

**ELECTRIC
COMMUNITIES**

280 SECOND STREET
LOS ALTOS, CA 94022
415.917.5640
ELECTRIC@COMMUNITIES.COM

Cyberspace Protocol Requirements

Version 27-February-1995

©1994, 1995 by Electric Communities, all rights reserved.
Proprietary and confidential.

Executive summary

This document presents the functional requirements for the protocols that will be the technical basis for the Global Cyberspace Infrastructure (GCI).

The protocol requirements are presented in four stages. First, we discuss the overall system goals which are the target of the Cyberspace effort. Second, we derive a set of specific technological and institutional features needed to realize these overall system goals. Third, we organize these features in a way that will guide us in protocol design. Finally, we identify a set of specific standards for protocols and other components that must be formally specified in order for the Cyberspace network to be implemented. This final stage provides a working roadmap that the Cyberspace protocol implementation team can use to organize its activities in the coming months.

We begin with the overall system goals. We have identified eight high-level characteristics that the Global Cyberspace Infrastructure architecture must possess:

- 🌐 Scalable The technological and institutional components should be sufficient for a system that includes every person and computer in the world.
- Open Cyberspace is open to new providers of services without regulation and at low cost.
- * Decentralized There exists no singular privileged technical or administrative nexus.
- ↔ Traversable Data and objects can move between users, between services, and between machines.
- \$ Commercial Cyberspace contains a complete foundation for economic activity of all kinds.
- 👥 Social Cyberspace contains the components necessary to support community life.
- 🔒 Secure The technology facilitates making good decisions about which entities can be trusted and protects users from the untrusted ones.
- 👜 Portable Protocols and service features are logically independent of the technical details of the physical network.

From these overall system goals we derive the following collection of features, specific medium- and low-level technological and institutional characteristics which any design must possess if it is to satisfy the above high-level goals:

🌐 Scalable

- Global object & service address scheme
- Algorithmically tractable message routing
- Multipath network topology
- Built on top of existing standards
- Extensible choice of media formats

O Open

- Many-to-many communications model
- Dynamically extensible object system
- Open standards review, certification, update and publication process
- Benign regulatory environment
- Fully connected network topology
- Published protocols with unlimited distribution

* Decentralized

- Peer-to-peer client/server relationship
- No global state information
- No monopolistic administrative authority
- No “superuser”
- No required proprietary components

∴ Traversable

- Distributed transportable objects
- Common format for messages, data types
- Object persistence

\$ Commercial

- Digital money
- Electronic credentials
- Product/service directories
- Banking and other financial services
- Reputation services
- Arbitration services

†† Social

- Sense of place
- Multinational language support
- Bill of Rights continuity
- Inhabited world
- Language translation services

🔒 Secure

- Link encryption
- Secure capability semantics
- Object behavior certifiers
- Identity and message authentication
- Identity certifiers
- Recognition of user right to privacy

👜 Portable

- Bandwidth independence
- User interface independence
- Transport medium independence

These features are then implemented by a collection of protocols and other standards. These protocols are organized into three broad levels, which we call the Foundation, the Ground floor, and the Superstructure.

The Foundation level provides the base on which everything else is built, including the fundamental syntax and semantics for interoperability. The protocols and standards at this level form a general purpose foundation for distributed computation. It contains the following elements:

Joule: A semantic model for secure concurrent programming

F1: Address Generation & Resolution Protocol

F2: Server Interface Standard

F3: Primitive Data Representation Standard

F4: Interserver Message Protocol

The Ground Floor level provides the essential building blocks of the Cyberspace world. It specifies the things out of which the services, places and other elements of the Cyberspace environment will be constructed. It contains:

- G1: World Object Model
- G2: World Object Transfer Protocol
- G3: Message Privacy Standard
- G4: Message Authentication Standard
- G5: Certificate Management Standard
- G6: Capability Management Model
- G7: Media Extension Standard

The Superstructure level provides standards for the core services that are required to make the Cyberspace environment viable as a marketplace and as a society. At this level are found service frameworks, as well as standard components for virtual world construction. This level contains:

- S1: Directory Framework
- S2: Minimal World Object Set
- S3: Financial Framework
- S4: Credentials Framework
- S5: Server Validation Framework
- S6: Contract Framework
- S7: Linguistic Framework
- S8: Juridical Framework

In addition to the three protocol levels, we also identify specific key projects needed to bring the system into existence. These aimed at proving and deploying the technology and its related institutions. They include:

- Cyberspace Standards Organization (CSO)
- Cyberspace Regulation Project
- Joule Programming System
- Implementations of protocols, models, and standards
- Reference Backend
- Exemplar Frontend
- Implementations of services

Table of Contents

Introduction	9
Overall System Goals	11
Scalable	11
Open	12
Decentralized	13
Traversable	14
Commercial	15
Social	16
Secure	16
Portable	17
Features To Realize Overall System Goals	19
Features required for Scalability	19
Features required for Openness	20
Features required for Decentralization	21
Features required for Traversability	21
Features required for Commerce	22
Features required for Society	23
Features required for Security	24
Features required for Portability	25
Organization Of Features	27
Protocols & Standards To Implement Features	29
Approach	29
Foundation	30
Ground floor	32
Superstructure	35
Projects	39
Conclusion	41
Appendix A: The Distributed Instantiation Object Model	43
Appendix B: Glossary	53

Introduction

This document describes the functional requirements for the protocols that will be the technical basis for the Global Cyberspace Infrastructure (GCI). The concepts behind Cyberspace are described in greater detail in an earlier Electric Communities document, "The Future Interactive Network Environment (FINE): An Introduction To Global Cyberspace", which was prepared in August 1993 as part of a research study which Electric Communities did for Fujitsu, Ltd. The FINE document describes the rationale behind the Cyberspace effort, the business and technological trends leading up to it, and proposes a course of action culminating in the deployment of a global open system that will ultimately reach everyone on earth. This document is the first step in that course of action.

For reasons of available time and space, the presentation of the Cyberspace protocols in the FINE document was somewhat cursory. The intent there was to give the reader a taste of the kind of system we envision. In contrast, this document will go into much greater depth. The intent here will be to present a detailed statement of the requirements for the next step in the Cyberspace implementation effort, the formal specification of the protocols themselves.

The protocol requirements will be presented in four stages. First, we will discuss the overall system goals which are the target of the Cyberspace effort. Second, we will derive a set of specific technological and institutional features needed to realize these overall system goals. Third, we will organize these features in a way that will guide us in protocol design. Finally, we will identify a set of specific standards for protocols and other components that must be formally specified in order for the Cyberspace network to be implemented. The ultimate objective is to have a working roadmap that the Cyberspace protocol implementation team can use to organize its activities in the coming months.

Two appendices are attached. Appendix A presents the Electric Communities distributed instantiation object model, one of the foundation technologies used in the Cyberspace design. Appendix B is a glossary that defines some key terms which are used in this document. In many cases the glossary describes our usage of common words like "object", which tend to have many different and conflicting definitions.

