ELSEVIER

Design and implementation of a collaborative molecular graphics environment

John G. Tate,* John L. Moreland,* and Philip E. Bourne*†‡

**San Diego Supercomputer Center, University of California, San Diego, La Jolla, CA 92093-0505, USA*
†The Burnham Institute, La Jolla, CA 92037, USA
‡Department of Pharmacology, University of California, San Diego, La Jolla, CA 92093-0636, USA

During the determination of macromolecular structures, scientists routinely use complex graphics software to display various representations of the molecule of interest. Once the structure determination is complete, coordinates are deposited in the Protein Data Bank (PDB), from where anyone with an Internet connection may download and view them or request them on CD-ROM. However, the currently available visualization software is such that casual users, whose expertise may not be in structure determination, often cannot obtain useful images of interesting molecules without expendiing considerable time and effort. Existing visualization software is generally very complex, requiring a high degree of familiarity to obtain the best results, or else it is too simplistic to provide users with the level of customizability needed to get the most out of the atomic coordinates. Few of the existing software packages have the capability for collaborative visualization via the Internet. These and other issues are being addressed by the Molecular Interactive Collaborative Environment (MICE) project (http://mice.sdsc.edu/). The core of the MICE project is the MICE application, an interactive molecular structure viewer with built-in collaborative capabilities. MICE not only addresses the issues of usability and flexibility but also extends the role of traditional visualization tools by allowing multiple users to view, manipulate, and interact with a single representation of a macromolecular structure. MICE is written entirely in Java, using the Java3D extensions for rendering and manipulation of the three-dimensional scene, and the Common Object Request Broker Architecture (CORBA) communications suite to enable collaborative manipulation of that scene. © 2001 by Elsevier Science Inc.

Keywords: MICE, molecular visualization, interactive, Internet, collaborative, Java, Java3D, CORBA

Color Plates for this article are on pages 369–373.

Corresponding author: Dr. Philip E. Bourne, University of California, San Diego, San Diego Supercomputer Center, 9500 Gilman Drive, La Jolla, CA 92093-0505, USA. Tel.: 1-858-534-8301; fax: 1-858-822-0873.

E-mail address: bourne@sdsc.edu (P.E. Bourne)

INTRODUCTION

At the present time, the majority of tools for visualizing protein structures fall into one of two categories: either the program is very powerful and difficult to use, but provides a highly customizable view of a structure, or else it is of limited scope but relatively simple to use. Few programs, if any, bridge this gap and provide users with both powerful features and a simple and intuitive interface. The Molecular Interactive Collaborative Environment (MICE) is a project that aims to address these and other issues surrounding the visualization of macromolecular structures and to leverage current and emergent technologies to improve the mechanisms for viewing, analyzing, and disseminating structural information across the Internet.

The MICE application is an interactive, three-dimensional (3D) structure viewer, and may be run as a standalone application or as a Java™ applet embedded in a web page. The structure viewer is coupled with a front-end that allows the user to generate a view of any structure in the Protein Data Bank (PDB) (http://www.rcsb.org/pdb/).[1] Generation of the interactive "molecular scene" is performed via a simple point-and-click interface: any PDB entry may be viewed without the need for a user to download any atomic coordinates. Instead, the user selects the attributes of a scene and the MICE client submits a Common Gateway Interface (CGI) request to a web-based server. The server obtains the atomic coordinates from the PDB via a fast network connection and then, on-the-fly, generates and returns only a Virtual Reality Modeling Language (VRML)[2] representation of the molecule. No atomic coordi-

Journal of Molecular Graphics and Modelling 19, 280–287, 2001

655 Avenue of the Americas, New York, NY 10010

1093-3263/01/$–see front matter
PII S1093-3263(00)00055-3

nates are returned to the MICE client. In addition to generating a custom scene using the web-based server, MICE can also load 3D content from a standard VRML world file, which may reside either on the same machine as the MICE client or on a networked machine. The feature of MICE that differentiates it from existing structure viewers is the ability to distribute an entire molecular scene across a network, so that multiple users on different remote computers may view and interactively manipulate the same scene simultaneously. The term "molecular scene" refers to a detailed three-dimensional view of a protein structure, comprising not only the geometry of the model itself but also features such as multiple viewpoints, annotations, and additional geometry elements such as a 3D pointer. The built-in collaborative capabilities of MICE allow a user to load a scene depicting a macromolecular structure into a MICE client on their local machine, and then "publish" this scene to whomever wishes to view it. Another MICE user at a different location may connect to this session and "subscribe" to all aspects of the scene. Subscribed users automatically receive the full 3D description of the scene, as seen from the viewpoint of the currently publishing user; as the publisher rotates, zooms, and even alters the content of the scene, these changes are seen, in real-time, by any subscribed user.

The concepts introduced in MICE, such as sessions and client/server interactions, are common in the numerous applications that implement non-domain-specific collaborative environments, such as Microsoft's NetMeeting[3] or Habanero,[4] from the National Computational Science Alliance (NCSA). These applications provide a range of tools and services that facilitate collaborative work environments, such as video conferencing, chat tools, and shared white boards. Aimed at a wider market, the generic collaboration applications tend to recreate an environment that is more closely tied to a business setting, rather than a scientific one, but the proliferation of such applications is clear proof of the power and appeal of interactive collaboration across the Internet. In the area of protein structure visualization, collaboration is an equally compelling concept. MICE raises the possibility of quickly and easily displaying the same view of a protein structure on multiple remote computers and gives rise to many exciting new scenarios, from simple remote interaction between collaborating scientists to distance learning involving a single teacher and many geographically distant students.

SCENE ACQUISITION

Molecular Scene Generator

MICE has been designed to require minimal user input to generate a useful representation of a molecule. Choosing the "Import PDB . . ." option from the "File" menu calls up a simple form interface (shown in Color Plate 1) that allows a user to choose the molecule to display and then the style in which to render it. The molecule must be one that is stored in the PDB; a user selects a molecule by its PDB ID code, sometimes called the accession code (MICE can also be used to view molecules whose structure is available from a local PDB-format file; see below). To alter the style of rendering, the user chooses the required attributes of the scene from sets of radio buttons and lists, arranged on five tab-panels. Many elements of the scene may be configured, from the style of representation of the protein backbone (e.g., smooth tube or secondary structure elements) to the coloring scheme used for depicting author-defined "sites" on the molecule (e.g., color by site or color by atom type). When the user has selected the molecule and style of depiction, pressing the "Import" button constructs a standard CGI request that is submitted via HTTP to a dedicated scene generator, the Molecular Scene Generator (MSG).[5] Color Plate 2 shows a series of scenes rendered in MICE, each with a different set of characteristics.

