"Living Worlds"
Concepts and Context

URL: `http://www.livingworlds.com/draft_1/lw_ideas.htm`
Last Update: 23 Feb 1997 14:30 PST [by Mitra]

This document describes the *Living Worlds* initiative: where it comes from, the problems it addresses, the process it adopted, and the conceptual model that is its first result.

---

1. Context

Back in 1994 the infant VRML community divided its long-term goal - establishing standards for implementing cyberspace - into three sequential tasks:

1. Defining *appearances*: how things in cyberspace will *look*
2. Defining *behavior*: how to make things *move*
3. Defining *distribution*: how moving worlds will be *shared*

Roughly speaking (i.e., ignoring a lot of overlap and the fact that the work has mostly gone into hammering out the thousand unforeseen details involved in creating consistent compromises from multiple threads of intersecting input), these tasks defined the focus of three successive major VRML releases.

VRML 1.0 was achieved surprisingly quickly, by agreeing to build on a pre-existing ASCII file format for the descriptions of virtual object clusters, and to use HTTP to access such files across the Internet. VRML 2.0 proved harder to achieve, since there was suddenly an install base to consider. The designers of VRML 2.0 needed to reconcile two somewhat different sets of requirements:

- to *refine VRML 1.0* based on the reports and recommendations of the pioneer content developers, and
- to *enable new kinds of content*, in particular sound and other data-streams, animation and other time-constrained interactions.

VRML 2.0 was introduced at Siggraph '96. VRML 2.0 enables designers to create an enormous range of applications from chemical or biomedical modeling to information visualization, and manufacturing design. It has now, in its turn, become the basis for a new two-pronged design process:

- *Content developers* have already begun to put the new VRML 2.0 animation and extensibility features to work, testing whether their semantics are adequate to define virtual worlds which can satisfy paying customers;

- *VRML architects* are equally busy, analyzing the requirements and articulating the design trade-offs involved in coordinating interactions among multiple human and mechanical participants in distributed virtual scenes.

Even before VRML 2.0 was formally launched, a number of new working groups had emerged to push this dual process forward. One of these, working under the name *Living Worlds (LW),* brings together a small group of developers who have already implemented proprietary systems for inserting dynamic objects -- human-controlled avatars and autonomous bots -- into VRML 1.0 scenes and distributing the results in real time over consumer-grade Internet hookups. The goal of the LW group is to distill the experience gained from these systems into a proposal for *a VRML 2.0 application framework* that would enable VRML developers to populate and share their "Moving Worlds."

This document reports some initial results achieved by the LW working group, and proposes how what they have begun to be used as a foundation for a broader community effort.

Living Worlds

The charter of the LW group is to distill its experience with avatar-based interaction in VRML 1.0 into a proposed standard for distributed object interaction in VRML 2.0. The LW effort aims at a single, narrow, well-defined goal: to define a set of VRML 2.0 conventions that support applications which are both *interpersonal* and *interoperable*.

By *interpersonal*, we mean VRML applications which support the virtual presence of many people in a single scene at the same time: people who can interact both with objects in the scene and with each other. By *interoperable*, we mean that such applications can be assembled from libraries of components developed independently by multiple suppliers, and visited by client systems which have nothing more in common than their adherence to the VRML 2.0 standard.

Draft 1.0 of of the LW specification, dated 24 Feb 97, includes interfaces for:

- coordinating the position and state of shared objects (including avatars),
- information exchange between objects in a scene
- personal and system security in VRML applications,
- a (small) library of utilities, and some workarounds for VRML 2.0 limitations
- identifying and integrating at run-time interaction capabilities implemented outside of VRML and its scripts.

These fundamental interfaces are, we believe, both necessary and sufficient to build a first generation of shared VRML worlds.

Interoperability is a requirement posed by all participants in shared worlds:

1. *Users* want to be able to take their Avatars, and their favorite interactive objects, with them to different worlds.

2. *Component developers* want to focus their efforts on building single components (e.g. avatars or portable multi-purpose components such as dice or playing cards), for which there will be a larger market, as more and more worlds become interoperable

3. *Application developers* (i.e. the authors of worlds and complete applications such as the board game) can combine best-of-breed components to yield richer results with less effort.

At the same time, it is crucial not to overconstrain component developers by trying too early for *too fine* a level of interoperability. Environments are only conducive to technical innovation when developers have enough space between the standard interfaces to make things they can sell at a profit. Thus it is crucial to try to match component standardization with a sense of the maturity of the market.

Principles

Living Worlds is governed by a fundamental commitment to VRML and to the process whereby it continues to evolve. The *content* of the Living Worlds proposal is thus governed by three working principles:

1. ***Build on VRML 2.0.*** It is always tempting to imagine ideal systems. It is more of a challenge to apply existing capabilities to achieve fundamentally new results. Living Worlds is not (yet) much concerned with what ought to go in VRML 2.x or 3.0. It is built entirely with existing VRML 2.0 mechanisms, defining as "implementation specific" anything which cannot be implemented inside the current standard.

2. ***Standards, not designs.*** The task of a standard is to simplify the integration of independently developed components. No standard can ensure good design. Standards which overconstrain solutions stifle innovation. The focus of the LW effort is:
    - to identify the *minimum set of system features* required to enable shared environments, and
    - to define the *minimum set of standard interfaces* required to enable application developers to combine feature-sets from multiple suppliers.

3. ***Architectural Agnosticism.*** As it happens, all of the LW initiators have based their solutions on architectures that exploit a central server. But enabling innovation requires that the standard not prejudge the question of how functionality is physically distributed.

LW is equally strongly, committed to *success in the marketplace.* That commitment leads to two further principles governing the Living Worlds *process*:

4. ***Require running code.*** The initiators of LW have all implemented interpersonal VRML environments, many for commercial use. The challenge is to distill that implementation experience into a framework for interoperable components. We believe that a proposed interoperability standard can only be evaluated on the basis of multiple interoperable implementations. The development of a final Proposal for VRML 2.x requires a body of real-world application experience.

5. ***Respect the role of the market.*** The goal is *not* to design the best imaginable system, but to give component and application developers a way to combine their best efforts: so that more VRML worlds will get built faster, and with them the base of user experience from which we will eventually learn what this technology is good *for*. Time-to-market is of the essence in this process. Our current goal is to complete a Draft 1.0 specification in time for the VRML'97 conference in February, and to have a first set of interoperable implementations of Draft 1.0 features sometime in mid-1997.

Anyone who accepts these principles is welcome to participate in the process.

