Astronomical Data Analysis Software and Systems IV
ASP Conference Series, Vol. 77, 1995
R. A. Shaw, H. E. Payne, and J. J. E. Hayes, eds.

The AstroVR Collaboratory, An On-line Multi-User Environment for Research in Astrophysics

D. Van Buren

Infrared Processing and Analysis Center, MS 100-22, Caltech, Pasadena, CA 91125

P. Curtis, D. A. Nichols

Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304

M. Brundage

Dept. of Mathematics, GN-50, Univ. Washington, Seattle, WA 98195

Abstract. We describe our experiment with an on-line collaborative environment where users share the execution of programs and communicate via audio, video, and typed text. Collaborative environments represent the next step in computer-mediated conferencing, combining powerful compute engines, data persistence, shared applications, and teleconferencing tools. As proof of concept, we have implemented a shared image analysis tool, allowing geographically distinct users to analyze FITS images together. We anticipate that AstroVR[1] and similar systems will become an important part of collaborative work in the next decade, with applications in remote observing, spacecraft operations, on-line meetings, and day-to-day research activities. The technology is generic and promises to find uses in business, medicine, government, and education.

1. Introduction

Collaborative research in astronomy is the norm. Most papers published in the Astrophysical Journal have multiple authors, and most multi-author papers involve collaborators at different physical locations. The reasons for this are many: the breadth of astronomy, the limited resources available to any one institution, and the social nature of the enterprise. In any case there is a qualitative difference in how a project proceeds when the collaborators are geographically distinct compared to when they are at the same place. Many of the differences are barriers: of communication, of access to data, of access to software, and of access to expertise. Researchers now cope with these barriers in a number of ways: telephone, e-mail, ftp, giving each other local computer accounts for access over the network, and more recently the creation of Web-based information services.

[1] http://astrovr.ipac.caltech.edu:8888

We were motivated several years ago by advances in network connectivity and multi-user database technologies to experiment with a new mode of remote collaboration. Our idea was to build a multi-user, network-accessible environment which we could populate with tools useful for astrophysics research. We thought such a system would be useful not only for distributed groups undertaking research projects, but also for teleconferencing, small seminars, browsing the astronomical portion of cyberspace, and providing a social space for participants. Eventually this technology could be hooked up to telescopes and other facilities; it could evolve in the direction of a "collaboratory," wherein geographic location becomes unimportant, and access to a wealth of services and tools is immediate.

During this interval a number of commercial vendors and NCSA have developed distributed whiteboard and teleconferencing applications, but these only superficially meet the needs of scientific workers. These rudimentary shared environments often require particular hardware, the purchase of proprietary software, have a limited toolset (NCSA Collage supports shared HDF data browsing and is probably the most useful) and are not extensible. Nor do they have persistence (i.e., maintenance of state between invocations), nor continuity of availability. We were interested in developing a more science-fiction style cyberspace where participants could interact with each other and the environment at any time, where changes to the environment could be persistent, and where new tools and facilities could easily be added, often synergistically extending the environment's potential.

At IPAC, Van Buren was monitoring the development of multi-user networked games called MUDs (Multi-User Dungeons). These client-server games were usually written by graduate and undergraduate students, and plain telnet was often the client. Generally they lacked the stability, support, extensibility, and/or persistence of data needed for a true collaborative environment, but they clearly were headed in the right direction as the idea of multi-user game servers became more and more sophisticated. One particular server, called MOO (for MUD Object-Oriented), written by Stephen White of the University of Waterloo, was chosen by Curtis to form the basis of his social virtual reality study at Xerox PARC, and he took over its further development. This server suddenly became professionally supported and maintained. In all other respects it was superior to the other MUD servers as well, so it became the choice for the underlying software for AstroVR. At the same time, the Xerox workers were thinking of extensions to the MOO technology that would support real-world applications. They initiated the Jupiter project, which is further described in Section 2.2. At this point we began a formal collaboration where AstroVR incorporates and tests the Jupiter technology, as well as prototyping new Jupiter extensions. AstroVR further extends the MOO and Jupiter technologies to include astronomically useful tools.

2. Architecture (with Examples)

2.1. The MOO Server

AstroVR's MOO server manages the many network connections and communications streams making up the environment. Contained in the server's database are objects, i.e., data structures containing code and information implementing the various capabilities. For example, one object generates "post-it" notes that users can pop onto each other's screens from a distance. Another object implements a shared Mosaic Web browser, and yet another allows a group of astronomers to join in the analysis of a single image, even though they may be sitting at workstations thousands of miles apart.