MSG is composed of a set of CGI scripts, written entirely in Perl and running under a standard web server. MSG accepts CGI requests either from a MICE client or from a standard HTML form, and returns a URL pointing to a VRML file containing the requested scene. To generate 3D geometry from the client request, MSG constructs a script file for the external program MolScript (version 2.1).[6] MolScript is a molecular graphics program that is widely used for generating Postscript, VRML, or photorealistic representations of macromolecular structures. Although it is most commonly used as a command-line-driven program, MSG runs MolScript internally as part of the CGI process, capturing and editing the VRML output. The VRML scene is modified to add extra information, such as secondary structure composition and residue numbering, before being written to a network-accessible file on the web server. Finally, a URL is returned as the output from the CGI process, giving the client the location of the VRML file on the server. Although apparently redundant for the simple case of a single, isolated user, MSG returns a URL rather than the raw VRML for the benefit of the collaboration service, since content that will be shared by multiple users must be web accessible.

Reading a Scene from an External Source

When MICE is used to generate a customized scene, the scene is loaded in the form of a standard VRML scene from a remote web server. However, the structure display component of MICE is an entirely generic 3D-geometry viewer and can read a VRML file directly from a local disk or from a URL. Hence, a user can create a VRML scene on a local machine, independently of MICE, and the scene can be viewed as normal. If the VRML file is web accessible, it can also be shared with other MICE clients, just as if it had been generated using the MICE/MSG interface.

Some examples of standalone applications that can generate VRML representations of a molecule are MolScript, InSight,[7] and Ribbons.[8]

MICE ARCHITECTURE

Java Technology

MICE is written entirely in Java,[9] Sun Microsystems' object-oriented, platform-independent programming language, which is rapidly becoming the de facto standard for writing portable, network-savvy applications. MICE also requires Java3D, an extension to the Java language that adds high-level support for 3D graphics. The Java3D API allows a developer to easily construct and manipulate a scene graph-based description of three-dimensional geometry but removes the need to deal with the low-level management of scene traversal and rendering. Although Java3D requires at least version 1.2 of the Java

Runtime Environment (JRE), MICE can use any stable version of Java3D.

Since it is implemented entirely in Java, MICE may be run either as a stand-alone application or as an applet, running under a Java-enabled web browser such as Netscape Communicator or Microsoft Internet Explorer. Any Java program running as an applet is subject to certain security restrictions, designed to limit the scope of interaction between the applet and the underlying local computer system. These restrictions are intended to prevent malicious or unintentionally harmful code from doing damage to the host system, by stopping untrusted applets using various low-level services, such as the ability to make arbitrary network connections or access the local file system. To circumvent these restrictions, applets may be "signed" to certify that they have been obtained from a trusted source, and then the user must agree to grant such a signed applet access to wider system services. Although the mechanisms for running signed applets are already available, they remain somewhat obscure and difficult for the end-user to apply. With future releases of Java and the Java-Plugin, Sun Microsystems will introduce many improvements to the infrastructure supporting applets and support for signed applets will continue to grow.

In addition to promised improvements in the mechanisms for granting security privileges to applets, the next major release of Java will also improve the mechanisms for automatic installation and configuration of the Java environment from within a web browser. The popular web browsers generally include support for "plugin" applications: these are separate, third-party applications that can be downloaded and installed within the web browser when they are required for viewing a particular web page. This mechanism can already be used, for example, to ensure that visitors to a given page are running a suitable version of Java for any embedded applets in that page. If the detected JRE is not suitable, the browser can be directed to a new location that includes instructions and links for downloading and installing the required JRE.

Although it goes some way toward automating the installation of Java, this mechanism cannot yet check for the presence of additional Java components such as Java3D, which are required for an application like MICE. Fortunately, Sun has announced that the next version of Java will remedy this problem and include better support for automatic installation and configuration of separate Java components. Under the proposed schemes, web pages that include a Java applet will include HTML code that first directs an improperly equipped browser to download the correct version of the Java environment. Once Java is installed and working, the embedded applet is loaded and the Java system will examine the applet to determine if it requires additional components. If extra components, such as Java3D, are required, Java will direct the user to download and install those components. Finally, the applet will be executed under its favored runtime environment and with the required Java components.

It is not clear what level of user interaction may be required in this process, but at best, this scheme may require only a single click from the user to initiate the first download of the Java environment. Whatever level of user interaction is finally required, the proposed scheme has the advantage that users can be directed to download and install Java packages without the need for them to sift through numerous web pages and run several installer programs.

Once the technology stabilizes and the automatic installation of Java and Java3D can be implemented more easily, MICE will be made available through the PDB as an embedded applet. At this point, Java and Java3D will be automatically installed where required, and MICE itself will be delivered to the user as part of a standard web page without the need for the user to download, install, or even run MICE explicitly.

User Interface

MICE is constructed from a series of independent modules (see Figure 1) that interoperate seamlessly to form the single application. The user interface portion of MICE is written using the Swing toolkit, part of the standard Java classes (from version 1.2 of the Java Development Kit onward) that implement many commonly used graphical user interface (GUI) components or "widgets." The Swing toolkit provides a similar "look and feel" across all of the platforms on which Java applications may be run, allowing the programmer to design a single user interface that behaves similarly on multiple platforms. The GUI components of MICE are able to communicate with one another using Swing-style event objects and listeners. This scheme allows any object that needs to be notified of state changes in another object (such a state change may constitute an "event") to register itself as an interested party (a "listener"). When an event occurs, all registered listeners are notified and an agreed method is called in each listener object, allowing data and events to be served as needed from a single source to any number of interested destinations. This architecture enables applications to be written in a modular fashion, such that a given module need not know or care what other objects its data are destined for, nor what method calls (in other objects) are being triggered by its internal events. Similarly, a module that accepts data from a different object does not need to know the source of the data, and a module that receives an event notification has only to act on the event, without understanding what triggered it.

An important example of how this behavior is useful in the MICE architecture is the implementation of collaboration. Consider, for example, events that cause changes to the camera view in the geometry viewer module, that is, the orientation of the molecule as seen by an observer. Such events may be generated either by the local user dragging with the mouse, or, in a collaborative setting, they may be the result of the local MICE client tracking the motion of a remote user's camera. In either case, the local geometry viewer module must respond to the change in the camera position by updating the local view. It need not be aware of the origin of the event, and can legitimately treat events from either of the two very different sources in the same way. In this way, the process of adding collaborative capabilities to the geometry viewer is greatly simplified: the display component requires no special methods or capabilities to cope with remote changes to its local view. Instead, the display module simply registers itself as being interested in any event that will cause changes to its camera view and acts upon these changes as they arrive, regardless of their source.