Community Process

Work on LW has followed the model used successfully for VRML 2.0: a small group of editors drafts a document based on extensive implementation experience, and then publishes the result for community review. The forum for community review of the Living Worlds documentation is the LivingWorlds news host, at:
news://news.livingworlds.com

The current draft 1.0 constitutes our proposed "charter" for a VRML Consortium Working Group on multi-user infrastructure, which we expect to see constituted at the VRML '97 conference , Feb 24-27, 1997. In line with Princples #4 and #5 above, we hope to see prototype implementations presented to the community as soon as possible, as users and developers begin to test the viability of the proposed interfaces, channeling their results into the established VRML standardization process.

This *Living Worlds* Library Specification proposes standard interfaces for each of the basic kinds of functionality discussed so far. The Architecture Guide explains how the separate nodes specified in the Library form a coherent conceptual whole. The Worldbuilders' Guide provides a basic introduction to the process of using the Living WOrlds library to build shared virtual environments.

At the VRML '97 conference in Monterey, 24-26 Feb 1997, the Living Worlds initiators will propose the creation of an official Working Group under the rules of the new VRML

Consortium, with these documents as the Charter for that group. The purpose of the group will be to review, refine and enhance these proposals, especially in the light of experience with the first prototype implementations. Our goal is to reach quick consensus on a minimal subset, and then to encourage the rapid creation of several reference implementations of that proposed feature set. Refinement of the concepts can then proceed in an atmosphere usefully disciplined by implementation experience. The eventual target is to produce a Draft 2 of this document set, which will include proposals for specific enhancements to the VRML 2.0 standard.

2. Requirements

As a way of anchoring the technical discussion in some recognizably real-life applications, consider the following scenarios:

Scenario #1: an evening at home

- *Art sits alone "at home" in his recently redecorated virtual living room. (Actually, Art is dialing in from a hotel in New York to an ISP which supports Art's virtual home from a server in San Francisco). The new "etchings" on the living room walls are courtesy of a museum in London, whose subscription service automatically updates Art's art every Friday afternoon.*

This scene is not (yet) multi-user, but it is already multi-developer. Note how the distinction between "scene authoring" and "scene use" is blurred when the scene may be composed dynamically from different sources.

- *There is a knock at the door: Betty and Chuck have come by for a visit. Sitting at her desk in Chicago, Betty clicks her mouse, knocking on Art's door.*

How does Art's laptop in New York "hear" Betty "knock" from Boston on a door that is "really" in San Francisco? When Art opens his door, he sees two figures outside; they also see him, and over his shoulder, his living room; the sound of his stereo, previously muffled (for them - Art could hear just fine), is now much clearer. How does all that happen?

- *The figure that both Art and Betty see as "Chuck" was bought off-the-shelf at Avatars 'R' Us. Betty has a custom avatar which she built herself, using the new Acme Avatar Builder. On her shoulder is something yellow: her new "birdbot" which suddenly takes flight, whirling around Art's living room, singing merrily and dropping the occasional ... feather.*

Avatars are independently designed objects inserted into a scene. This further undermines the author/user distinction, and raises any number of data-integrity issues. How does this insertion happen, and under whose control?

- *An animated discussion ensues (in at least two senses). While Art and Betty type chat-style messages to one another, Chuck makes faces and maintains a constant verbal voiceover on their spelling and style -- comments only Art can hear, since Betty's machine has no speakers.*

How does one system find out whether another has voice capability? How do the necessary synchronizations (e.g. changing an avatar's expression to match an emoticon, merging Chuck's voice with the background music) get handled?

- *To clarify a point, Art draws a figure on his virtual whiteboard. Both Betty and Chuck can see what he draws, but Betty can also edit the drawing, since her PC has the special pen-pad hardware required by the whiteboard application.*

The whiteboard, separately purchased, is now an integrated part of Art's living room. It depends on aspects of Art's system about which the designer of the living room (a virtual interior decorator in Berlin) neither knew nor cared. Art, on the other hand, does not know in advance whether a given visitor will have a system able to use the full range of the whiteboard's capabilities. So how and when and where do the several systems involved discover and hook up the capabilities they share?

- *Chuck has brought his "Virtual Monopoly" set, which uses the new Zapp-O Digital Dice, certified by VeriPlay (an independent testing organization) to produce a statistically random "roll."*

A different kind of external reference here: a re-usable, in fact portable, object guaranteed by a third party to fulfill a certain specification. Other examples suggest themselves: a deck of cards certified to shuffle thoroughly and always deal from the top; a cash register which guarantees secure and anonymous transactions. This kind of 3rd party functionality could conceivably override or enhance or constrain elements that are pre-defined in the scene (e.g. by putting a lock on a door, or making an opaque wall suddenly transparent)...

- *The game has two different modes of play: in addition to the familiar board-and-pieces mode, which prevents players from moving out of turn and always moves them the right number of steps in the right direction* [Remember: the dice, the counters and the game were all developed independently], *there is also an "immersive" mode, in which the players shrink down so that the houses and hotels on the board seem life-size.*

How does the browser know how much to shrink the avatars, and how to adjust their location and speed of motion? Alternatively, if the "shrinkage" is realized by simply transporting the players into a different scene, how are the actions in that scene replicated on the board in Art's living room?

- *Two days after the memorable Monopoly game, Art discovers that the birdbot has left him a gift: a shiny blue egg with bright yellow spots.* [Recall: Betty's avatar

brought the birdbot into the scene; this imported object then created a new object which remained in the scene after its (uninvited and unmanaged) creator left.] *When clicked, the egg starts to tremble, then rock, and then crack open -- hatching a whole army of sandaled warriors with crested helmets, who bellow threats and insults in ancient greek as they swing their battle-axes, knocking all the pictures off the walls.*

In virtual reality, the term "Trojan horse" can really come into its own... Multi-user virtual worlds, like any network application, will require a full range of reliable mechanisms for protecting content on both clients and servers from inappropriate access and manipulation.

This scenario suggests some of the complexity of the "distribution" that must be managed to make immersive social applications possible. If this example seems too far-fetched, here is more sedate but no less challenging scenario: this one based on a application already in commercial use.

Scenario #2: a virtual Trade Show

- *In an effort to increase the attractiveness of its flagship trade show, SoftBank decides to add a VR auxiliary to the Spring Comdex.*

Trade shows are a complex community of interlocking interests: vendors, visitors, producers, contractors. All contribute both transient and persistent objects to the scene, which may have different and not wholly consistent priorities and constraints as regards world size, transparency, familiarity, security, etc.

- *The show booths and their robot/avatar sales staff are authored separately, as are the wandering "Advertars" who represent various show sponsors.*

This last concept opens up both a large market and a complex set of problems. Communities, even (or perhaps especially) such short-lived ones as trade fairs, exist as structures of *communication*. VRML 1&2 are only concerned with how things look and move. But implementing Cyberspace means mastering how things - especially things that represent or are controlled by *people* - interact with each other, and to what effect.