The basic architecture is client-server. In fact, the environment is heavily distributed, in that it makes use not only of the MOO server, but also auxiliary servers, remote information services, and a potentially distributed multi-process client. AstroVR also includes the entire public portion of the World-Wide-Web in its data space. The MOO server provides the persistent database and manipulative functions that manage the environment. It comprises a general purpose, multi-user, object-oriented database and an embedded C-like language: the MOO language. A "core" database, the LambdaCore, served as our starting point. This database included a general-purpose object-class library and sufficient code to further extend the environment. The MOO server and LambdaCore, as well as a programmer's manual[2] for the MOO language, are available in their most recent form via ftp[3].

The embedded MOO language makes AstroVR an extensible, evolving system. The behavior of all objects in the AstroVR database is defined by verbs (methods) and properties (data). Objects, verbs, and properties are created and altered from within the environment, making it easy to add and test new functionality without recompiling the server. An object attains its behavior in several ways. First, it inherits the behavior of its parent object, and the parent's parent, etc. Secondly, new verbs (possibly overriding inherited verbs) can be attached to objects. Thirdly, there is a large class of utility objects holding large libraries of general-use verbs. Finally, objects have data attached to them which represent their state, which of course affects their subsequent behavior.

The act of extending the functionality of existing objects and creating new objects in AstroVR is called "building." All users are potentially builders. Because the environment also facilitates communication between users, building becomes a collaborative effort itself. Many of the tools that exist are in fact building tools. In this sense, AstroVR is a meta-tool: it is used to create itself!

Building often takes the form of creating an instance of a previously existing object and then defining new behaviors by attaching verbs and properties to the new object. As a hypothetical example (the MOO server does not yet support floating-point math), consider how we might create an astrophysically interesting calculator, the H II region calculator. We want this specialized calculator to know how to figure Strömgren radii. Suppose that there is already a calculator object, `$calculator`, embodying the behavior of a general-purpose calculator. We create a child of `$calculator` to be our starting point because all of behavior of `$calculator` will be inherited. The new object is placed in our "namespace" and we operate on it, referring to it by a unique identifier. In particular, we next add a verb "r_stromgren" to the calculator which will do the work in calculating Stromgren radii, but which is initially empty. To create the verb code we invoke

[2] ftp://parcftp.parc.xerox.com/pub/MOO/ProgrammersManual.texinfo_toc.html

[3] ftp://parcftp.parc.xerox.com/pub/MOO

the AstroVR verb editor GUI and edit the verb code in a textedit widget. The textedit widget supports many **emacs** commands and is used in many places inside AstroVR for data entry. The MOO server side of the verb editor is implemented entirely in MOO code.

There is a sophisticated permissions system inside AstroVR, because in a multi-user environment not everyone should be able to do anything at any time. It turns out that writing secure verbs is not that difficult, but it is necessary to guarantee that objects behave properly. For example, suppose users Galileo and Copernicus were happily working out Strömgren radii with the H II region calculator. Then user Herschel, who is not participating in the collaboration, should not be able to punch numbers into the calculator, turn it off, or otherwise alter its state.

2.2. The Jupiter Client

The Jupiter user client is the same client that Xerox is using for its Jupiter project, where a virtual environment is being overlaid on the physical environment at PARC, using a MOO-based cyberspace. For most game MUDs, telnet is an (almost) adequate client, providing a single typescript where users issue commands and receive output from the server. But in a collaborative environment, where a number of tools may be in simultaneous use, a more sophisticated client is needed. The Jupiter client meets this need by supporting window interfaces to AstroVR objects; other capabilities include multicast audio and video support, transparent file transfer, and a generic interface for local applications to be shared with other AstroVR users.

The Jupiter window support is similar in architecture to the GUI layer for IRAF (Tody 1995), but with a slightly different widget set and window definition language. In a nutshell, the client builds and manages GUIs after receiving text strings from the MOO server which encode their character and behavior. Subsequently, only significant events (such as mouse clicks, but not mouse motions) initiate messages to be sent back to the MOO server. This approach represents a drastic reduction in network traffic compared to running X windows across the network.