Just as the event/listener model may be used to simplify the implementation of collaboration between different MICE clients, it also greatly simplifies the passing of data and events between components within the application. For example, the PDB Import module enables a user to choose the viewing

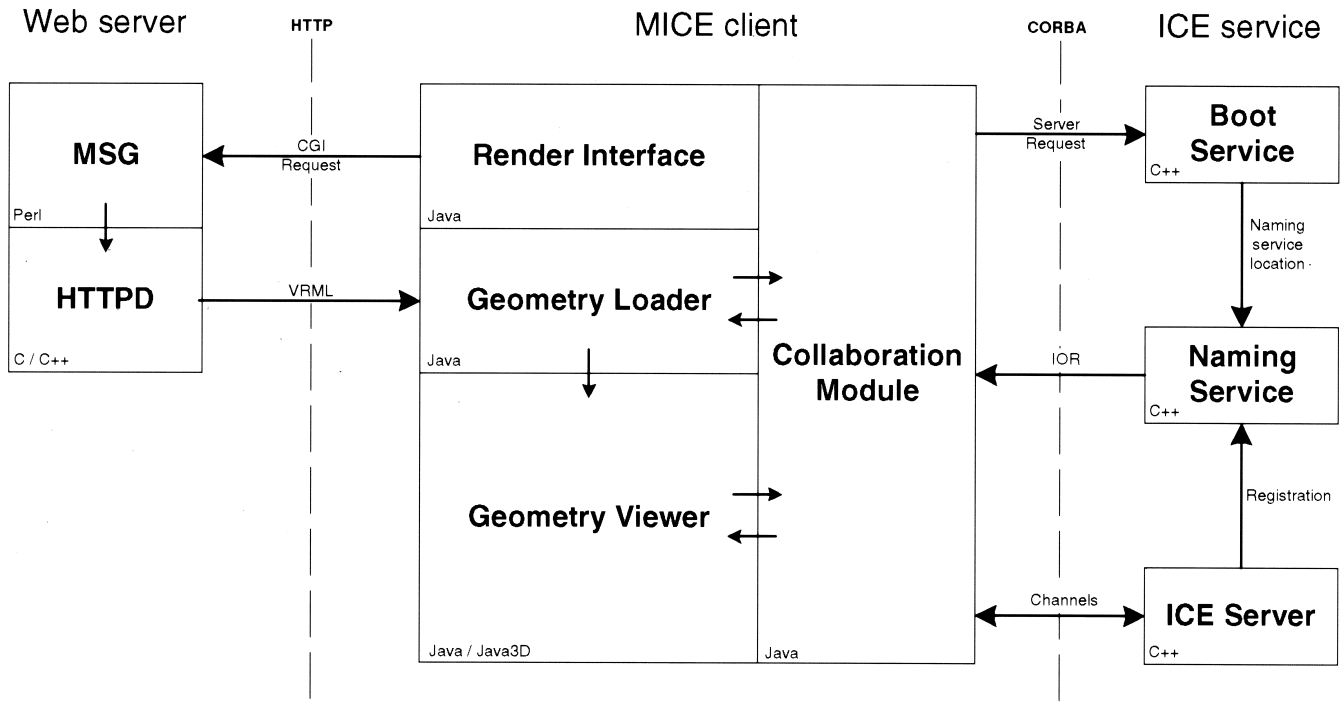


Figure 1. The internal architecture of the MICE client, and the interactions between MICE and network based scene-generation and collaboration services, using HTTP and CORBA protocols respectively.

characteristics of a particular molecule, and handles the submission of a CGI request to generate a 3D scene. When the user presses the "Import" button, the module sends the CGI request and receives back a URL giving the location of the VRML scene data. The import module now "fires" an event containing the URL and in response to this a geometry loader is invoked to download the scene and convert it from VRML into a native Java3D scene graph. Finally, the resulting scene graph is passed to the geometry viewer module and rendered for display. Because of the modular architecture of MICE, in which each component performs its function independently and simply passes on its events/data to any registered listeners, the core application need not be involved in this relatively complex transaction, allowing it to remain small, simple, and easy to maintain.

Geometry Viewer

The display and manipulation of the 3D molecular scene is performed using the Java3D extensions to the standard Java classes. Java3D provides a rich set of classes for manipulating 3D geometry using a scene-graph approach and can be layered on top of either OpenGL or Microsoft's Direct3D APIs, enabling for the first time the development of platform-independent, high-performance, and high-quality 3D graphics applications.

The MICE geometry viewer is a completely generic 3D-geometry viewer and does not include built-in support for molecular graphics scenes. Instead, it uses a "geometry loader" that converts a piece of plain VRML into a Java3D scene-graph object. The current geometry loader is the standard VRML loader from the VRML working group of the Web3D Consortium (http://www.web3d.org/). The geometry loader approach means that the geometry viewer is inherently reusable, and can accept content from a variety of sources, rather than from just a single source. Although VRML is currently a convenient transport format for 3D geometry, it has certain limitations that make it somewhat unsuitable for descriptions of molecular scenes. As Java3D-aware molecular graphics programs become available, the MICE geometry viewer will be able to accept molecular scenes as raw Java3D objects with only a few small changes to the wrapper code.

Once the scene data are loaded into a Java3D scene-graph, the Java3D environment takes over the low-level tasks associated with displaying the scene content, such as traversing the scene graph or rendering the geometry elements, which frees the application developer from the tasks associated with basic scene maintenance. Furthermore, since Java3D can take advantage of any available graphics hardware acceleration and can perform a range of scene-graph optimizations, the runtime environment automatically ensures that the finished Java3D application automatically runs as quickly as possible on any given system.

Once the scene has been displayed to the user, the geometry viewer provides various tools for manipulating and querying the 3D content. Most importantly, the representation of the molecule may be interactively rotated and translated, so that the user may view the scene from any angle or distance. Two viewer behaviors are currently implemented: the examine viewer and the walk viewer. The examine viewer implements a "virtual-sphere," which allows the user to rotate the molecular scene about its center point by dragging the mouse around the viewer window. The walk viewer allows the user to "fly" into and out of the scene by moving the viewpoint toward or

away from the visible objects, as if they were walking around the scene.

The geometry viewer also has rudimentary scene interrogation capabilities, which enable the user to "pick" objects within the scene and have basic information about the clicked point returned in a separate window. "Picking" is currently limited to specially tagged scene elements (tagging is performed in the VRML by MSG at the time the scene is generated) and the information returned is only that which has been specifically included in the scene description by the MSG server. This limitation is a direct result of the way that the scene is initially generated, and of using VRML as the transport format for the scene geometry. This and other limitations of the interface will be removed in future versions of MICE.