Overall System Goals

We will begin by laying out the overall system goals. These derive directly from the Global Cyberspace vision as presented in the FINE document. While this vision is very broad, we have abstracted from it eight high-level objectives which we feel are key to any putative Cyberspace environment.

In brief, the Global Cyberspace Infrastructure architecture must be

- scalable,
- open,
- decentralized,
- traversable,
- commercial,
- social,
- secure, and
- portable.

Each goal will be stated as a one sentence description followed by a rationale. The short descriptions serve to capture the essence of the goals in phrases that are brief enough to be memorable but long enough to be complete. The rationales explain why we have chosen these particular goals and how they fit into the big picture.

Scalable

The technological and institutional components should be sufficient for a system that includes every person and computer in the world.

The most fundamental property of the Global Cyberspace Infrastructure architecture is that it must scale to arbitrarily large sizes. This has profound ramifications. The fundamental mechanism for dealing with any kind of large population, be it a population of people, a population of computers, a population of documents, or a population of ideas, is the same: parallelism — enable many different things to happen at once. All of the requirements we identify here have to do with enabling parallelism in one form or another. In other words, scalability leads to all of the other requirements which we describe below.

Our requirement that we be able to deal with the *entire* population of people and computers in the world is simply a statement that we must be able to cope with the maximum possible extent of system growth. In other words, we don't want to be limited by anything other than our own ability to evangelize the system.

Requiring that we be able to accommodate the entire world is also a statement of our ambitions. We intend Cyberspace to be *the* framework for telecommunications in the information age. If users feel they must take some telecommunications outside it for some reason, then we are lacking something essential. Note that this is why we talk about both the technological and the institutional components — both are key to the system's ultimate success. Inadequacies in the present telecommunications infrastructure are often more institutional than technological. Even technological problems often have institutional roots; we need to take this into account and plan accordingly. We must consider the institutional aspects because many of the obstacles to acceptance, growth, and continued evolution of the system will come from within this sphere.

Finally, though this is a bit of a paradox, taking in the whole world is something of a simplification. Many of the complexities of current systems result from requiring them to interoperate with external systems. If the external systems are defined away, then this concern vanishes. Keep in mind that the complexity of all those grungy layers of interoperability will still have to be accounted for. We can't naively assume that the whole world is going to change to our way of doing things simply because we think we have a good idea. Rather, we mean that we are allowed to define our ultimate mission at a higher level of abstraction. It means we can treat interoperability as an implementation issue rather than an architectural one. This will yield a degree of conceptual simplicity that will be very liberating, as we shall repeatedly see below

Open

Cyberspace is open to new providers of services (or of the network itself) without regulation and at low cost.

We believe that our objectives for Cyberspace are achievable only in an open system. The comprehensive scope of the Cyberspace vision means that the number and variety of specialties that will be involved is almost limitless. The range of things to be done, not to mention the range of opportunities to be exploited, exceeds the capacity of any single organization. A large organization cannot afford to attend to the endless specialized needs of thousands of small and medium sized market segments, yet these are what make the world go around. On the other hand, the smaller organizations which serve these markets cannot individually afford to create the entire environment in which they operate. They must work cooperatively, with each other and with larger companies that serve the mass market and the general needs of the infrastructure as a whole. Since, almost by definition, these smaller organizations cannot spread themselves very widely, it takes many of them to cover the field.

Furthermore, variety and competition are necessary to drive the evolutionary engine that generates successful systems. No single person or organization, no matter how brilliant or dedicated, has a complete understanding of everything there is to know about the environment in which Cyberspace will operate. Furthermore, there is no monopoly on creativity. The good ideas which propel progress can come from any corner of the marketplace. It is often small organizations who make the key innovations that advance the state of the art. A diversity of approaches is what enables this process to occur. On the other hand, some projects are simply large and require the efforts of a large organization to get them done. Thus we see that it is necessary to have organizations over the entire spectrum of size and specialty.

In the FINE document we defined three levels or elements of our open systems model. The terminology used in that document was not ideal, so we will take this opportunity to relabel the three levels: *platform*, *support*, and *delivery*. The platform level is the actual network infrastructure itself: data transport, message switching, protocol software, communications hardware, and so on. The support level contains those services that are available within the Cyberspace environment and make it usable, such as banking, transaction processing, or file storage. The delivery level contains products and services which are delivered through Cyberspace as a medium, such as video-on-demand, market research studies, or contract programming. The arguments above in favor of an open system apply at each of these levels.

Many of the established American telecommunications carriers — telephone companies, cable TV operators, and so on — have recently begun efforts to establish pieces of a more

advanced telecommunications infrastructure. However, due to the business and regulatory roots of these companies, they almost invariably adopt the open systems approach at just one or two levels of the systems model, for example at the delivery level, while they attempt to retain a proprietary lock, or even a government sanctioned monopoly, on the remainder. The managements of these companies believe that they are working in their own best interests. However, we think that their failure to accept the open systems model wholeheartedly is not only wrong from an idealistic point of view, but is, in fact, poorly matched to the evolutionary trajectory of the industry. Trends in regulation, customer demand, and the needs of the international marketplace all put tremendous pressure in favor of the fully open systems approach, an approach we expect ultimately to prevail. Organizations which prepare now to work within the open systems paradigm are going to be in a far better position to compete in the telecommunications industry of tomorrow. Embracing the open systems model is the strategy best aligned with the innate natural tendencies of this particular business.

Note also that an open system implies open standards. A large, diverse and rapidly growing environment cannot be organized around a proprietary standard. For a large business contemplating the substantial investment that will be associated with the development of Cyberspace, the temptation to institute a "protocol tax", to reserve some key element of the protocol structure as a proprietary component for which, say, license fees must be paid, is nearly overwhelming. However, this temptation must be resisted. The global market has little tolerance for such things. Consider, for example, the overwhelming success of the IBM PC compatible architecture in comparison to the Macintosh. The Macintosh is clearly superior in nearly every aspect. Nevertheless, the freedom from the legal and bureaucratic entanglements associated with Apple's intellectual property means that the market has been able to produce a vastly larger range of products, services and configuration options for the PC and sell them at a substantially lower price. The PC standard is clearly the winner and Apple is left gasping for breath — and Apple's probable strategy for trying to cope with this situation is likely to involve opening up their architecture to one degree or another. Proprietary standards which *have* succeeded, for example the Nintendo Entertainment System, have done so by having an extraordinarily narrow focus combined with an essentially static product design and a correspondingly bounded product lifetime. But Cyberspace will be neither narrow nor static.

Decentralized

There exists no singular privileged technical or administrative nexus.

In order to function at the large scale we require, there can be no bottlenecks in the system's normal operation. Centralization leads to bottlenecks. Decentralization avoids them. This principle is well understood at the technical level, where distributed processing has for years been the approach of choice for large systems, even in the most autocratic of organizations. Massive centralized facilities tend to be difficult, inefficient, and very, very expensive, which is why almost nobody builds them. The argument for decentralization on the technical level is clear and uncontroversial.