Since the same network connection passes in one direction both plain text meant to be read by the user and requests for the client to do something, and in the other direction both plain text representing user commands to the server and client commands to the server to do something, we use an out-of-band (OOB) protocol. OOB messages across the network connection are `#$#`-escaped, newline-terminated strings. When sent to the server they are intercepted by a special object for further processing and dispatch. For example, the user may have changed the microphone gain using the volume slider on the main client window. In this case, the following OOB command is sent to the AstroVR server:

```
#$#win-event id: ''315'' widget: ''mgain'' value: ''27''
```

The OOB protocol in the client-to-server direction is defined by `#$#` followed immediately by a request type string, then a set of key and quoted value pairs. An extension allows an arbitrarily long list of newline-terminated strings to be sent. In the other direction, from server to client, there is an additional datum

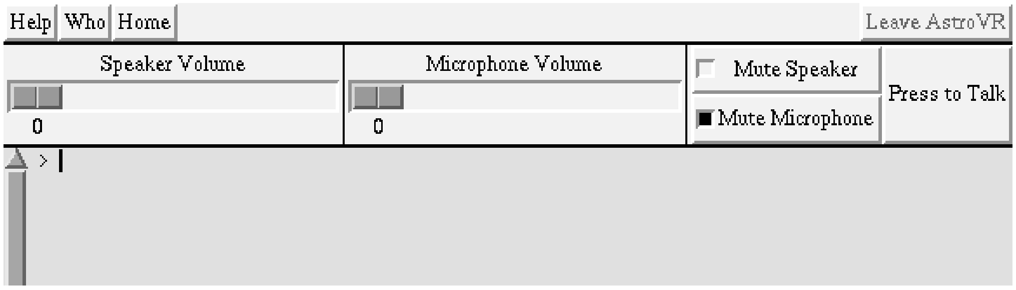

Figure 1. A particular client window. Buttons along the top are: *Help* to invoke the built in help browser, *Who* to pop up a constantly updating window that shows who else is on-line, *Home* to move to your default location, and *Leave AstroVR* to shut down your user object, client and network connection. Below are the audio gain and muting controls. The bottom panel, which is truncated for display, is the main input/output typescript where users issue AstroVR commands and receive messages.

required to provide some level of security. An authentication key known only to the trusted portion of the AstroVR server and client is included, and must be correct for the request to be carried out:

```
#$#audio-set-microphone-gain ''10565-258830'' value: ''27''
```

In this case the client is being requested to change the microphone gain. Note that the request to change the gain was made in the main client GUI, but the actual change was made in response to the server's notification to the client. In Figure 1 we show a screen dump of a particular AstroVR client GUI. It is only a particular one because users are free to redefine their client GUI's (and many other GUIs) according to taste. The user's ability to customize the interface is a key feature of our environment.

2.3. Auxiliary Servers and Remote Information Services

The MOO server is capable of opening arbitrary network connections. In principle this allows the integration of network based astronomical data services in AstroVR. We created simple interfaces to NED, SIMBAD and the STELAR abstract services to demonstrate the concept. Consequently, the data contained in these remote services is available for further computing by AstroVR. If the remote service's protocol consists only of newline terminated strings of printable characters, like the WAIS protocol for example, AstroVR can make the connection directly. Otherwise, as in the case of SIMBAD, a simple intermediate server is created as a stand-alone process that does the appropriate protocol translations.

One interesting auxiliary server is used to overcome the MOO server's current lack of floating-point support. We have the Unix calculator program **bc** running as a server that can be connected to directly using MOO code. AstroVR makes use of this by loading to the **bc** server a list of strings in the **bc** language representing the calculation to be performed, and then reading the results as a list of strings. Although the floating-point numbers are represented in AstroVR as strings, they are computed as reals. This service suffers some overhead per use, but a calculation proceeds quickly once the connection is made.

2.4. Multicast Support

Audio, video, and dynamic screen broadcast make use of the multicast network, often called the "mbone." This is a broadcast technology (one copy out) which sends packets to destinations in a way that eliminates redundancies and is very network friendly. The AstroVR server does not handle any of this data. Instead it computes and sends switching information to the Jupiter client, telling it to listen to a particular (or set of) mbone channel(s). For example, all AstroVR users in the same virtual conference room are all tuned to the same mbone channel. The mbone audio gives telephone-quality sound.