- *All visitors get Registration cards, which all booths can read, hooking their data into a show-wide "lead tracking" database.*

Here establishing a trustworthy identity is an entrance requirement. But note the other side of the coin: who decides what information goes into that lead-tracking data base?

- *The exhibition includes a Conference: people give presentations, entertain questions, take part in panels and surveys.*

What is the VRML equivalent of a lecture hall? How does an author arrange for a much bigger audience to see, hear and respond to a presentation than can ever be rendered on most desktop machines?

- *This VR show was distributed 2 weeks before the real show, at which real-world visitors are given an update-CD with more/cooler sites, new games, different ads.*

The new world needs to be able to merge seamlessly with its predecessor (not a new problem in the software world, of course, but one that takes on a new character when the application is evolving dynamically as it is used).

Cyberspace Infrastructure: Technical Targets

The sort of shared VRML environments described in these scenarios will not be designed as single structures, but assembled *ad hoc* as they are used. They will not be constructed solely in VRML, but invoke scripts an external applications in a variety of languages. They will not be under any single operator's control, but will grow ad hoc, as individuals and small groups begin to visit each other's private VRMLsites and to plug them together.

In sum, they will be based on continual negotiations among independently developed software objects, many of which will encounter each other for the first time at run-time. Some of these will invoke input from multiple, more or less arbitrarily interacting application programs (e.g. the whiteboard, the dice, the voice-chat package). Other objects will be "driven" by still less predictable humans.

Clearly, the individual authors of local living rooms or trade-show booths cannot be expected to cope with this level of complexity. The evolution of cyberspace, like that of any other modern human living space, depends on the establishment of a reliable *infrastructure:* facilities that provide commonly accessible solutions to the basic problems of transport and communication, traffic regulation, commercial transactions and property protection. Sifting our scenarios for infrastructure requirements, we can identify two basic sets of infrastructure functionality: *scene-sharing* and s*ecurity*:

1. **"Scene-sharing"** - *the coordination of events and actions across the net.* This is what enables multiple clients to interact: the mechanisms for letting each other know that someone has arrived, departed, sent a message or changed something in the scene. The most basic functions are:

    - **Insert/Delete Objects in a Scene at runtime**
      How else can an Avatar make its entrance or exit? Or a game suddenly produce the RoboCop who takes you directly to jail? In a single-user world, this can be handled by existing VRML 2.0 scripting mechanisms; in Living Worlds, the scene graph on every client must in principle be dynamically modifiable by any other client.

- **Track and communicate the state and behavior of objects in real time**
  The challenge is to let user A see the result of whatever user B is doing (moving furniture, dodging dive-bombing birdbots). As soon as we get past C and begin thinking about virtual crowds, the question "*Who* needs to know *what how often?*" becomes increasingly critical to insuring overall system performance, given that both network bandwidth and client processing power are always in limited supply. Keeping everyone as up-to-date as their equipment allows becomes a subtle problem in dynamic data distribution.

- **Coordinate local and external object "drivers"**
  This extends "scene-sharing" to include not just multiple users but also multiple developers and non-VRML technologies. On the one hand, it is the basic enabling mechanism for all *personal* interaction, i.e. avatar motion and behavior; on the other, it is the basis for all *component-based* applications, from the use of electronic cash to the database access that validates your right to enter the Dragon's Lair.

- **Support information exchange among Objects,** from simple message strings and multi-media data arrays (e.g. a business card with your picture on it) to essentially arbitrary data streams and containers (e.g. a file with your resume in it, or your company's complete set of 1997 product maintenance manuals).

- **Let imported objects become persistent,** i.e. make them a permanent part of the Scene. This is what makes it possible for Art's art to be updated, or Betty's birdbot to lay an egg that doesn't vanish when Betty leaves the scene. This issue is easily overlooked, but neither technically trivial nor possible to postpone, raising as it does the twin issues of permanent storage and access control. If you can make permanent changes to any scene you enter, then you become *de facto* one of its authors, with all the security issues that implies. If you cannot, i.e. if you can't make anything *happen* in an online world, and how long are you going to find that interesting? And if we can all potentially alter the worlds we visit together, then how and where are those changes recorded and communicated, and under whose control? **[*Note:* *it seems clear that implementing object persistence in shared VRML worlds will require* interfaces to external data base management systems -- *the focus of a separate Working Group with which close coordination will be needed.*]**

The Living Worlds authors have all developed proprietary solutions to most of the above challenges. This proposal is a first attempt to devise a common VRML 2.0 interface to support basic social interaction in multi-user virtual scenes: a public infrastructure for cyberspace. But bringing people together also means creating opportunities for conflict, both inadvertent and deliberate. To enable effective

Sharing, we must simultaneously provide effective ...

1. **Security** - Given the possibility of multi-user interaction in shared scenes, it takes little imagination to picture the potential for problems caused by such interactions. Faith in humanity (or in the harmlessness of pretty pets carried in by friends) is a fine thing, but it's still a good idea to cut the cards, and to carry a big insurance policy. We can group the functional requirements of any security system under the basic headings *Concurrency Control, Coditional Access*, and *Authentication*:

    - **Concurrency Control**
      The most elementary issue in any shared system is that of managing conflicting user actions. If you try to open a door while I am trying to lock it, who wins? What happens if I try to sit in a chair you are trying to move?

    - **Access control for both data and functionality**
      The normal, procedural approach to access control is simply to bracket all messages between objects in security routines: a pre-check to determine whether a mesage should be accepted at all, and a post-check to verify that no damage has been done, or if so, to note who was to blame. Unfortunately, VRML 2.0 has no built-in way to determine where a message came from. Until this bug can be fixed, it must be worked around (e.g., by sticking messages in an "envelope" with a "return address"). The harder problem is that VRML, being declarative rather than procedural, has no reliable definition of "before" and "after," and thus no easy way to specify conditional execution...

    - **Authentication of Users and their Agents**
      In principle, the Net has a solution for this problem: encrypted certificates validated by an independent Authority. Inside a given client system, the problem is to insure than certified users don't get impersonated by rogue software. This requires the creation of a "trusted path" along which a message which *says* it comes from component A on Art's desktop machine cannot be hacked to look as though it came from component B on Betty's.

This basic infrastructure for secure scene-sharing goes a long way toward enabling the creation of VRML environments that are truly *interpersonal*, i.e. places people can actively share. Living Worlds provides standard interfaces to support all these features, along with the architectural framework required to implement them from a set of *interoperable* components.