COLLABORATION

The collaborative capabilities of MICE represent perhaps the most significant difference between MICE and existing molecular structure viewers. Although there are many applications available for visualizing protein structures, there are very few that allow users to share their visualizations with other observers in real time. Another such application is Chimera (http://www.cgl.ucsf.edu/), which is based on the popular MidasPlus application from the Computer Graphics Laboratory at the University of California, San Francisco. Chimera is currently in the design and development stage, but is intended to build on the capabilities of MidasPlus to allow interactive and collaborative model building and manipulation. Although both the MICE and Chimera projects have the same basic goal of providing interactive collaborative molecular visualization, they aim to provide widely differing sets of features at very different levels. Chimera will provide not only visualization tools but will also allow very low-level interactions with a molecular model, even to the point of model creation and editing in a collaborative setting. Achieving this level of interactivity between many users and a single model is a complex task and perhaps inevitably leads to a large and equally complex application. Additionally, the implementation of Chimera and other such applications usually relies upon low-level communication and graphics toolkits, which are often platform specific and nonportable. By comparison, the current version of MICE aims to provide a somewhat shallower interaction with the molecular scene, limited to display rather than modification of a scene. The focus of development has been on portability, ease of use, and, perhaps most importantly for a web-based application, ease of deployment. Using Java and Java3D technologies for the user interface and 3D graphical portions of MICE, all of these criteria have been met elegantly. To maintain the same levels of portability and easy of use in a collaborative environment, development of MICE has proceeded in parallel with development of a generic and flexible collaboration service, which may be readily tied into MICE thanks to the modular design of the application.

Communication between MICE clients is mediated by the Interactive Collaborative Environment (ICE) server, which enables collaborative applications to synchronize state across multiple clients. In practice, this allows users to share information and control of that information between multiple clients anywhere on the Internet. In MICE, the ICE technology is a means of sharing representations of molecular structures so that changes to a shared scene that are made by one user are visible to all members of a given collaboration.

ICE ARCHITECTURE

The ICE architecture was designed to provide a general-purpose mechanism to add collaborative capabilities to many different kinds of applications. For the most part, the ICE system has no understanding of the data that is being shared through a collaborative session. This design allows the architecture of the ICE system to be kept simple while maintaining a high level of generality: the collaboration mechanism makes no assumptions about the type or structure of data to be shared between applications, and therefore does not impose any constraints on the design of applications requiring collaborative services.

The core component of the ICE service is the IceServer, which creates and manages the resources necessary to enable client applications to share and synchronize state. The IceServer is composed of three main layers of objects (shown in Figure 2), which work in unison to manage collaborations. The first "point of contact" for a collaborative client is the IceMaster object. The IceMaster object accepts connections from clients and is responsible for managing ICE sessions. Typically, when an ICE client wishes to initiate a collaborative exchange with other ICE clients, a new session is created by the IceMaster object. Clients attempting to join a session subsequently are then simply handed off to existing sessions, provided the requested session name matches that of an existing session. The IceSession object manages a collection of users in a given collaborative session, so that participants may be notified when new users join or leave the session. Finally, each IceSession object manages a list of IceChannel objects. An IceChannel acts like a telephone line, over which all exchanges of events and data in a session occur. An application may choose to open multiple channels so that network traffic traveling through one IceChannel does not interfere with or impede traffic within another IceChannel. This design enables multiple asynchronous events to be traveling between different components of an application without components being required to wait for other unrelated events to be processed elsewhere. As an example, the MICE application typically opens four IceChannel objects as soon as a session is opened. First MICE opens a "Data" channel, which it uses to inform other clients about the dataset that it is displaying, that is, the representation of the molecular structure. Secondly, the MICE client opens a "Camera" channel to allow synchronization of the 3D view with other clients. Third, a "Pointer" channel is opened, which is used to control a shared 3D pointer. Finally, MICE opens a special session manager channel, which it uses to coordinate its own internal state changes, including, for example, the list of users in the current session.

There is currently no notion of a "private" collaborative session, accessible to only a certain group of users. Currently, any session is available to any MICE client, without any access restrictions. There are plans to add support for password-protected sessions, whereby users attempting to join a private session would be challenged when attempting to connect and only admitted on receipt of the correct password.

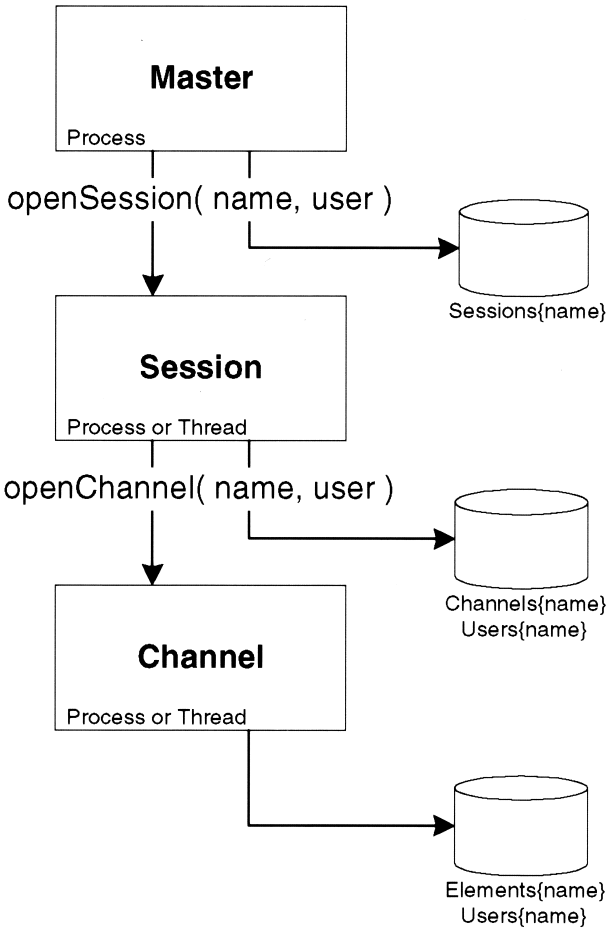


Figure 2. The architecture of the Interactive Collaborative Environment (ICE) service. When the collaboration service is started, a single IceMaster process is created to manage the service and accept connections from clients. When a client requests a new session an IceSession object is created, either as a new thread or an entirely new process, depending upon implementation. The current ICEserver uses individual threads within a single process. If a client wishes to join an existing session, the connection is simply handed off to the appropriate IceSession process and the user is added to that session. The IceSession object is responsible for managing users and channels within a given session, and creates new IceChannel objects as required. IceChannels are simple conduits through which clients exchange data. The ICE protocols allow for locking to be implemented at either the user- or the channel-level, but this feature is not used by MICE. When a session is ended, the IceChannel and IceSession objects are destroyed and their resources freed, but the IceMaster object remains active and continues to listen for new connections.

CORBA

All interactions between ICE clients and servers are facilitated by the Common Object Request Broker Architecture (CORBA). CORBA provides mechanisms for high-level language- and platform-independent connectivity between applications. As an example of the universality of the CORBA protocols, the ICE server is written in C++ and runs on Sun, SGI, and IBM server hardware. The MICE client, which interacts with the ICE service to manage collaborative sessions, currently runs on Sun, SGI, Linux, and Windows (95/98 and NT) platforms and will become available for additional platforms as Java and Java3D themselves become available.