Microphones and speakers/earphones are inexpensive and readily available for practically all workstations, and allow users to make use of this one feature that goes the farthest in creating the impression of a virtual space. We also have the capability for reduced frame rate video conferencing, but the expense of a video board and camera currently restricts its use. Users without this hardware can still receive video. Users with video boards can also define an area or areas on their screens to broadcast to collaborators, allowing some sharing of data for which a multi-user interface is not otherwise available, or to show the output of a program that for security reasons a shared interface is unwise—for example a telescope control system.

2.5. Client-Side Shared Applications

A large class of stand-alone applications may be shared through AstroVR and its client. One can imagine two modes for sharing an application: in one mode a single copy of the application is executing, with fan-in of user commands and fan-out of outputs. This method requires a rather restrictive set of circumstances because of the large variety of platforms and operating systems. In the other mode each user sharing an application has a local copy which is synchronized via AstroVR messages with those of the other users. Both modes are supported, and we have implemented instances of each. There is a shared MONGO interface which requires one collaborator to have MONGO locally. The other members of the work group can issue commands, for example to overlay their own data points on a plot being collaboratively constructed. The shared application interface knows how to properly distribute the Tek4010 commands to display a plot on an X windowing device (though more generic X windows are not supported). What the user sees is an xterm for entering mongo commands and a Tek4010 window where the plot is constructed. The user commands are prepended by the user-name of the person issuing the command to maintain accountability.

The second mode of sharing, where multiple local copies are synchronized via AstroVR client requests issued by the server, is used for sharing the image browsing and analysis tool "SkyView." This tool was created originally for analysis of *IRAS* images and is quite powerful, yet has a fairly simple command set and grammar. Inside AstroVR we have built a shared interface that operates with the shared application client software to present a SkyView GUI. Even in single user mode, the GUI provides much added value to SkyView users: it allows the user to define macros and assign buttons to them, to manage separate image frames easily, and generally to program SkyView in arbitrary ways using MOO code attached to GUI callbacks.

A particularly useful shared application is NCSA Mosaic. AstroVR can start a local Mosaic for document viewing in single-user as well as synchronized/shared mode. At least in terms of data display, this gives AstroVR users a vast cyberspace, but with the ability to do computations on URLs, and so implement intelligent Web navigators. (In fact, we had implemented many of Mosaic's transparent fetch and display functions inside AstroVR and its client, but the advent of the NCSA software means we can switch over and inherit all their updates and be assured of a well maintained package.)

An issue that arises when sharing applications is exactly how are they to be shared. One possibility is that commands are executed in the order they are typed, no matter which participant does so. This can be thought of as a first-come, first-served model, and is adequate for small (1–4 person) groups working collaboratively as equals. When a large group is sharing an application, this can break down and the users might want a different sharing model to be implemented. An extreme model is leader-follower, where only a single user is allowed to execute commands, and any other synchronized processes duly follow suit. In this mode, commands issued by the other users are ignored. This model is useful for doing software demos or when giving a seminar or on-line lecture. One of the strengths of the AstroVR system is that both models are readily implemented, and anything in between the two. We will discuss these models a little more in the section on teleconferencing.

Finally, we show in Figure 2 the overall architecture of the AstroVR system: server, clients, auxiliary and remote services, client-side processes, and the multicast network.

3. A Sought-for Richness

The goal of building is to create a large enough set of objects to be useful for conducting collaborative research. The technologies upon which AstroVR is based allow and encourage users to contribute to defining and building this toolset in a collaborative fashion. Our hope is that AstroVR users themselves will generate much of the environment's richness and functionality. But even a small toolset allows interesting combinations and we fully expect that a much larger set will allow for extremely powerful synergisms. Our thinking is based on the experience leading to the ISSA Postage Stamp Server[4], described elsewhere in this volume by Van Buren, Ebert, & Egret (1995).