That is why the LW effort seeks to:

1. Identify the major *components* that make up a multi-user system, where by "component" we mean, "any piece which may be independently supplied"

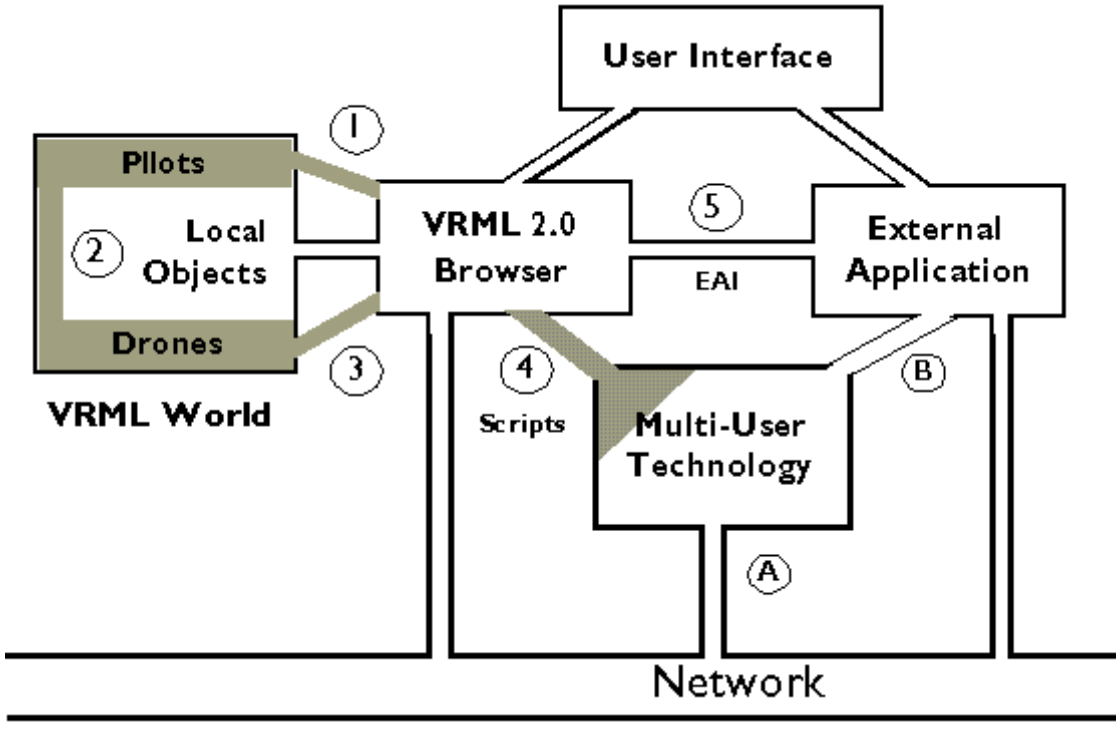
2. Identify the key *interfaces* between those components, and consider how they communicate with each other

3. Design a *VRML2.0 application framework* to support that component architecture.

The first task in this effort is to get agreement on a basic conceptual framework for implementation. The requirements analysis above distilled a general view of some basic interaction facilities needed to support shared virtual worlds. The next section seeks to locate this list of needs in a general conceptual framework that can effectively *guide* our investigation into potentially standardizable program interfaces.

3. A Conceptual Framework

The diagram below shows the place of the Living Worlds interface proposals in terms of a very generic web-browser architecture. *Note:* the blocks do not necessarily represent code modules; the point is to make a *logical* partitioning of the relevant functionality.



Legend:

The shaded blocks and the interfaces 1-4 indicate the domain of the Living Worlds proposals. Interface 5 is the External Authoring Interface proposal currently under

review; its approval is assumed, but it is unrelated to this proposal. Interfaces A and B are how the multi-user technology and other expernal applications communicate across the net; they are of course essential to the implementation of shared environments, but are outside the reach of a VRML 2.0. node library.

The first thing to note about this model is that it is derived from our Principle #2: our goal is not a system design, but an analysis that can inform a standards proposal. Thus, the only interfaces it identifies are those which support interoperation between components *from independent suppliers.* This approach leads to some definitions which may be unexpected:

- **User Interface** - accepts all user input, and renders the virtual world in media the user can interpret. This definition bundles the mouse and video and sound drivers together will all widgets, windows, frames, menus, dialog boxes, error messages, help files etc. - any and all media used by the system and its human user to interact with each other. Living Worlds proposes no UI standards or conventions.

- **VRML browser** - parses the content of VRML files and passes it to the UI for display to the user, and through which any and all changes in the actual run-time world are effected. By this definition, the Browser is the only component which interacts directly with the World. (Note: in these terms, Live3D is a browser which uses Netscape Navigator's UI). Also by this definition, *all* interfaces to the browser are defined by the VRML standard (as it continues to evolve). The two unlabeled Browser interfaces are those which exist today as part of any VRML 2.0 system: its access to the Network for files (including scripts and inlines), and its link to its own UI.

- **World** - the VRML code which specifies a (set of) scene(s), *plus* the local code invoked by its script nodes (Java, JavaScript, TCL, etc.) Since our purpose is to make it possible for objects to be inserted into scenes, and for changes made to objects in one instance of a scene to be mirrored in other instances of that scene, it is crucial to distinguish all objects in the world as either *local* or *shared*. The latter, i.e. those whose state and/or behavior is to be distributed, communicate via the interfaces 1-3 above between "pilot" and "drone" objects (see Terminology). The rest of the World is either static, or else behaves with local effects only.

- **External Application** - code which interacts with a VRML browser to realize some (set of) feature(s) not encoded in the VRML or its associated Scripts. Example: a videoconferencing application that places its images on your livingroom wall. The VRML standard will soon include an external API through which such applications can interact with the browser. Living Worlds provides a framework whereby objects in a world can negotiate their shared use of external applications.
  Note that external Apps may well have their own independent access to the network. This is how most current proprietary systems for placing avatars in

VRML worlds work (including those of Black Sun, Paragraph and Sony).

- **Multi-User Technology (MUtech)** - a component proposed by *Living Worlds* to implement shared behaviors/states across the network. By definition, MUtechs provide all network communication needed for multi-user interaction beyond that provided by the Browser itself. (Thus, to say that future versions of VRML will include multi-user capability as part of the standard is simply to say that part of the functionality we here label *MUtech* will someday become the responsibility of the Browser).
  The MUtech connects to the Browser via standard VRML 2.0 scripts (interface 4 above), the semantics of which are a key part of the LivingWorlds proposal. Interface B gives MUtechs a direct link to external applications, and through them back into the world via the Browser's external API. In real-world applications, such interactions are likely to be extensive, as VRML authors incorporate such things as external authentication systems or electronic cash transfers into their worlds, but those interfaces are outside the scope of this proposal.