Unfortunately, people are much more prone to want to centralize things on the administrative side. Alas, centralization here leads to bottlenecks just as readily, and for the same reasons. Hierarchies are a partial solution to this problem, but they have a number of structural failings as things grow large. In an administrative hierarchy, the peak of the organizational pyramid is an information bottleneck. As the system grows, the base of the pyramid spreads out and the amount of information that must be processed at the peak grows geometrically. The only way to cope is to compress, abstract and discard

information as it flows upwards. The result is that, in a very large system, by the time the information gets to the top it has ceased to have any useful connection to the underlying reality. Decisions get made on the basis of this grossly compressed and distorted picture, and things go awry.

The reason that people continue to be seduced by centralized administrative structures is that they are easy to understand, even when they are failing, whereas the way a large but highly decentralized system functions, even when it is working well, is much less obvious. The principles of order and structure in a network lacking any point of global control are in fact well understood in the economic and sociological literature. However, these principles tend not to be universally believed, since they contradict many of the naive intuitions that human beings have evolved about the way the world operates. Consequently, these principles tend to be controversial and often politicized. It is very difficult to get existing organizations to accept them. Nevertheless, Cyberspace will function in a decentralized manner because it must.

The need for decentralization follows from the openness requirement. An inclusive open system will necessarily involve the participation of a very large number of entities, who will have divergent and often directly contradictory aims and who must nevertheless be coordinated successfully. No single principle or authority can adequately reconcile the many interests and agendas involved. The only hope is to allow subsets of the participant community to work out whatever piecemeal accommodations they can. This cannot happen if there is a board of directors or a regulatory commission or a telecommunications czar who must approve every move.

Another pressure for decentralization arises from our global scope. The international telecommunications environment is regulated by a collection of overlapping treaties and other agreements between sovereign nations and competing transnational corporations. Nobody is in charge of the whole thing, yet this is the environment into which we seek to deploy Cyberspace.

Finally, it should be pointed out that, on both the technical and the administrative sides, a singular nexus of control or communications is an opportunity for single point failure. That is, any breakdown at that nexus effects the entire system catastrophically. This too is better understood and better accepted at the technical level, but is, in fact, a general principle.

Traversable

Data and objects can move between users, between services, and between machines.

In order to operate in a decentralized manner, it must be possible for the activities taking place at different locations in the network to contribute jointly to some output. In other words, I should be able to take the result of some service performed for me by one person and give it to a second person for them to perform yet another service, or to combine that output with something I do and deliver this result to a client of my own. To do this, we need a common representation for the information shared among us. While in a strictly logical sense it is possible to speak a different language with each entity you communicate with, in a practical sense this is not feasible. Supporting a large number of different representations and structures for essentially the same information is expensive and unwieldy. Furthermore, a complete babel of languages is terribly problematic when

initiating contact with someone new — without some a priori basis for mutual understanding, establishing communications is impossible.

Often, information and services in Cyberspace won't merely be passive data but objects with a behavioral component. To share information, these objects must be able to move from machine to machine over the network — the network must be traversable. I would like to be able to pass an object down a wire to someone else and have them be able not merely to understand it but to incorporate it usefully into a running system of their own. Doing this requires solving a number of thorny technical problems, of course, which we will discuss in further detail below when we begin describing actual system features. However, even with the solutions to these problems in hand we still must agree on a particular binding to actual representations of some kind.

Achieving the degree of interoperability implied by this goal will require complex and detailed technical standards. In some measure this conflicts with the objective of having no central nexus of administrative control, for what is a standards body but just such a nexus? This conflict can be resolved by noting that standards can (and often are) effectively generated by consensus rather than by fiat. Note that consensus does not imply design by committee. A standard can be generated and promulgated by a small group or a single organization (and good standards often are). Consensus appears in the acceptance of this standard by the community which it effects, and in the degree to which this community's needs and concerns are able to feed back into the standard's further development. In the realm of human affairs, good examples include natural languages and common law. In the field of computers and communications, the Internet protocol suite comes to mind. This is an example not only of a protocol family to be considered closely in our deliberations, but also an example of a standards making process with no central control but a tremendous degree of effectiveness. The role of standards bodies in such a process is to document the emergent consensus in a clear and systematic way. To play such a role, an organization need not be a monopoly nor does it require enforcement powers.

Commercial

Cyberspace contains a complete foundation for economic activity of all kinds.

In order to be open and decentralized, commerce is a necessary ingredient. In addition, commerce is likely to be a large part of what people will want this system for in any case. Thus support of commercial activity is a crucial ingredient. The institutions of the commercial world currently have few analogs in the electronic environment. However, the emergence of such analogs is only a matter of time, as interest in such things is heating up in the business and financial communities.

Our concern here is not whether such features will ultimately emerge but what form they will take when they do. As stated above, we want the set of features to be complete. It is not clear that without our intervention that they necessarily will be. For example, cash transactions are not part of the commerce model being considered by any of the major players in the electronic commerce field that we are aware of, yet cash transactions are critical to certain types of business.

We also require the commercial protocols to enable low transaction costs. It is very easy to specify a set of commercial protocols if transaction costs are not a major issue. This is the approach taken, for example, by EDI and EFT, which require complex institutional arrangements made outside the electronic portion of the system. Such external arrangements, however, are extremely costly and only warranted in situations where the

transactions have high value or where there is an expectation of an ongoing long-term relationship between the parties. However, neither of these characteristics applies to a large fraction of consumer transactions. Buying one's groceries, for example, does not generally involve paperwork, lawyers or credit checks.

Social

Cyberspace contains the components necessary to support community life.

Alongside the commercial component of human affairs is the social component. Our experience has shown that the social element of human interaction is a large part of what people use computer networks and the online environment for. The urge for people to get in touch with other people is what drives much of the development of these systems. Email, for example, is all about person-to-person communications. Similarly, the Usenet is the world's largest ongoing free speech forum, with literally millions of people engaged in a gigantic extended conversation about every aspect of human existence.

The ability to support social interaction is thus in large part an answer to market demand — people want it so we must provide it. However, it is also really the underpinning for the institutional elements that will be required to make Cyberspace functional, to make it, in fact, habitable to its inhabitants. All of the other elements of the system require people to coordinate, to negotiate, to stay in contact with each other, in order to maintain the system and to support the ongoing evolution needed in order for it to remain viable.

Note once again the interaction between this requirement and those of openness and decentralization. People must be able to interact with the other people on the network, not merely with services. This argues in favor of a symmetrical communications relationship between each user and the network. People in this architecture are not merely the recipients of a broadcast nor are they isolated islands. People are, in a sense, what the system is ultimately made of. This principle is not universally recognized by many of the companies now seeking to establish themselves in the "information highway" business.