A significant advantage of using CORBA as a communications layer is that configuration parameters that would normally define the relationship between client/server applications (such as host names and TCP/IP port numbers) do not need to be hard-wired into an application, but can be obtained instead from a third-party network service. This lends CORBA applications a degree of generality and, in the case of MICE, allows a client to initiate or join a collaborative session without the user having to specify any configuration parameters. Color Plate 3 shows the simple dialogue box used to create a new session or to connect to an existing one, and the session manager dialogue, which controls the publish/subscribe state for the current user.

On startup, MICE automatically makes a connection to a simple boot service (Figure 1), which is located at a fixed point on the network. The only purpose of the boot service is to return an Initial Object Reference (IOR), encapsulating the host name, port number, and other information pertaining to a CORBA Object Service (COS). This COS acts as a naming service, providing a network-based hierarchical object registration service for CORBA applications. Functioning in a similar way to the Internet's Domain Name Service (DNS), the COS allows CORBA-based applications to register their service and stores the network location of that service as an IOR. MICE clients register with the COS and are then able to obtain information about the available ICE servers on the network from the COS rather than from local configuration values. The net result of this transaction is that a MICE client can connect to any active ICE session with a single mouse click—the user is not required to supply TCP/IP numbers, port numbers, or host names for either the server or their client.

Firewalls are a commonly used network security mechanism, enabling a system administrator to simply block all access to a wide range of communications ports on protected machines, in an attempt to prevent unauthorized access to those machines by untrusted or unknown systems. Unfortunately, this type of security measure can cause significant problems for CORBA-based applications. Typically, firewalls leave only well-known and well-used ports open and accessible to the outside world. Ports that are commonly used for FTP or HTTP traffic, for example, are usually left open, with the remainder simply being blocked and made unavailable to any machine outside of the firewall. Because CORBA applications are never concerned with precisely which port or even which protocol they use for client-server communication, applications are simply assigned port numbers on-the-fly by the underlying operating system. If, as will usually be the case, a CORBA application like MICE is assigned a port that is blocked by a firewall, the application can only make outgoing connections, for example, to initiate a socket connection with a remote server. The inbound connections on the assigned port are blocked by the firewall, causing MICE to hang indefinitely as it waits for a reply to its connection request. It is important to realize that this problem is not caused by the ICE protocols or by the CORBA communications suite, but

is simply an unavoidable side effect of using a firewall to control access to certain systems or networks.

DISCUSSION

The MICE application is in the final stages of development and testing and has been used successfully in a variety of demonstration settings as well as in several real world situations. As a test of the collaborative capabilities of MICE, demonstrations have been run with some 30 clients connected to a single session. Other tests have shown that collaborations can be carried out across long distance and/or slow Internet connections, such as modems, without significant degradation of performance.

MICE has been successfully used in teaching a molecular modeling course at the University of California, San Diego, where 16 clients were simultaneously connected to a single session. The impact of MICE as a collaborative learning tool, as well as the modifications and enhancements to the client that will improve the collaborative learning experience, are still being evaluated. Several possible adaptations are under consideration, such as the creation of separate "teacher" and "student" clients with different sets of features. These versions of MICE could include provision for special permissions for teachers, allowing them to take control of the view of all clients, or the ability to create "movies" that highlight certain features of the molecule of interest.

The collaboration service is provided by the San Diego Supercomputer Center (SDSC), which is committed to supporting ICE as a full production service, ensuring continuous high-speed access for any applications requiring the ICE service. Although SDSC is the only site that is currently running an ICE server full time, the ICE architecture is scaleable and as demand grows additional ICE servers can be added at different sites across the Internet. Alternatively, sites requiring an ICE service for local use, for a lecture or taught lesson, for example, can run an ICE server on-site and ensure high performance access to the server for a specific group of clients. Similarly, although SDSC is committed to maintaining an MSG server for the generation of the VRML through the MICE client, the scripts required to run an MSG server and instructions for their use are all available from the MICE web site. Any user can install and run a local MSG server to reduce network load or scene download times.

Although the use of a remote server for scene generation is a convenient solution to many implementation problems, this scheme does have its problems. Clearly, although the reliance on a network-based server to create scenes significantly simplifies the MICE application, it also means that an Internet connection is always required to create scenes using MICE. Additionally, if the user makes a single small change to the appearance of a scene, the entire scene must be regenerated and the new VRML file must be shipped to the MICE client. In a collaborative setting, this new scene must also be distributed to all members of a session, making a simple change to the scene both time consuming and heavy on network resources.

Future plans for MICE call for scene generation capabilities to be added to the MICE client so that all changes to a scene can be performed locally and only information about the changes needs to be broadcast to members of the collaboration. This will also overcome the current limitations of using MolScript to generate scenes on a server, since the restrictions of having to convert user requests into MolScript scripts currently limit what choices a user can be given. Translating user requests into valid scripts is already a complex process and adding additional features to the MSG server is not a trivial task. When the MICE client has the ability to generate scene geometry internally from atomic coordinates, rather than via a server, it will be possible to provide users with much finer control over the contents and appearance of the scene, even down to the level of residues and atoms. Transferring responsibility for scene generation to the MICE client also overcomes the current problems with using VRML as the transport format for scenes. VRML is an entirely generic geometry description language, and as such it has no built-in mechanisms for describing molecular structures at the required level of detail. In the current version of MICE, a user can perform only the crudest interrogation of the scene, since the MSG server must explicitly add queriable information about the molecule to the VRML scene. Querying of the scene at the atomic level is therefore impossible, since the scene description never contains the coordinates of the constituent atoms. Once MICE is able to generate scene geometry directly from PDB data, it will necessarily have access to all of the atomic coordinates for a structure and will be able to query the atomic coordinates in response to a user request.

An intriguing notion that is currently being developed alongside the plans for the next generation of the MICE client, is the idea of a Molecular Scene Description Language (MSDL). As stated, a VRML file contains no domain-specific information about the scene that it describes (that is, data pertaining to a structure at the atomic or molecular level), but is simply a scene-graph based description of graphics primitives. A better approach is to define an MSDL that allows scenes to be described in terms of "molecular primitives," such as helices, β-strands, etc. We are currently drafting such an MSDL using eXtensible Markup Language (XML), a derivative of the Standard Generalized Markup Language (SGML) that uses tags to add context and meaning to text format data.