This model of five major conceptual components is, as we have said, above all a *suppliers* model. We can foresee both a need and a market at each of the itnerface thresholds in our model. But it does not yet begin to address the fundamental difference between shared virtual environments and their predecessor platforms: that they are both multi-user and interactive. More: that each user potentially can interact with any other.

It turns out to be surprisingly difficult to speak clearly with precision about the requirements or the functionality of such apps. That is why our next step is to spend more time than many readers will find reasonable defining our terms.

Terminology

Sometimes definitions are chosen to minimize controversy, just to move the discussion forward. All of the definitions used in Living Worlds are intended to *expose* potential controversies, to be sure we are all talking about the same things. Thus, most of the rest of this document is devoted to terminology, expanding on those terms which are either new or used in a new way. We begin with the simplest:

User	Someone operating a *client* in real time
Client	A hardware/software system serving a single user
Server	A system coordinating communication among clients

Two critical distinctions should be noted here: first, a *user* is always a human; a *client* never is. Second, *client* and *server* are functions, not machines. The Living Worlds

specification makes no assumptions about the physical distribution of client/server functionality across machines. See Principle #3]

Worlds / Scenes / Zones

World	One or more *scenes* linked together both technically & conceptually.
Scene	A set of VRML objects which is a) geometrically bounded and b) continuously navigable, i.e without "jumps".
Zone	A contiguous portion of a scene; a container for SharedObjects.

To begin with, note that we do not call any file with a ".wrl" extension a "world." Indeed, the coming ISO standardization seems likely to replace ".wrl" with the more generic ".mdl" for "model." But that begs the question, "model of what?" -- in VRML, the world on the screen is not separate from the world but an extension of it, not "virtual," merely digital. We need some simple ways to refer to the different ways in which VRML files can be assembled and used.

The World / Scene / Zone set of terms allows us to distinguish easily between aggregations of separately implemented, independently presented *scenes* into conceptually integrated wholes (in this usage, a virtual city assembled from many individual neighborhoods would be a *world*), and subdivisions of what the user perceives as seamlessly navigable spaces into *zones* with very different characteristics (e.g. and a chess board on the table in a living room).

Technically, the mechanism for identifying objects as "shared" is to place them in a **`Zone`**, a spatially contiguous (though not necessarily fixed) region of a scene. Different zones may be used to group objects together for joint handling (e.g. on a game board). [*Note:* it is up to the MUtech to handle the case of overlapping zones, in particular to insure that only 1 instance of any Pilot is loaded. It may or may not be possible for overlapping zones to have different MUtechs; in any case, there are likely to be pair-specific rules for doing this.]

Thus, a **`Zone`** is the range of control of a MUtech. We anticipate that scene-authoring is likely to be dependent to some extent on the choice of the MUtech(s) used in its zone(s). To build a scene which supports high-speed interaction (shoot-'em-ups), you will need a MUtech that guarantees extremely low latencies; while for a shopping mall it will be more important to have a MUtech with integrated electronic commerce functions. (This raises the possibility of changing the functionality of a scene by switching MUtechs, e.g. using the Doom-style MUtech to stage a battle in a shopping mall?)

Avatars and other things in shared Scenes

SharedObject	Any object whose state and behavior are to be synchronized across multiple clients.
Instance	The SharedObject on one of these clients
Pilot	Instance of a SharedObject whose states/behaviors are replicated by other instances, its *drones*.
Drone	Instance of a SharedObject replicating the state/behavior of another instance, its *pilot*.
Bot	A SharedObject active under autonomous program control.
Avatar	SharedObject whose pilot is under real-time user control.

As used in the Living Worlds documents, **`SharedObject`** usually refers to all instances of the object (i.e. one on each participating client). For any change of state or execution of a behavior, the **`pilot`** instance is *emitting* the change, the **`drones`** are replicating it. The behavior repertoire of a `SharedObject` may well include the ability to change its state from `pilot` to `drone`.

Pilots / Drones

All shared objects have *pilots*, i.e. are capable of originating behaviors and state-changes. They also have a switch which by default sets the pilot to "off," i.e. most shared objects are drones: they only replicate locally behaviors and state changes originating elsewhere.

An avatar is thus a shared object for which the instance on its user's client is declared by default to be the pilot. To give your avatar a "take my hand" behavior means temporarily transferring control over its position from its pilot to the drone of some other avatar. Similarly, any instance of a light switch becomes the pilot for just long enough to switch the light on/off, then returns immediately to drone status. [Note that such complexities as sorting out conflicts among multiple instances who all want to be pilot are left to the MUtech developer.]

Avatars vs. other objects

The most obvious example of a pilot/drone pair is the avatar, but the concept applies equally to all objects behaviors shared across the network. If I pick up a bottle and throw it across the room in my local world, and that behavior is meant to be distributed (e.g. so it can hit another avatar on the head and invoke its "fall down unconscious" script), then in so doing I make that bottle a "pilot" which will drive "drone" representations of itself on all browsers currently linked by a given MUtech (see below).

Any shared object may in principle be used as an avatar, i.e. a real-time representation of a user. Conversely, there is nothing to prevent objects normally used as avatars from

being placed "on automatic" e.g. being left behind in a scene to function as answering machines. The Living Worlds specification provides a way to note whether an object is currently under live human control, and a persistent place in the scene graph for any object that needs one. (See the specification under "Object Insertion" for more details.)

Most existing implementations of shared VRML spaces recognize two very different object classes: transient and arbitrarily mobile (because human-driven) *avatars,* and persistent, predictable (because program-driven) *objects.* But the effort to distill our different implementations into a common framework forced us to recognize that these are not orthogonal classes. Chatterbots (avatar-like but program-driven) and persistent imports (such as Betty's birdbot) forced us to break the mold.

The avatar/bot distinction is taken from work by the Universal Avatars group, as documented here. There is still no community-wide consensus on how to best to think about -- let alone label -- the potentially very wide spectrum of autonomy/control which may characterize the relationship between a person and the object(s) representing that person in a scene. Suppose, for example, that I am participating simultaneously, via 3 different browsers, in 3 separate online conversations, and my onscreen representation has enough behavioral sophistication to respond to simple social cues (e.g. to greet certain people upon their arrival). If you respond to such a greeting, are you dealing with an autonomous bot or with me, via my avatar? The terms *puppet* and *agent* have been proposed to describe bots operating at the behest, but not necessarily under the real-time control, of a user, but it does not yet seem likely that such distinctions call for separate interfaces at the Living Worlds level.

