenear/960916/explode.html*).

(6) Krieger, J. H. Computational Chemistry Impact. New Developments in Underlying Software Propel Technology into Activities Spanning Drug Development Cycle, *Chem. Eng. News* (12th May 1997), 30−40 (*http://pubs.acs.org/hotartcl/cenear/970512/comp.html*).

(7) Borman, S. Lab Systems Embrace the Web. *Chem. Eng. News* (27th January 1997), 25−27.

(8) National Institute of Science and Technology (*http://webbook.nist.gov/*).

(9) Town, W. G. Impact of New Technologies on the Delivery of Scientific and Patent Information Products: Delivery of Chemical Information via the Internet. In *Proceedings of the 1996 International Chemical Information Conference*; Collier, H., Ed.; Infonortics: Calne, England, 1996; p 149.

(10) Dalby, A.; Nourse, J. G.; Hounshell, W. D.; Gushurst, A. K. I.; Grier, D. L.; Leland, B. A.; Laufer, J. Description of Several Structure File Formats Used by Computer Programs Developed at Molecular Design Limited. *J. Chem. Inf. Comput. Sci*. **1992**, *32*, 244−255.

(11) McDonald, R. S.; Wilks, P. A. JCAMP-DX: A Standard Form of Exchange of Infrared Spectra in Computer Readable Form. *Appl. Spectrosc.* **1988**, *42*, 151−162.

(12) Davies, A. N.; Lampen, P. JCAMP-DX for NMR *Appl. Spectrosc.* **1993**, *47*, 1093−1099.

(13) Lampen, P.; Hillig, H.; Davies, A. N.; Lindscheid, M. JCAMP-DX for Mass Spectrometry. *Appl. Spectrosc*. **1994**, *48*, 1545−1552.

(14) Molecular Design Ltd., San Leandro, CA 94577 (*http://www.mdli.com/*).
(15) Cottenceau, G., unpublished work in these laboratories.
(16) Greer, T. *Understanding Intranets*, Microsoft: Redmond, WA, 1998.
(17) Sullivan, M. Chemistry on the Web: A New Approach to Chemical Discovery. *Today's Chemist at Work* **1997**, *6(6)*, 13−16 (*http://pubs.acs.org/hotartcl/tcaw/97/jun/chem.html*).
(18) Object Management Group, Framingham, MA 01701 (*http://www.omg.org/*).
(19) Sun Microsystems Inc., Palo Alto, CA 94303 (*http://www.sun.com/*).
(20) *http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC-.doc.html*.
(21) Horton, I. *Beginning Java*; Wrox: Birmingham, England, 1997; Chapter 19.
(22) Object Design Inc., Burlington, MA 01803 (*http://www.odi.com/*).
(23) Tonge, A. P.; Rzepa, H. S. The VChemLab Database System. In *ECTOC-3. Electronic Conference on Trends in Organometallic Chemistry*; Rzepa, H. S.; Leach, C., Eds.; Royal Society of Chemistry: London, 1997. (*http://www.ch.ic.ac.uk/vchemlab/*).
(24) An online version is available at: *http://www.ch.ic.ac.uk/vchemlab/cos/*.
(25) Netscape Communications Corp., Mountain View, CA 94043 (*http://home.netscape.com/*).

CI9803280