The MSDL is defined using a Document Type Definition (DTD), which specifies the possible tags that an author can use to describe a molecule. MSDL will allow authors to create scenes using most commonly used molecular graphics styles, such as those used in programs such as Rasmol, Chime, or MolScript. The intention is that MSDL will provide a common exchange format for scene information, which is both platform- and application-independent, so that a scene may be displayed using any molecular graphics program and still retain the same semantic content. Care is being taken to ensure that the structure of the language makes the creation of scenes easy for authors, while at the same time, display of the scene can be readily implemented by applications developers.

One of the advantages of using XML as the basis for MSDL is that SGML-based languages are immediately amenable to database archival query and retrieval. A major aim of MSDL is to facilitate the capture and search of molecular scenes from a wide variety of sources. By making scene deposition easy and worthwhile, a database of molecular scenes will be constructed, with the ultimate goal of allowing complex searches such as "give me all scenes depicting cAMP-dependent protein kinase that show the detailed interactions between ATP and the protein." Existing structure and sequence databases have provided a wealth of information about macromolecular structure and function through the comparison of multiple entries and the

observation of trends within the databases as a whole. In the same way, a repository of detailed three-dimensional descriptions of molecules will enable consideration of multiple structures on a visual level, literally adding an extra dimension to existing databases and web archives.

Perhaps the most important goal in the design and implementation of MICE is to maintain a modular architecture throughout the application. All of the major components of MICE, such as the user interface, the geometry viewer, and the collaboration management system, are implemented as separate, independent modules. In keeping with the paradigm of object-oriented programming, these components are inherently reusable and self-contained. Indeed, several components of the MICE application have already been used without modification in other projects at SDSC. The advantage of maintaining a modular design is clear: since additional components can be readily added without major reconstruction of the core of the application, new features and capabilities can be easily added to the application, as and when such modules become available.

The client-side MICE application is available from the MICE website (http://mice.sdsc.edu/) as a self-extracting archive for the Windows platform. MICE can also be supplied in a format more suitable for Unix platforms upon request, although the unmodified bytecode from the Windows archive can be run on any platform with a Java/Java3D implementation. Readers wishing to install an ICEserver should contact the corresponding author.

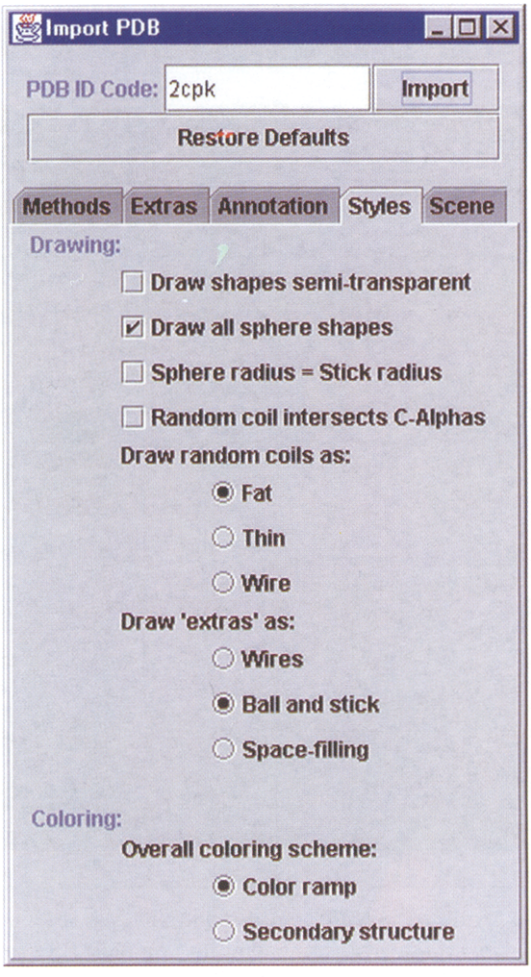
ACKNOWLEDGEMENTS

The authors would like to thank Dave Nadeau for his help with the 3D graphics component of MICE, Doug Greer for help with the design and implementation of the CORBA-based ICE communications protocols, and Helge Weissig and Dave Martinez for providing the wrapper scripts for MolScript that formed the basis for MSG. The MICE project is funded by grant DBI 9723475 from the National Science Foundation.