MUtech: the Core Concept

MUtech *("mew-tek")*	"Multi-User technology" - Living Worlds' term for the component which manages the `SharedObjects` in a `Zone`.

The concept of a **`MUtech`** is the heart of the Living Worlds standardization strategy. The point is to isolate, conceptually and technically, the functionality which tracks and synchronizes object state changes and controls access to object behaviors, and to provide it with a standard VRML "wrapper." This allows MUtech suppliers to experiment freely with alternative implementation designs, and to access both network data and arbitrary external applications, without having to introduce any non-standard extensions to VRML 2.0. All world-builders need to do to profit from all this innovation is to adopt the VRML 2.0 Living Worlds conventions for scene management and object access when defining their multi-user environments.

From the standpoint of the world-builder, the MUtech links the local scene graph to a set of external network functions. From the standpoint of the MUtech developer, the MUtech provides a standard interface for tracking and manipulating VRML scenes. This allows world-builders to design a world for multiple uses, each implemented by a different

MUtech. It also gives MUtech developers the assurance that they can support avatars and other active objects of all kinds across multiple worlds.

To minimize the impact of differences between MUtechs, all functionality that can be MUtech-specific is split out into separate nodes to be supplied by the MUtech provider. These have a standard interface to the world-builder (see the Specification: MUtechNodes for details).

Coordinating Capabilities

We introduce the concept of *capabilities* as a standard means of access to non-standard functionality: ways for objects in a scene (avatars or bots) to interact with each other. Such interaction functionality may be implemented as part of the world, or as something independent of it. For example two users might have NetWriter (ParaGraph's whiteboarding) installed and when they "meet" in a virtual world they should be able to exchange the appropriate parameters (e.g. IP number and port) and launch the application.

Capability	SharedObject functionality implemented by an *application* or *script*.
Application	Functionality available in a World via the External API, rather than having been implemented via a VRML Scripts.
Script	as defined in VRML 2.0 spec

Capability is a generic term that allows us to speak about interaction functionality without implying that it is supplied by the author(s) of the world, or of its various objects, or by the MUtech provider(s), or by someone else entirely. Example: the Whiteboard in Scenario #1, which assumes special hardware and client software, but uses the generic MUtech features to distribute its visible state. Thus, a *capability* may be implemented as an external *application*, and invoked within the scene by a VRML *script*.

At present, developing a virtual community means designing and implementing virtually all aspects of the system. A key purpose of Living Worlds is to allow virtual communities to be assembled from independently developed components. This implies that interaction capabilities may come from several places:

- *Avatar* - they may be supplied as part of your avatar, available to you to carry with you from world to world
- *Client* - they are available as part of the browser, or installed separately by the user on the client (perhaps involving special hardware)
- *MUtech* - they may be implemented as part of a MUtech, and thus available in any scene supported by that MUtech.
- *World* - they may be implemented as part of the world, to be made available to anyone entering the world.

We say that a user "has" a capability if it is in any of the places above. Note that these sources are listed in the sequence of their effect: capabilities implemented as part of an avatar are the longest lived; they may be constrained or enhanced by the current Browser, and again as the avatar enters each new Zone, where the MUtech may add new features/constraints. This merged set of capabilities in turn becomes the basis for further enhancements and constraints by the World-author, who gets the last word at run-time (**TBD: should these be subject to user veto?**).

At run-time, exercising a capability presents design challenges in 3 stages:

1. *Selection* - giving each capability the opportunity to determine whether or not it should make itself available in a particular situation. For example, a capability which requires a partner with complementary capabilities should be able to determine whether to "gray itself out" rather than allow itself to be selected and then fail to initialize (e.g. a proprietary network phone program), or it might require a partner to belong to some group (e.g. Adults over 18).
2. *Initialization* - going through whatever handshaking is needed to establish an interactive connection prior to ?
3. *Running* - the actual operation of the capability may be wholly outside the scope of this proposal (e.g. because its comes with its own networking capabilities), but the capability author must be able to depend on a certain set of functionality from the MUtech, including the ability to communicate with both its own pilot and that of its interaction partner.

Each of these steps is complicated by the need to co-ordinate among the 4 possible sources of (and thus, potential constraints on) the capability. A crucial contribution of Living Worlds specification is to establish a standard framework (interfaces and protocols) for this process.

Dynamic vs. Download/Install. In the medium term, the goal is to handle everything via dynamically downloaded Java applets. But at present, most systems prohibit Java from accessing local files, which makes it impossible, for example, to connect to locally installed 3rd party software features. Until this problem is generically solved by the Java community, the management of downloads and local access are left to proprietary MUtech solutions.

4. Related Efforts

LivingWorlds is not the only initiative aimed at accelerating the evolution of shared virtual environments.

- The *Universal Avatars (UA)* initiative has attracted wide participation to its online discussion of technical, social and philosophical issues involved in trying to unify the multiple competing avatar-based systems springing up around the Web. (e.g. *Worlds Away*, *The Palace*, *Alpha Worlds*).

- In October '96, some of the UA folk, working together with the Mitsubishi Electronic Research Lab (MERL), proposed an *Open Communities* (OC) architecture based on the SPLINE system.

- Recently a proposal for a new Virtual Reality Transfer Protocol (VRTP) was floated, and *Tientien Li* wondered whether Living Worlds could be viable without proposing its own Level 1 solution (wire protocol). .

In the following sections, we distinguish a 4-layer model of potential interfaces linking shared virtual worlds, and use this model to relate the *Living Worlds* proposal to existing or potential efforts at the other design layers.

Interface Standards: 4 Layers

In proposing any interface standard, an essential design question to resolve is the choice of "layer" for the standard, in the sense spelled out by the OSI metamodel for communication layers. We can identify four layers of interface standard relevant to the construction of shared VRML worlds:

1. a wire protocol: the meaning of the bit-sequences in an IP packet. This is the level addressed by the proposed "Virtual Reality Transfer Protocol" (VRTP).
2. an API: the commands which create, send, fetch and interpret IP-packets. *Open Communities* clearly understands itself as a Level 2 (API) initiative.
3. VRML: the nodes, events, routes and scripts needed to support secure scene-sharing. This is the role of *Living Worlds*.
4. Grammar: the vocabulary and structures of interaction (e.g. social rights, roles & rules). In our terms, *Universal Avatars* is an effort to define Level 4 (grammar) standards.

A standard which offered interfaces at very different layers -- say, by trying to define both a standard catalog of Avatar gestures (hello/goodbye, yes/no, pride/shame, etc.) *and* the wire-protocol for communicating those gestures across the net -- would run into a number of problems. For one thing, it is unlikely that the group compiling the standard would be equally competent at both the taxonomy of social gestures and the design of efficient network traffic protocols; the risk that one or the other will be less than optimal seems pretty high. More importantly, such an interface would be hard for other groups defining other standards to mesh with. Suppose that a "yes" is meant to OK an electronic cash transaction. At what level should the Ecash standard be designed: to take its input from the "yes" (and if so, how does it know when the "yes" refers to a cash transaction?) or from the gesture-packet (and if so, how does it know which packets to inspect?)