Secure

The technology facilitates making good decisions about which entities can be trusted and protects users from the untrusted ones.

Discussing security makes many people uncomfortable, for it implies distrust of others. However, in the extended global network we envision, people will not be able to employ the kinds of reassurances that they rely upon in face-to-face interaction. Security is an essential ingredient in meeting the goals of decentralization and traversability and in the support of both the commercial and the social sides of the environment.

Security is required because of decentralization. Since there is no ultimate authority to whom one can appeal in the case of misbehavior, it is best to ensure, insofar as it is possible to do so, that such misbehavior can't happen in the first place.

Security is required because of traversability. If you accept an object produced by somebody else into your own system, you need to have some assurance that it is not going to do anything bad to you. Similarly, if you give an object to somebody else, you are, in a sense, giving them a piece of you, and again you need this assurance. In either case, it is in the interest of each party involved to be able to convince the other that it is safe to proceed. Such assurances are only possible within a framework that defines what can

happen in a way that lets people make informed judgments about what they are willing to accept.

Security is required because of commerce. Commerce implies money, of course. Any time money is involved, security automatically becomes part of the equation, since the incentives for fraud and other sorts of criminal behavior can be very great.

Finally, security is required because of society. People interacting with other people need to be able to protect their identity, both to control who they reveal themselves to and to prevent other people from masquerading as them. Intimacy of human contact, even through an electronic channel, can expose deeply rooted emotional vulnerabilities. People need to be able to control when and with whom they open themselves up in this way.

Security does not lie in the attainment of some mathematically perfect property of information theory, nor does it come from eliminating any possibility of fraud or mischief. Rather, security means that the harm that can occur when something or somebody goes wrong can be contained within acceptable bounds. It means being able to know what the risks are in any particular choice of interaction, so that sensible judgments can be made about when to proceed and when to stop and demand further assurances. Such judgments amount to cost/benefit tradeoffs. To paraphrase cryptographer Eric Hughes, "Security is all economics."

Portable

Protocols and service features are logically independent of the technical details of the physical network.









The need for software portability has long been understood in the computer industry. Computers evolve rapidly, so that this year's state-of-the-art wonderbox is next year's obsolete doorstop. Yet software must continue to be supported, even though it rests on this foundation of sand. So too will it be with the Cyberspace network. We wish to build on top of the existing telecommunications infrastructure, yet we know that it is going to change in the future. We can see that new developments in data communications technology are coming. We can also expect, if we are at all successful in our Cyberspace effort, that new types of communications infrastructure will be developed solely to enhance the operation of the Cyberspace protocols themselves. It should go without saying that we need to be able to encourage and exploit such developments.

Features To Realize Overall System Goals

In this section we will derive a specific set of features in order to meet the set of overall system goals described above. Those goals are, of course, somewhat abstract. We will get more concrete and start getting into particulars, by presenting the set of features which we derived from the overall system goals. These features form an interlocking web that collectively realize the goal set.




An important criterion for any prospective system feature was not only that it fill in some coverage gap for one of the overall system goals, but also that it not undermine other goals while it supports its main target. For example, some naive approaches to security can undermine the goal of decentralization, and vice versa. Any feature which promotes a particular goal does so in a context containing all of the other goals as well. Thus this set of features works as an integrated whole — each piece requires the others before it can completely achieve its intended function within Cyberspace.

Although a rather large number of feature requirements emerge, we have tried to keep the set as small as possible. The objective is to specify a set which provides the complete functionality implied by the overall system goals. For convenience of presentation, each feature is grouped with others that support the same goal. However, each feature actually supports multiple goals simultaneously. We have grouped each feature with the goal that we felt it most closely served and attached a list of other related goals to it. After the title of each feature will be brackets containing a series of icons, indicating the overall system goals which that feature supports. Here is the key to these icons:

Icon	Goal
	Scalable
	Open
	Decentralized
	Traversable
	Commercial
	Social
	Secure
	Portable

Each feature may also be loosely categorized as to whether it addresses a technical issue or an institutional one. Since the boundary between these two categories is often somewhat blurry, we have not made any particular attempt to separate them. Instead we will discuss them all together. However, we will comment upon the distinction as we discuss each of the pieces. After the list of supported goals for each feature, we will also tag “Tech” for technical features or “Inst” for institutional ones. Some features may be tagged as both.

A Scalable system requires...

Global object and service addressing scheme [ *   Tech] — We require a universal addressing scheme so that any object or service in Cyberspace that wants one can have a unique identifier, so that it may be made the recipient of a message or referred to in a

message, assuming that the sender of the message has the ability to see the object or service in the first place and has the right to disseminate this information.

Multipath network topology [🌐 * Tech] — This simply means that there is, in principle at least, more than one communications pathway from one point to another. This provides for redundant routing that enables messages to bypass congestion or network damage.

Algorithmically tractable message routing [🌐 Tech] — The mechanism used for picking the communications pathway from one point to another must not “blow up” as the size of the overall network becomes large.

Extensible choice of media formats [🌐 ○ * :- Tech] — Protocols must be capable of handling data in a variety of formats, including text, audio, video, and so on. It must be possible to determine the format used for any particular piece of information so that it may be interpreted correctly. It must be possible to extend the set of formats supported so that future developments can be incorporated.

Built on top of existing standards where applicable [🌐 ○ Tech Inst] — Protocol designs should use as much of the existing network infrastructure as makes sense. For example, if TCP/IP is satisfactory as a transport mechanism, a new transport level protocol is not required. This feature will maximize the value of what has already been done and minimize the cost of the additional required Cyberspace research and development.

An Open system requires...

Many-to-many communications model [○ * 🗑️ \$ Tech Inst] — The essential communications paradigm is the many-to-many model, rather than the point-to-point (telephone) or broadcast (television) models. This means that any node in the network has the ability, in principle, to transmit a message to any group of other nodes. Note that the point-to-point and broadcast models are proper subsets of this, so that services built on top of those models are not excluded.

Fully connected logical network topology [○ Tech Inst] — This simply means that any point in the network can be reached from any other. In other words, it is a single logical network, not a partitioned set of networks that do not intercommunicate.

Dynamically extensible object environment [○ * :- Tech] — Any node in the network can add new objects and services to the Cyberspace universe. Furthermore, any node may add new types of objects and services.

Published protocols with unlimited distribution [○ 🌐 Inst Tech] — The protocols that implement Cyberspace are to be a matter of public record. Furthermore, they should be readily available, through the network itself, to anyone who might have an interest in them, especially potential implementors. Copying and redistribution of the documents describing these standards should not be inhibited by copyright or license entanglements.

Open standards review, certification, update and publication process [○ Inst] — Once they become established, the process by which the Cyberspace protocol

standards evolve should be open to interested parties. We explicitly cite the Internet protocol engineering process as a model for this.