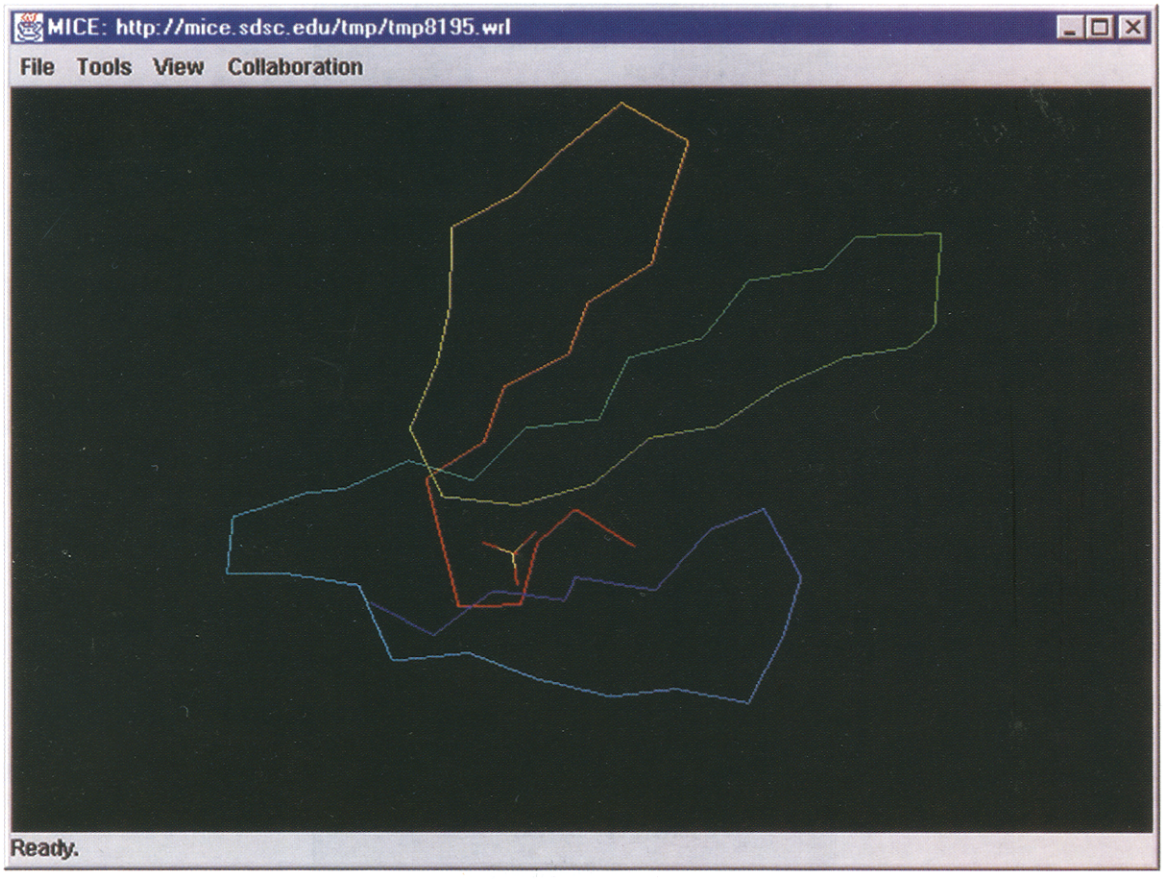
REFERENCES

1 Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E. The Protein Data Bank. *Nucleic Acids Research*.2000, **28**, 235–242
2 VRML Consortium, http://www.vrml.org/Specifications/VRML97/
3 Microsoft, Redmond, WA, http://www.microsoft.com/windows/netmeeting/
4 National Computational Science Alliance, Urbana-Champaign, IL, http://havefun.ncsa.uiuc.edu/habanero/
5 Tate, J.G., Moreland, J., Bourne, P.E. MSG (Molecular Scene Generator): a Web-based application for the visualization of macromolecular structures. *J. Appl. Cryst.* 1999, **32**, 1027–1028
6 Kraulis, P.J. MOLSCRIPT: a program to produce both detailed and schematic plots of protein structures. *J. Appl. Cryst.* 1991, **24** 946–950
7 Molecular Simulations Inc., San Diego, CA, http://www.msi.com/life/products/insight/
8 Carson, M. Ribbons 2.0. *J. Appl. Cryst.* 1991, **24**, 958–961
9 Sun Microsystems, Palo Alto, CA, http://www.javasoft.com/
10 Richardson, J.S. Schematic drawings of protein structures. *Methods Enzyml.* 1985, **115**, 359–380

Design and implementation of a collaborative molecular graphics environment



Color Plate 1. The "Import PDB . . ." dialogue from MICE, showing one of the five panels of customization options. The user chooses the molecule to be displayed by entering the PDB ID code in the entry box at the top, and then chooses the drawing style and scene attributes using the check boxes and radio buttons on the various panels. Pressing the "Import" button sends the CGI request to generate the required scene.



Color Plate 2. Several molecular scenes rendered in MICE. (a) Erabutoxin A, a sea snake venom (PDB ID code 5ebx), drawn as a simple wireframe backbone trace. (b) Major histocompatability complex B8 (1agd), including the bound HIV-1 GAG peptide (shown as a short section of red coil between the light blue and green helices at the front of the image), drawn using a schematic secondary structure representation in the style of Richardson[10]. (c) cAMP-dependent protein kinase (2cpk), again shown as secondary structure, but using cylinders to represent helices. (d) A close-up of a phosphorylation site on cAMP-dependent protein kinase, with a phosphate drawn as a ball-and-stick object, (e) A scene showing another section of cAMP-dependent protein kinase, with the collaborative pointer highlighting a feature of the protein. The pointer allows rudimentary interrogation of the scene by returning details of the last point that the user clicked (information window shown inset).

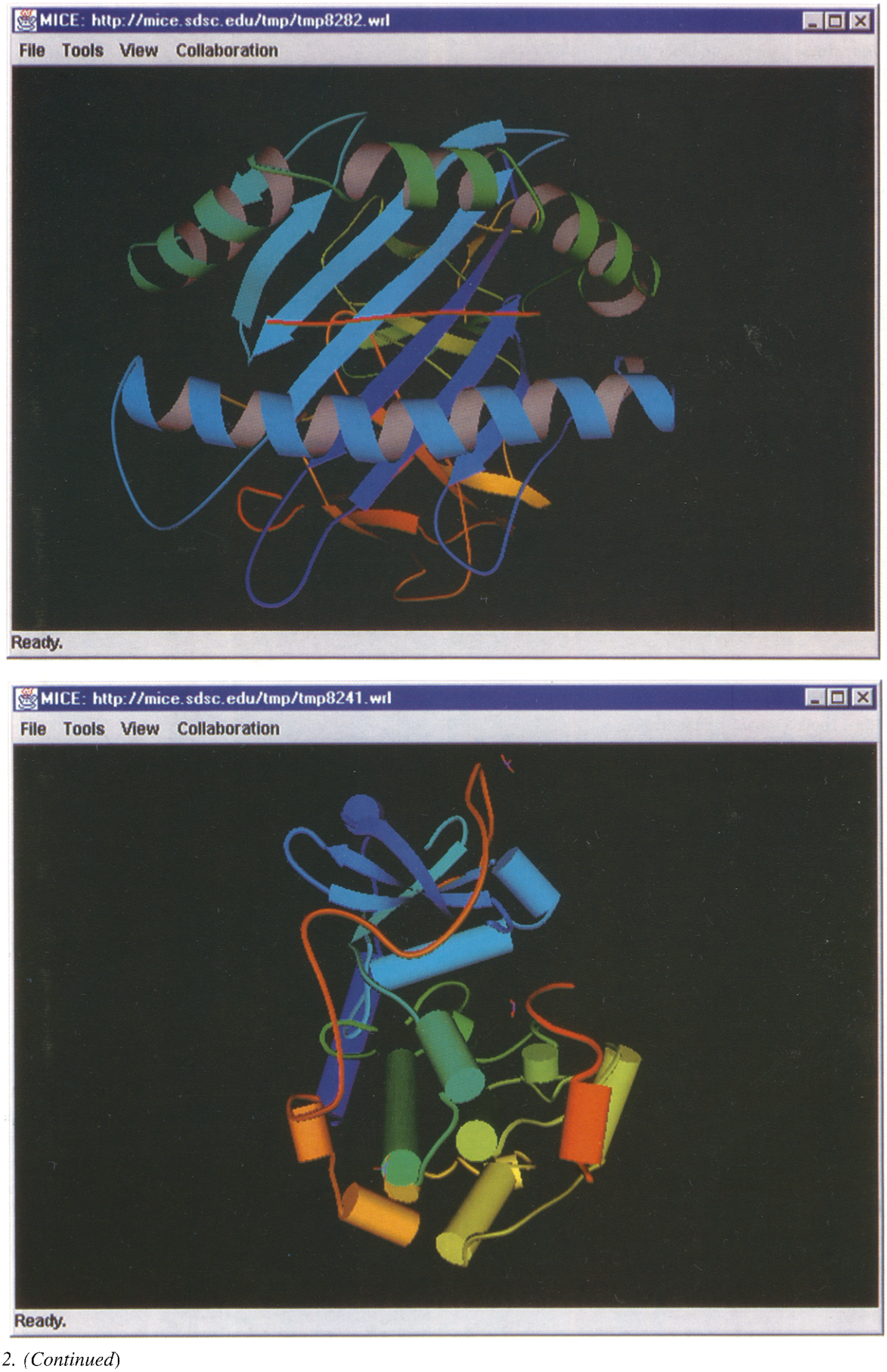


Figure 2. (Continued)