Living Worlds group is clearly focused on level 3) in this model. The specification provides a set of VRML 2.0 interfaces, and care has been taken to avoid anything that would constrain the design of higher or lower interface layers. The hope is that this will make it easy to connect worlds built using the Living Worlds library to other emerging standard interfaces.

Why not a wire protocol?

TBD: This section should focus on the new VRTP proposal ...

Recently, as part of an evaluation of the first partial Living WOrlds draft, Tientien Li voiced concern that Living Worlds lacks a wire protocol definition. Other multi-user protocols on the Internet (IRC, multicast) have had such a protocol as their foundation. Why not Living Worlds? Here is his argument:

> In this kind of approach, vendors/developers are allowed to use proprietary wire protocols in the middle and, consequently, making both ends of the wire vendor/implementation dependent. This will likely rule out any possibility for browser/vendor independent multi-user VRML worlds. In the current LW spec, if a multi-user LW world uses MUtechs from BlackSun, then all its users will have to use BlackSun's. The same is true for LW worlds built from using Sony's MUtechs or ParaGraph's MUtechs.
>
> Umm... does this sound exactly like what we have right now?
>
> The lacking of a wire protocol making LW different from other successful Multi-User Virtual World technologies that I've worked with. For example, the Internet Relay Chat (IRC) is based on an ASCII wire protocol. The Distributed Interactive Simulation (DIS) and its predecessor SIMNET are based on binary wire protocols. Any MUtech implementation that confirms to the wire protocol specification can interoperate with others independent of its vendor/platform. This approach significantly opens up the concept and scope of ***sharing*** that can be achieved.
>
> However, this kind of interoperability is not possible with the current LW specification. Personally, I believe that a wire protocol is a necessity for any distributed open system standard. Without one, there will be no open interoperability. IRC, SIMNET, WWW, and the Internet were successful primarily due to the vendor/platform independent interoperability.
>
> from: "Tientien's VRML Notes," 4 Dec 96

This argument reflects a misunderstanding of what Living Worlds is up to.
Our goal a simple Library of VRML 2.0 nodes allowing:

1. users to experience multi-user VRML worlds supported by different MUtechs using only their standard browsers

2. world developers to select the MUtech that best supports the particular features they want to exploit

3. toolmakers to create generic tools that address the LW-MUtech interfaces, knowing that the MUtech vendors will be doing the same.

Note the priorities: end-users should need *nothing* special, and developers should be given a subset of basic functions that will be supported by all MUtechs. Compatibility should not be chased in areas outside VRML where the industry is still in flux (e.g. the protocol whereby a whiteboard optimizes the distribution of its drawings, or the radio the compression of its soundstream).

That is why it was not a goal to enable world-builders to mix and match MUtechs *within* a world. MUtechs will compete by being able to support different sorts of worlds (e.g. a Myst world with super multimedia, vs. a stock-trading world linked to huge databases and proprietary international wire feeds). It should not be impossible to have different Zones in a world supported by different MUtechs, but it seems far too early to try to standardize, for example, what should happen where those zones overlap. That's the sort of thing a world-builder will have to *design*.

Perhaps our experience with server-based systems colors our thinking here. We tend to assume that most worlds that are more than personal playgrounds will be supported by various central services: they will access data bases, refer to certification authorities, maintain world states that are both complex and persistent, etc. Someone designing and building such a world will not have interoperability among MUtechs at the top of his priority list. They will want to be sure that their target customers will face no barriers to entry, i.e. that their world will work with any standard browser. And they will look for particular features, quality of the provider's service/support, and perhaps price or licensing conditions will play a role. They will pick a MUtech provider, and build their world to exploit that system's particular capabilities.

We think that the best target for a standard at the current state of our art/technology/industry is a set of interfaces that encourages competition on features based on technical innovation. That means keeping the semantic level of the interfaces fairly high, to allow the innovators to work out ways to do more things faster "under the covers."

We share the view that, ultimately, the world will want a standard wire protocol. But we are doubtful that the time for that is ripe. All of us have implemented wire protocols. And one of the big lessons we've all learned is that, the more application experience you gather, the more you change your mind as to how the wire protocol should look. In particular, different protocols are optimal for different purposes: exchanging digital whiteboard ink has different priorities than tracking avatar motion; managing a living-room (or even a ballroom) full of people, all of whom may equally be the source of interesting behavior, is very different from conveying a stage performance by 5 people to

a stadium-size audience. Adventure games and action games have conflicting priorities, neither of which is much like those for online customer service.

At this point, it seems to us a better strategy is to build on VRML 2.0, and risk that the browser-builders may take a while to conform to the standard, than to try to persuade the community to commit to a wire protocol which seems likely to be either a) too simple to cover most of what needs doing, which means everyone will end up writing their own extensions, leaving us with no standard at all; or else b) too deeply identified with some existing product set, giving one vendor such an advantage that others will be forced to find new features to compete with, which will require proprietary extensions to the protocol, bringing us back to a).

In short, while we share Tientien's concern, we don't believe that trying to establish a wire protocoal standard would work at this stage. The simple things you could get agreement on won't cover a range of features broad enough to build commercially successful worlds with. The more complex protocols that successful apps will require are not yet so well proven that we could easily get agreement on them.

It can be argued that a good approach would be to create a standard API which hides the wire protocol but doesn't rest on the "shifting sands of VRML" which you mentioned in your message. The LW proposal does not preclude the design or even the standardization of such an API ; for that matter, it doesn't prevent the creation of a standard wire protocol either. What it does do is provide a standard for *content creators* to write to; both an API and an over-the-wire protocol can (and will!) follow later.

Open Communities: an API-based Approach

Open Communities is a proposed API standard for communication within and between shared virtual worlds. It is based on the well-known SPLINE system developed at the Mitsubishi Electronic Research Laboratories (MERL). *Living Worlds* and *Open Communities* both ask the same question: what aspects of the technology for shared virtual worlds should be standardized, what left open for innovation?