Benign regulatory environment [○ 🌐 🏢 \$ 🛡️ Inst] — As far as possible, the implementation of Cyberspace should attempt to avoid the entanglements of any form of government regulation of telecommunications that would undermine any of the goals or features outlined here. In particular, it should be an explicit objective to avoid reliance on any form of government subsidy or support in order to proceed. Governments, to the degree that they will permit, should be treated like any other user of the system. To do otherwise risks dangerous intervention in the Cyberspace development process, since the goals of openness, decentralization, and security often conflict with the agendas of many governments. In an imperfect world, the interests of governments are not necessarily coincident with those of their citizens or of the international community.

A Decentralized system requires...

Peer-to-peer client/server relationship [* ○ 🏢 \$ Tech] — The distinction between a server object and a client object is simply which object at some particular time issues a request for service and which object responds to the request. Object A may be a client to Object B but a server to Object C. Two objects might mutually be both clients and servers of each other. The protocols themselves recognize no hierarchy of clients and servers.

No “superuser” [* 🛡️ Tech] — There exists no mechanism for global privileges that would allow an entity to override security barriers or resource allocations in a general way. It may be the case that the operators or administrators of a particular subset of the network have such privileges with respect to those parts they are responsible for, but these privileges will not function outside a limited sphere of influence.

No global state information [* 🌐 🛡️ Tech] — There is no information that must be shared globally across the entire network in order for it to function. Note that the protocol standards themselves are potentially a form of global information. Thus one corollary to this feature is that the protocols are not guaranteed to be global standards; all that is required for interoperability is that the communications route from one point to another perform any necessary protocol translation. As a practical matter we expect much of the protocol suite to in fact be global, as a de facto standard if nothing else. All this feature states is that this cannot be mandatory.

No monopolistic administrative authority [* 🌐 ○ 🏢 🛡️ Inst] — There is no governing body, in either the public or private sectors, with jurisdiction over the entire network. This is the institutional analog to the “no superuser” feature stated above.

No required proprietary components [* 🌐 ○ 🛡️ Inst Tech] — Protocol designs cannot incorporate elements which are subject to intellectual property restrictions that would limit their ability to be freely distributed or freely implemented. Note that we do not require that particular implementations be free of proprietary components. Rather, we simply mean that the protocols should be defined so that no proprietary components are essential to their implementation.

A Traversable system requires...

Common format for messages and data types [:- ○ \$ Tech] — The protocol standards include a common format for messages between objects, including how requests for services are to be stated. In particular, the mapping of low-level data types (integers, characters, etc.) to a transmittable binary representation must be stated as part of the standard.

Distributed transportable objects [:- ○ * \$ █ Tech] — The protocol standards define an object model specifying the computational semantics of distributed objects, so that these objects may be spread between machines and transmitted from one place to another. This includes a procedure for an object entering a new system for the first time to declare the capabilities that it requires to operate and to negotiate for additional capabilities that it might desire to make use of.

Object persistence [:- * \$ █ Tech] — The objects that are being passed around from machine to machine need to have persistent state that is semantically independent of any particular machine that represents them. In other words, if a machine containing an object should crash, when the machine comes back up again, the object should still be there, in the same process state as before. This is accomplished by a combination of fault-tolerant backup storage techniques and distributed representation so that no particular machine is critical.

A Commercial system requires...

Digital money [\$ * █ █ Tech] — The protocols include mechanisms for handling all forms of financial instruments electronically, including but not limited to credit cards, checks, cash, stocks, bonds, options, and any other form of negotiable or non-negotiable security. It should be possible to operate a full-service financial institution entirely within Cyberspace.

Banking, credit, and other financial services [\$ * █ █ Inst] — Cyberspace will have financial services institutions operating within it. These are necessary for its normal functioning. In addition to the technical elements required to support this (the “digital money” feature just mentioned), a successful financial services sector needs all the same kinds of non-technological standards as are found outside Cyberspace. It also needs conventions for interaction between the financial services sector within Cyberspace and the one outside, especially the regulatory environment of the various jurisdictions in which Cyberspace operates.

Electronic credentials [\$ * █ █ Tech] — The protocols must include a full-featured credentials mechanism, so that reputation information about an entity inside Cyberspace, be it a person, a company, a computer, or a transmission pathway, can be shared, tested and validated. It should be possible to determine what a particular credential asserts and does not assert, who stands behind it, and what standards were applied in its generation. It should also be possible for entities in the network to voluntarily participate in the collection of information concerning their transaction history, so that documentation for such credentials can be compiled and authenticated.

Reputation services [\$ * █ █ Inst] — A reputation service provides a repository for credential information, and various degrees of certification and testing to validate its quality. Examples of such services outside Cyberspace include credit bureaus such as TRW or Equifax, as well as institutions such as Underwriter’s Laboratory, Consumer’s Union, the Good Housekeeping Seal of Approval, and so on. All of these types of

services will have analogs within Cyberspace. Such services are an integral component of the commercial and social elements of the Cyberspace society.

Product/service directories [§ ○ * 🗄 Inst] — A directory service provides a mechanism for products, services and users of Cyberspace to make their presence known to others who might be interested in communicating with them but who might not otherwise have any means of learning of their existence. It should be possible for a service provider to register with various directories, publishing its address, a list of the services it is offering, specification of the interfaces necessary to make use of those services, and the terms (pricing, etc.) under which they are offered. Directories themselves are services, so that the directory structure in general may be a hierarchy, with a few, well known “directories of directories” feeding users out to a large number of other directories, each with a more specialized emphasis.

Arbitration services [§ * 🗄 🗄 Inst] — In any environment where there is exchange of goods and services between people, or any kind of interaction between them at all for that matter, disputes will inevitably arise. Some mechanism is required for the resolution of these disputes. This mechanism should impose an extremely low cost on normal (i.e., undisputed) transactions, and should still be relatively low cost in cases where disputes do arise. It should enable people get on with their business with a minimum of disruption, yet provide real remedies in the case of material breaches of contract or other harm. Arbitration services can fill this need and as such are a critical institutional element of Cyberspace.

A Social system requires...

Sense of place [🗄 ○ * :- \$ Tech] — The model of the Cyberspace world that is presented to the users is of an independent universe with places and objects that have an objective existence separate from the various participants themselves. Thus, if you leave an object somewhere, it should still be there when you come back later, unless of course somebody else has come along in the meantime and carried it away. If you meet someone in Cyberspace, you meet them in some specific location that can be identified and returned to in the future. Places are connected, and people and objects can move through these connections at will, subject, of course, to the security constraints and other house rules of any of the particular places involved.

Inhabited world [🗄 ○ * \$ Inst] — Though it may seem redundant to emphasize this point once again, we must not forget that there are people here who can interact with each other. The principal purpose of Cyberspace is to connect people to people, rather than simply connecting people to databases (the latter function is already handled more or less adequately by the existing services). Cyberspace without interpersonal interaction is a dead and lonely place.