Almost two years ago a simple text interface was built in AstroVR for SIMBAD; at the same time we were experimenting with transparent AstroVR-mediated file transfer, and separately providing on-line access at IPAC to all the ISSA[5] data. These ideas came together when we constructed a "virtual *IRAS* satellite" inside AstroVR that would deliver to users small pieces of the infrared

[4]http://astrovr.ipac.caltech.edu:8888/ISSA-PS

[5]http://www.ipac.caltech.edu/ipac/iras/issa.html

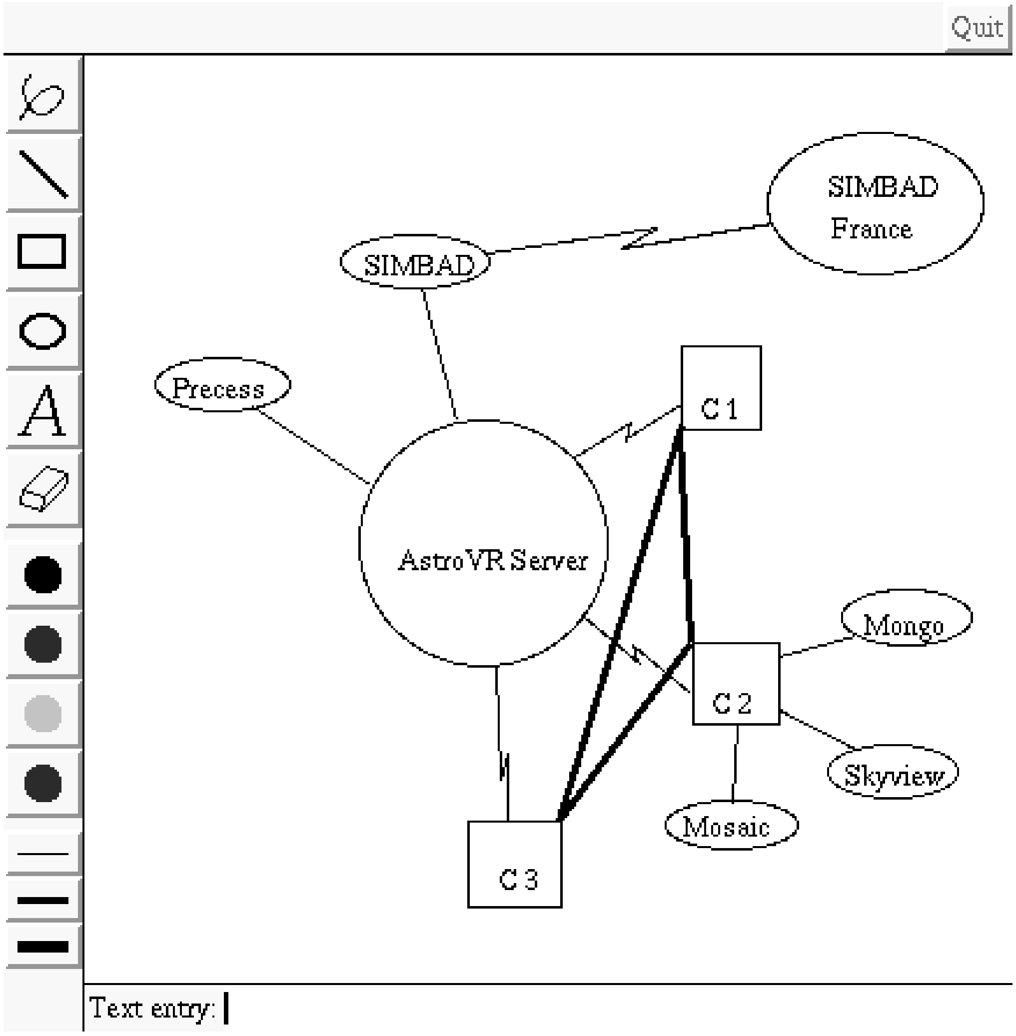

Figure 2. The AstroVR architecture, composed and displayed using the AstroVR shared whiteboard application. One connects to the main server via user clients (C1, etc.), which in turn run local shared applications and tune in to the multicast network (*heavy lines*). The AstroVR server also makes connections to other local and remote services, perhaps through intermediate protocol-translating servers (e.g., SIMBAD). The whiteboard itself can be shared by an arbitrary number of users. Commands are serviced on a first-come, first-served basis. The buttons along the left affect the drawing mode: *curlique* for free-hand drawing with spline smoothing; *line* to draw a line between two points given by mouse clicks; *box* to draw a rectangle; *oval* to draw an ellipse; the letter "*A*" to write some text; *parallelopiped* to erase; four *dots* to specify drawing color as black, red, green or blue; and three line-width selectors. An arbitrarily large number of whiteboards may be active at any one time. Whiteboard data is also easily saved and restored, giving an archive capability.