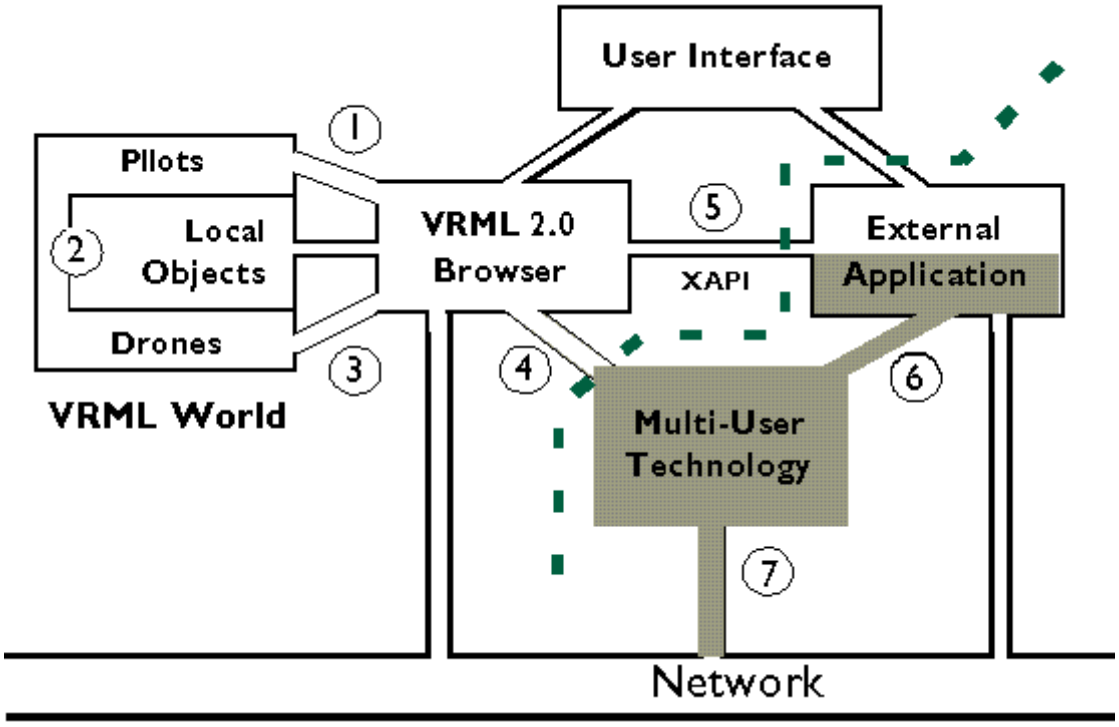
Living Worlds assumes that VRML will be the language of choice for implementing multi-user virtual environments. LW focuses clearly on what that can be accomplished using VRML 2.0 -- anything that can't be done inside the current standard is declared by definition to be "MUtech specific," i.e. an implementation matter outside the boundary of the interface standard. [cf. Living Worlds Principles #4 and #5, *Respect the Market* and *Require Running Code.*]

By contrast, the Open Community work makes no assumptions about the use of VRML. Instead of starting with objects in a virtual space, they start with a "world model" that is a pure semantic abstraction, explicitly leaving open the matter of object *appearance* -- in fact, OC worlds need not be 3-dimensional. Their goal is to insure that *behaviors* in different worlds -- however the behaving entities may be visualized -- can be coordinated among multiple participants on distributed platforms. Thus, they offer a standard set of

multi-user functionality, and make this available by APIs in Java and C that can be used to implement arbitrary (in our sense, "external") applications.

Thus, OC proposes standardizing precisely those interfaces which LW proposes to keep open. In the diagram above, the two initiatives are neatly separated by the dotted line. At first blush, this would seem to make their work wholly complementary: each is working on what the other has chosen to avoid. Does that mean all we need to do to achieve a *complete* standard is to integrate the two together? A quick diagrammatic comparison suggests how neat the fit might be.

Here the domain of the *Open Communities* effort is mapped onto the LW conceptual model presented in Concepts and Context. (*Note:* the blocks do not necessarily represent code modules; the point is the *logical* partitioning of functionality.) Click here to compare the Living Worlds diagram:



This neat congruence of boundaries has not gone unnoticed. Work is already underway from both sides to bring the two efforts closer together. The OC team is looking at implementing Living Worlds' MUtech interface, so that VRML developers will be able to build components that interoperate with OC environments. And the Living Worlds MUtech providers are looking at how their work might profit from the OC APIs.

One difficulty is that of timing. When you try to standardize everything at once, there is no room left to experiment in: remember the lesson of OSI vs. TCP/IP.

The original strategy considered by the Living Worlds group was in fact to propose an API standard, working solely through the newly proposed external API, and thus allow the line protocols to remain proprietary. The thinking was that any attempt to standardize the transfer protocols would pose an unreasonably hire entry barrier to new suppliers, and/or give one of the existing suppliers an unacceptable competitive advantage.

These same arguments, however, eventually proved to apply with almost equal strength to the idea of a wholly API-based standard. Also, we were concerned that this approach did not make appropriate use of the new ROUTEing capability of VRML 2.0.

The final decision was to effect all communication among objects within a scene using VRML language mechanisms, and to provide a customizable set of VRML "wrappers" through which providers could introduce proprietary features. The chosen architecture is flexible enough to allow developers to work either through the external API or directly within the Browser's scripting capability.

Universal Avatars: The Grammar of Social Interaction

Universal Avatars is the name of a proposal, but perhaps more importantly, of a mailing-list. Its focus is on the challenge posed by the rapid emergence of multiple avatar-based virtual communities on the WWW. The fascinating and valuable discussion generated on the UA mailing list, dealing with such issues as gender representation, ID authentication, personal expression *vs.* social constraints, avatar/world scale and the communication of emotion, is semantically far "above" the LW group's Level-3 technical concerns with data distribution and scene synchronization.

To make a (clumsy) analogy: if we were designing real-world cities instead of virtual ones, the UA discussion would be about the design requirements of office spaces, residential subdivisions and amusement parks, while the LW discussion would be about the need for standard plumbing joints and strategies for delivering electrical power at half a dozen different electrical voltages.

In terms of their detailed content, the UA and LW discussions hardly overlap at all. They do, however, inform each other. The LW distinctions between *world*, *scene*, and *zone*, with their technically separate initializations, was in part a response to an extended UA discussion on the conceptual difficulty of taking one's avatar into worlds at either molecular and galactic scales. Similarly, the design of the LW `sharedObject` and `pilot` nodes is intended to both technically enable and philosophically undermine the distinction between *avatar* and *bot* which has occupied so much UA bandwidth.

Most of those involved in the LW and UA efforts understand all this quite well. There is nothing new about the need for different levels of discussion between people who are trying to sort out application requirements and priorities and people who are trying to identify the relevant technical possibilities and constraints. Only the widespread habit of looking at every new initiative in terms of its potential for intellectual and/or commercial rivalry makes it necessary to engage in this kind of "media weed control."

- End -