Multinational language support [🗄 🌐 ○ * :- \$ Tech] — Cyberspace data formats should adopt one of the emerging international character set standards for text types, allowing all languages to be handled transparently.

Language translation services [🗄 * Inst] — In a multilingual environment, language translation is an essential service. Telephone companies already provide this service for voice communications. Similar services should be readily supportable for text, both in real time (to support conversation) and in batch (to support document conversion). Such services will benefit from a standardized protocol for requesting translations.

Bill of Rights continuity [† * █ Inst] — In the USA we have a Bill of Rights as part of our Constitution, the highest law of the land. The Bill of Rights guarantees certain fundamental freedoms to citizens of the USA, protecting them from various egregious abuses by their government. Although the USA has one of the strongest bills of rights of any country, most nations of the civilized world have some kind of analog in their own laws. These rights should continue to be recognized in Cyberspace. Important rights found in the US Constitution that are applicable in this environment include freedom of speech, freedom of the press, freedom of assembly, freedom of worship, freedom from unreasonable search and seizure, the right to due process, and the right to equal protection under the law. Part of the effort of deploying Cyberspace will have to include participation in the ongoing campaign to insist that the US Government recognize these rights in Cyberspace as well as the physical world, and for similar recognition of corresponding rights in other countries as relevant.

Although Cyberspace will be governed by the laws applicable in each of the jurisdictions it intersects with, in many ways it is outside all physical jurisdictions. For example, when a user in Canada purchases a service from a vendor in the United States using a server in England, a bank in Switzerland and a communications pathway routed through Germany and France, where has the transaction taken place and which country's laws should govern it? Who gets to tax it? Eventually the various nation states will sort this out, but in the meantime Cyberspace must continue to function regardless.

A Secure system requires...

Link encryption [█ Tech] — Many of the transport services that Cyberspace will be built on top of are inherently insecure, in that communications pass through computers that provide no protection against eavesdroppers or persons tampering with the communications flow. Consequently, the Cyberspace protocols will include a link encryption feature to ensure end-to-end communications privacy.

Identity and message authentication [█ * † \$ Tech] — The protocols will provide a mechanism for authenticating the content and sources of messages. This will ensure that messages cannot be forged or tampered with, and that contracts that are digitally signed cannot be repudiated. It will enable continuity of relationships, even when two parties each have no idea who the other is, since they will be able to verify that a series of communications all originate with the same sender.

Secure capability semantics [█ :- Tech] — The fundamental primitive for controlling access to resources, to services, to features and to information will be capabilities. Capability semantics are a well understood formalism for the management of trust relationships which will be built into the foundation of the Cyberspace protocols.

Identity certifiers [█ * † \$ Inst] — An identity certifier is a service that validates the binding between some address or user name and an accountable entity of some sort, be it a real-world person or company or simply another address or user name which is more highly trusted. These services can certify a range of additional attributes besides identity, including such things as credit ratings, physical addresses, appearances, and so on.

Object behavior certifiers [█ * \$ Inst Tech] — An object behavior certifiers are services that are important for supporting the “distributed transportable objects” feature. These services validate the behavior of objects, subjecting them to intense scrutiny, to certify that they meet the interface contracts which they carry on their outsides. Such

services provide a reputation-based mechanism for ensuring that objects provided by others can be incorporated into one's own system without significant security concerns.

Recognition of user right to privacy [🔒 * 🗑️ Inst] — The Cyberspace protocols will contain no “trapdoors” in their security measures. That is, there will be no mechanism provided to enable the override of cryptographic secrecy by authorities. Whereas in the physical world, any so called “right” to privacy is a privilege granted by the state and subject to occasional violation and some controversy, in Cyberspace it can be a matter of technical fact, provided that the requirement for it is recognized at the time the protocols are designed. This feature explicitly recognizes this requirement as part of the design process.

A Portable system requires...

Bandwidth independence [📶 Tech] — Protocol designs will be independent of bandwidth. That is, they will be considered in low-, medium- and high-bandwidth contexts during the design process. The basic protocols should function in a situation of minimal communications capacity, while they should be able to take full advantage of maximal capacity if it exists. The limitations imposed by bandwidth should be limitations only on the assortment of services that are available. Actually, we will consider all aspects of communications performance in this analysis, including not just bandwidth but also latency, packet size, reliability, and so on.

Transport medium independence [📶 ○ Tech] — The Cyberspace protocol designs will be independent of the transport protocols and other peculiarities of the underlying communications medium. The same arguments and analysis will apply here as above in the description of the “bandwidth independence” feature.

User interface independence [📶 ○ 🗑️ \$ Tech] — The Cyberspace protocols will not be based on any particular user interface model or display metaphor. In particular, they should be capable of supporting environments based on text or graphics; based on 3D rendering or flat displays; based on a document model, a “virtual world” model, or something else. The interfaces of current online services, Internet utilities, and other interactive products should all be compatible with the Cyberspace protocols.

Organization Of Features

The 39 features given above vary widely in their subject matter, degree of technicality and level of specificity. In this section we will structure this large collection into five groups, depending on the way they influence the protocol definition process, so as not to be wallowing in an undifferentiated mass of constraints as we go on to define the protocols themselves. These groups will drive the protocol configuration that will follow in the next section.

The features can be broadly classified as process constraints, environmental constraints, protocol constraints, problems to be solved, or services. Each of these categories applies to a different facet of the protocol design, implementation and deployment process.

Process constraints

These features are constraints on the protocol development process itself. They specify aspects of the way we must go about creating Cyberspace or things that we must do in the process, rather than determining the form it will ultimately take when it is done. These features include:

- Built on top of existing standards where applicable
- Published protocols with unlimited distribution
- Open standards review, certification, update and publication process

Environmental constraints

These features are constraints on the institutional environment which Cyberspace needs to exist. They are difficult and possibly controversial, as they define what is essentially a political agenda in what is otherwise an entirely commercial enterprise. However, these features define important aspects of the social milieu of Cyberspace that are essential to its healthy functioning, and so they must be pursued despite the difficulties. These features include:

- Benign regulatory environment
- Bill of Rights continuity

Protocol constraints

These features are constraints on the design of the protocols. While they do not specify what the protocols must be, they do specify attributes that these protocols must possess in order to be acceptable for our purposes. You could say that these are litmus tests for our designs. This set of features can be used as a checklist in the evaluation of any prospective protocol. These features include:

- Multipath network topology
- Algorithmically tractable message routing
- Many-to-many communications model
- Fully connected logical network topology
- Peer-to-peer client/server relationship
- No “superuser”
- No global state information

- No monopolistic administrative authority
- No required proprietary components
- Inhabited world
- Recognition of user right to privacy
- Bandwidth independence
- Transport medium independence
- User interface independence

Problems to be solved

These features represent protocol elements that must be designed. They are actual things that specific protocols need to do. One or more protocols will implement each of these features directly. These features include:

- Global object and service addressing scheme
- Extensible choice of media formats
- Dynamically extensible object environment
- Common format for messages and data types
- Distributed transportable objects
- Object persistence
- Digital money
- Electronic credentials
- Sense of place
- Multinational language support
- Link encryption
- Identity and message authentication
- Secure capability semantics

Services

These features are services which must be present in the Cyberspace environment in order for it to function. They represent businesses that we must either start or encourage. They also represent protocol requirements, in that each of these services needs a standard interface to its core facilities. These features include:

- Banking, credit, and other financial services
- Reputation services
- Product/service directories
- Arbitration services
- Language translation services
- Identity certifiers
- Object behavior certifiers

Protocols & Standards To Implement Features

The feature set organization just given makes it relatively easy to see what must be done. In this section we will outline a specific collection of protocols and other standards that implement our target set of features. We will begin by explaining the overall approach that we are taking to structure this family of protocol standards. Then we will present the protocols themselves, in three levels that we have labeled *foundation*, *ground floor*, and *superstructure*. Finally, we will discuss some specific projects that are necessary components of this protocol effort but which themselves are not protocols or standards per se.

Approach

A collection of protocols and standards that is as complex and ambitious as the Cyberspace Protocols requires some overall organizing principles if it is not to degenerate into a chaotic jumble. Here we will describe aspects of this overall approach which unify the collection.

Sound and unified formal semantic basis — The protocols are all to be built on a common, clean semantic base — the Joule computational model. This is to enable us to analyze the semantics of the protocols themselves, so that we can have confidence that they do what they are supposed to do and don't do what they are not supposed to do. It will help us to ensure that the pieces fit together reliably. It will also allow us to give clear and rigorous specifications, enabling the protocols to be readily implemented as well as enabling us to more readily verify that the implementations match the specifications.

Simple protocol abstraction layering — The protocols are grouped into just three layers, which we refer to as the *foundation*, the *ground floor*, and the *superstructure*, respectively. We will describe below what these consist of, specifically, once we get into the actual protocol descriptions.

In any given layer, protocols assume the presence of all protocols from the layer beneath. In addition, they may have dependencies on specific other protocols in the same layer (since certain protocols are designed to work together). However, in no case does a protocol make reference to higher layer protocols.

This three-tier layering structure was deliberately chosen to be simple. It would have been feasible to more finely differentiate the hierarchical relationship between the protocols; it is common in the world of communications protocol design to do exactly this. However, in the case of the Cyberspace Protocols we concluded that such extra gradations would only introduce added complexity with little benefit in return, other than pedantic purity. Furthermore, it has been our experience that excess levels of protocol tend to introduce inefficiencies in the ultimate implementation. Such inefficiencies are typically overcome in practice by breaking the clean modularity boundaries that the layer structure imposes. These are the very boundaries which we are here struggling valiantly to erect in the first place. In other protocol families, such modularity violations are merely opportunities for bugs and problems in software maintenance. However, our security model depends vitally on the absolute inviolability of these modularity boundaries. Because our boundaries are so rigid, it behooves us to erect them only where they truly make sense, under the assumption that they will be de facto impenetrable rather than just impenetrable on paper.

Distributed transportable world objects — A fundamental mechanism around which many of the protocols and services of Cyberspace will be constructed is a distributed

object system based on the Electric Communities distributed instantiation object model. Many of the protocols assume this system, so it makes sense to say a few words about it here. For all the details of distributed instantiation and the other particulars of the object model, consult the companion document, "The Distributed Instantiation Object Model", which is attached below as Appendix A.

Objects within this model are distributable (i.e., can spread over multiple computers on a network), persistent (i.e., exist for long periods of time, regardless of whether or not the computer(s) they are running on are up or down at any particular moment), and transportable (i.e., can be moved from one computer to another and still continue to function). Such objects can implement essentially any virtual world model or service which someone in the Cyberspace network cares to support.

Several of the ground floor protocols are concerned with implementing this object system, while many of the remainder make use of it.

Common data formats — The protocols rely on a set of common representations for all data types. These common data formats enable us to assume interoperability from the ground up — any data that gets onto a wire is going to be understandable by whoever receives it. This also allows us to ignore the problem of format conversion in the protocol design process.

Capability security — We stated above that the fundamental primitive for security management will be capabilities. Capability semantics are a well understood formalism for dealing with the management of trust relationships. Capabilities are expressed very naturally in the Joule computational model. We will have explicit protocols for managing capabilities, so that the capability model can then be assumed when designing the security aspects of the other protocols.

Capabilities are a superior formalism for trust management than their more popular competitor, access lists. Not only are they easier to handle in a distributed environment, but they can preserve privacy in a way that access lists cannot, since access lists require tracking of identities.

Service frameworks — Core services will supported by standards which we are calling *service frameworks*. A service framework is a package of protocol standards, institutional procedures and marketplace conventions which collectively enable a particular type of service transaction. The service frameworks are all found in the superstructure level, since they rely on the rest of the protocol suite to operate.

Foundation

The foundation level provides the base on which everything else is built. It provides the fundamental syntax and semantics for interoperability. The protocols and standards at this level are not specific to our Cyberspace vision per se, although, of course, they have been chosen because they provide the underpinning that Cyberspace requires. However, they form a general purpose foundation for all manner of distributed computation.

Joule: Secure concurrent programming semantics standard — Joule is a concurrent programming system being developed by Agorics, Inc. of Los Altos, California, in cooperation with Electric Communities. Joule provides an extraordinarily clean semantic model for concurrent computation and distributed processing. Joule, in fact, was conceived with a distributed Habitat-like cyberspace system as a principal

motivating application. Consequently, it is well suited to our needs here. The Joule computational model lies at the heart of the Cyberspace protocol family.

The Joule programming language embodies this computational model directly, making it the most convenient format for expression of programs in this paradigm. However, the computational model itself is independent of the language and may be implemented, for example, via class libraries in a more conventional object oriented programming language such as Smalltalk or C++. In fact, Agorics, Inc. is currently developing such a class library for C++, designed to interoperate with applications developed using the Object Management Group's CORBA (Common Object Request Broker Architecture) interface standard, IDL (Interface Definition Language).

For details of the syntax and semantics of Joule, consult the document, "The Joule Reference Manual", which is available from Agorics, Inc.

F1: Address Generation & Resolution Protocol — This standard defines a method for allowing any object, service or other entity in the Cyberspace environment to have a unique address. Such an address can be used to direct a message to an entity, to refer to an entity in the parameters of a message, or to uniquely identify the sender of a message or the creator of some piece of information. This standard must satisfy the following requirements:

- Globally unique identifiers

No two addresses can be the same. That is, a given address must be guaranteed to always refer to the same entity, regardless of the location of the addressee, the location of the addresser, or any other context dependent attribute of anyone involved.