sky identified by the name of a target or celestial coordinate. In the meantime the idea of the World Wide Web exploded, leading us to build a general-purpose Web server inside AstroVR to provide Web access to portions of the environment. The virtual *IRAS* satellite was rewritten to make use of this interface, and was recast as the ISSA Postage Stamp Server. Inside AstroVR users now have new tools to further manipulate this service. One drawback of using NCSA "Mosaic" to access the postage stamps is that it requires repetitive human input to get data for a number of objects (unless one wants to write Mosaic drivers). The ISSA Survey Engine was therefore created. It takes a list of target names or positions edited with the built-in AstroVR text editor and delivers all the FITS files to the user's local system for further study. The lesson learned from this exercise was that tools can be combined in new ways that are very powerful. In the past, large resources at IPAC were spent delivering equivalent data to users that is now automatically provided at miniscule expense.

4. Teleconferencing

The technology of teleconferencing is still in relative infancy. Most people have experience with telephone conference calls. For small groups this can be an effective mode of communication since we are all fairly well socialized with correct phone behavior. But with larger groups, the ambiguity of meaning, confusion of speaking order, missing facial expressions, body language and other signals that help facilitate a large conversation reduce the effectiveness of conference calls drastically.

4.1. Floor Control

Floor control can be improved using in-server software to manage audio and other data streams. Each user has as part of the client an audio "receive" channel and possibly a different audio "send" channel. For example, each distinct location (or "room" in the VR metaphor) inside AstroVR normally has a unique send/receive broadcast channel. As a user changes location, her audio channel changes transparently so she can participate in whatever audio activities are taking place in the new location. Other audio behaviors are possible because the "send" and "receive" channels do not have to be the same. In lecture mode, all the users' "receive" channels are set to the lecturer's "send" channel, but everyone else's "send" channels are unconnected, so their audio data are not broadcast to the group. In "talking stone" mode, an object is passed from user to user, essentially giving them temporary lecturer status. Or participants could register an interest to speak with a meeting-chair object, and then talk when their turn comes up in a first-come, first-served fashion. Another mode is where a meeting facilitator services speaking requests with the assistance of a GUI that keeps track of outstanding requests. Arbitrary floor control methodologies are possible, so depending on circumstances the best one available can be chosen for any given purpose.

One situation where floor control becomes very important is sequence planning for flight projects. The science team typically has a phone teleconference with a dozen or so participants. Each member rightfully advocates a particular course of action and all must be reconciled. Without proper floor control such meetings are inefficient, unsatisfying and sometimes the results are incorrect due to errors arising from attention lapses, confusion or uncorrected misunderstandings. Facilitator mode conferencing, especially if augmented with minute-taking software (possibly working with audio records), shows great promise in increasing the effectiveness of such meetings.

4.2. Human vs. Virtual Presence

For the forseeable future computer-mediated conferencing and on-line collaborative environments will not take the place of actually being with someone. There is no substitute for seeing a person's body language, feeling a casual touch or sharing a meal. These kinds of social interaction are crucial to the quality of many professional relationships and cannot be recreated in a virtual environment. On the other hand, much effective work can be undertaken on-line between visits, and if the environment is easy enough to use, the work can proceed as any other work, unconstrained by geography.

5. A Rosy Future for Collaborative Environments?

This past summer the National Institute for Standards and Technology issued a call for proposals to develop ideas for distributed multi-user software technologies in the field of manufacturing, explicitly targeting MUD and derivative technologies. AT&T television advertisements feature multi-user, on-line environments as what we can expect in the future. As network bandwidth increases and the computing power of desktop machines does likewise, the technical ability to create an immersive on-line environment will lead to the creation of multi-user virtual spaces in many disciplines. Some obvious applications include medicine, where consultations with distant physicians will be possible; education, where classes and seminars can be held in fields that are too small to support a local effort; government, where many meetings will no longer require travel; and business, where far-flung operations can keep in touch and conduct business in a virtual environment. The technology is a humanizing technology, enabling people to come together for work or play from all over the world, to develop new connections and to discover new ideas.

Acknowledgments. We acknowledge the support of US taxpayers through a contract to the Jet Propulsion Laboratory, California Institute of Technology from the National Aeronautics and Space Administration. The original author of the MOO server is Stephen White. The LambdaCore software was written as a collaborative effort by a large group of pioneering MOOers.

References

Van Buren, D., Ebert, R., & Egret, D. 1995, this volume, p. **??**

Tody, D. 1995, this volume, p. **??**