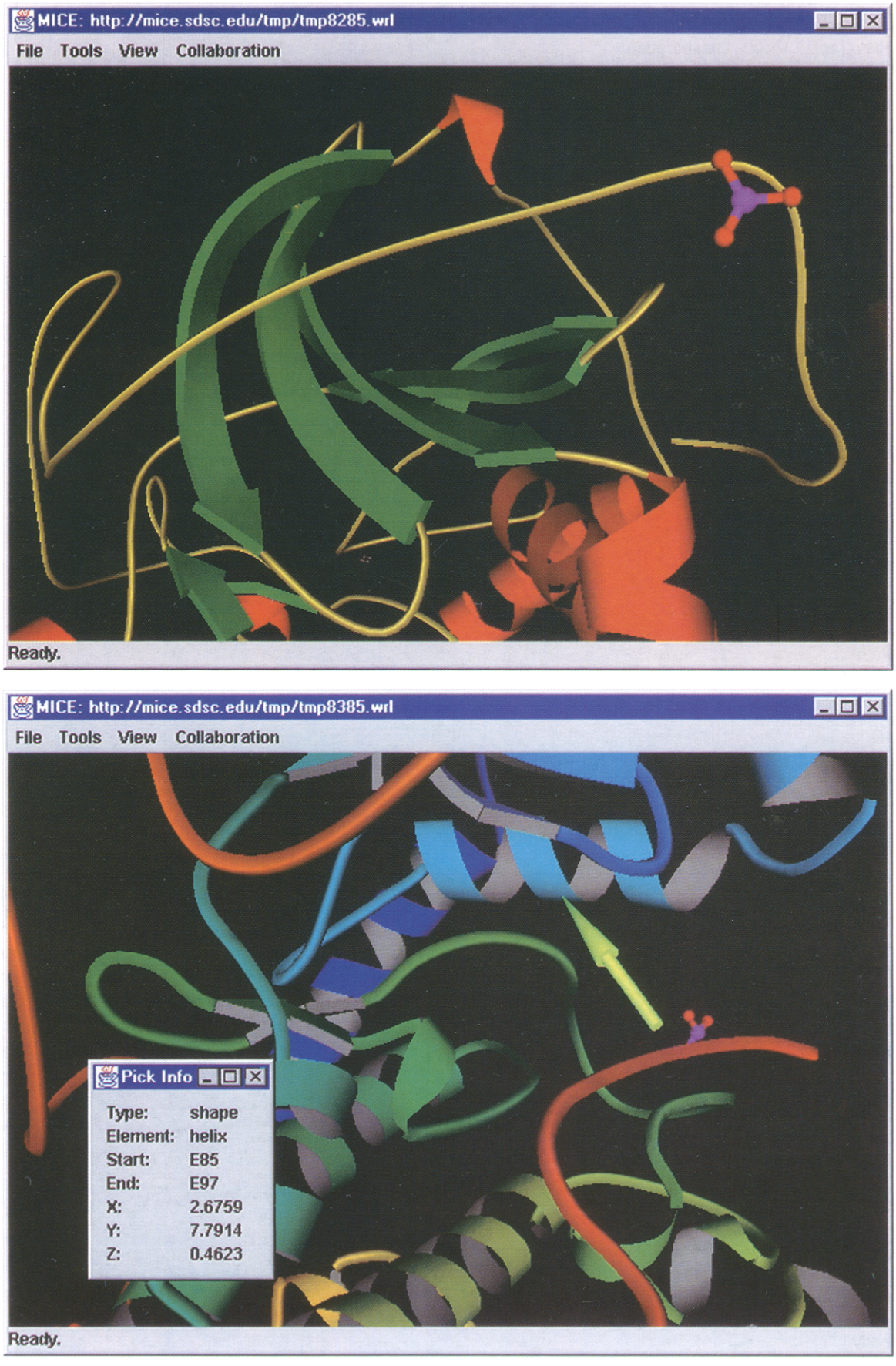
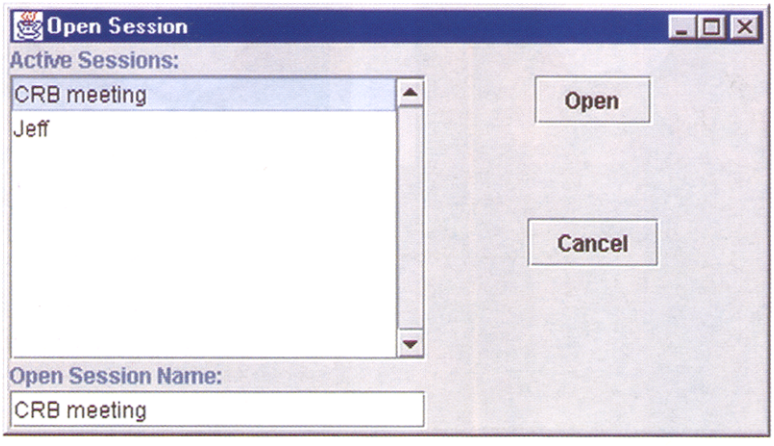
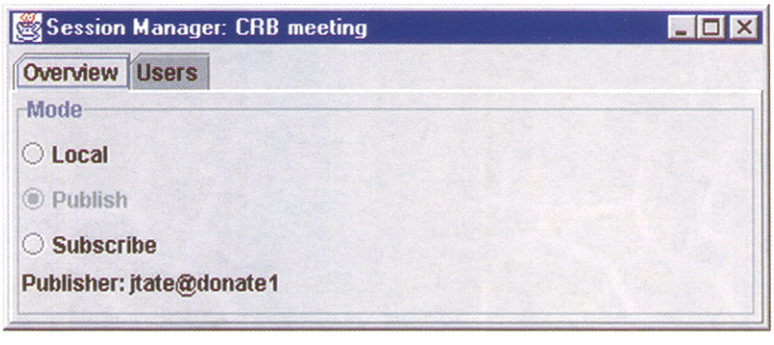


Figure 2. (Continued)



Color Plate 3(a). The MICE collaboration dialog. Users may join an existing session by choosing it from the list at the top, while new sessions may be created by entering a name in the text entry at the bottom of the dialogue.



Color Plate 3(b). The session manager dialog. Members of a collaboration may exchange control of the scene using the publish and subscribe buttons, typically while talking on the telephone or via video conferencing software to synchronize the event. To "opt out" of a session temporarily, the "local" button offers complete control of the scene in the user's own MICE client. Subscribing to the scene again will automatically resynchronize the scene with the collaborative view, including downloading any new scene content that was created while the user was not participating in the collaboration.