- Permanence

Once assigned, an address should be valid for all time; that is, it should not "expire". Furthermore, an address should not be recycled when the object to which it refers is destroyed. It may be desirable to add an alternative form of "pseudo-address" which contains an expiration date, after which it is no longer considered valid (allowing, for example, directories to be garbage collected). However, such a standard is a supplement to the more fundamental permanent address mechanism.

- Distributed namespace allocation

Assignment of addresses to new entities must be possible in a distributed fashion. That is, no central allocation authority should be required. It is not specified at this time whether this is to be accomplished by a hierarchical delegation of namespace ownership, by a random number scheme, or by some other method.

- Non-geographical organization

Since objects can move around, addresses should not be geographically organized. More specifically, they should not be required to contain information which would be used for message routing on the basis of different parts or fields of an address.

- Compact representation

The actual representation of an address should be relatively compact. That is, including addresses as destinations of or parameters in messages should not result in messages so bulky that communications performance is impaired. It is conceivable that meeting the other requirements given above will require large addresses of some kind. In this case, the standard should include a mechanism for temporary assignment and caching of “nicknames” or other abbreviated forms of address between two communicating entities.

F2: Server Interface Standard — This standard defines a method for describing the interface to an object or service in the Cyberspace environment. This standard will have two components.

The first component is a semantic model that incorporates standard primitive data types, standard methods for composing these into more complicated aggregate data types, and a way of declaring operations which take certain types as parameters and return other types as results. Our current assessment is that the CORBA IDL standard meets the needs of this component satisfactorily.

The second component is a common representation for these interface specifications. This will enable descriptions of interfaces to be shared across a network, transmitted from one machine to another, stored in directories, and so on.

F3: Primitive Data Representation Standard — This standard specifies the way that data as defined by standard F2 are to be converted to binary representation for transmission through a communications medium. Note that one form of “communications medium” is permanent storage. Thus this standard also specifies how things are to be written to disk or tape. This is important because the computer that does the writing may be different from the computer that, possibly at a much later time, does the reading. Objects must be binary-compatible across platforms.

This standard will also specify the means of encoding text so that documents written in various different human languages may be freely interchanged. Our current assessment is that the Unicode character set standard is adequate for this purpose.

F4: Interserver Message Protocol — This standard specifies the way that messages from one server to another are to be represented in transmission. Such messages correspond to the operations declarable using standard F2. The message protocol standard defines the way parameters to an operation are to be packed into the bits representing the message, as well as the way the destination of the message and the operation desired are to be encoded.

Ground Floor

The ground floor level provides the essential building blocks of the Cyberspace world itself. It specifies the things out of which the services, places and other elements of the Cyberspace environment will be constructed. The ground floor protocols build on top of the distributed computation and communications facilities provided by the foundation protocols.

G1: World Object Model — This standard is the formal statement of Cyberspace’s realization of the distributed instantiation object model described in Appendix A. All services and goods delivered through the Cyberspace marketplace are implemented by objects corresponding to this model.

G2: World Object Transfer Protocol — This standard defines a protocol for moving all or parts of an object, as defined by standard G1, from one machine to another. It specifies the representation for descriptions of an object's overall structure and of its various pieces, so that these can be placed in a message and transmitted over a wire. It provides for an interactive dialog between the entity transferring the object and the entity to which it is being transferred, so that they can negotiate which pieces need to be sent, what resources and capabilities the object being transferred requires at the receiving end, what obligations and limitations the receiver is placing on the object once it is there, and so on. This standard also specifies the protocol for transferring host authority over an object from one machine to another (the concept of host authority is described in Appendix A).

G3: Message Privacy Standard — This standard defines a method for encrypting messages sent from one object to another so that they will be secure from the prying eyes of potential eavesdroppers. This standard consists of two components.

The first component is a set of formatting rules for encrypted messages. These rules specify:

- The syntactic form of encrypted messages, so that the portions which are encrypted (i.e., the actual message contents) and the portions which are not (i.e., destination addresses and other header information required by the transport mechanism) can be separated from each other.
- How messages are to be marked so that the specific encryption algorithm used can be determined.
- How ciphertext is to be encoded so that the apparently random binary bits of an encrypted message can be passed through various restricted channels that might have difficulty with raw binary data (e.g., 7-bit ASCII serial lines).
- How individual message keys are to be encoded and embedded in messages. (Cryptographic message protocols commonly use a unique key to encrypt each message. This key is then itself encrypted with a separate key that is known in advance to both the sender and the receiver. This encrypted message key is then attached to the message that it secures, and the package is transmitted. This provides an additional layer of privacy protection and can simplify key management if public-key methods are used.)
- How entire messages (header information and all) can be encrypted inside a wrapper message (or "digital envelope") for transmission to a forwarding service. The forwarder can then open this wrapper, determine the final destination, and send the unwrapped message on its way. By nesting this process several times in succession, it is possible to route a message through a chain of forwarders (called "mixes") and thus secure the transmission against traffic analysis attacks.

The second component of the message privacy standard is a set of specific cryptographic algorithms to be used for encryption. This set is intended to be extensible, so that new cryptographic algorithms may be incorporated in the future as they are developed. The specific choice of algorithms is currently undecided, but will include, at minimum, the following sorts of things:

- A symmetric (secret-key) block cipher, such as DES, EDE triple-DES, IDEA, Khafre, or RC4, for message encryption or other applications requiring efficient encoding of large blocks of bits.
- An asymmetric (public-key) cipher, such as RSA or ElGamal, for key encryption or other applications requiring public-key semantics.
- A null cipher, which does no encryption, so that cleartext messages can be sent through channels that use cryptographic protocols.

G4: Message Authentication Standard — This standard defines a method for authenticating messages and other pieces of data. This will enable both assurance of message integrity as well as digital signatures for message source verification, credentials certification, signatures on electronic contracts, and so on. The standard will include:

- How authentication codes are to be encoded and associated with the things that they authenticate, including marking as to the signature and hash algorithms used.
- A choice of cryptographic hash algorithms, such as MD5 or Snefru, for the generation of message integrity check (MIC) codes.
- A choice of digital signature algorithms, such as RSA or DSS, for the actual authentication of data.

G5: Certificate Management Standard — This standard defines formats for certificates, which are blocks of data that contain a binding between some information (typically a public key) and an identity, authenticated by some certifier. A certificate may contain other information about the identity it certifies which the certifier wishes to publicly vouch for. It is a general-purpose credential format, although its principal use will be certification of public keys. Certificates may contain other certificates within them, typically certifying the certifier. By this means, a variety of trust models may be supported, including the Internet PEM certificate hierarchy model and the PGP “web of trust” model. The certificate format will include:

- The identity of the entity being certified.
- One or more pieces of information being certified with respect to that identity, with a standard tagging model so what each piece is can readily be determined (i.e., whether it is a public key, a credit rating, an address, etc.).
- The identity of the entity doing the certifying.
- Optional recursively embedded certificates for the certifying entity and possibly others.
- Marking as to the authentication and encoding algorithms used for the certification.

G6: Capability Management Model — This standard defines formats and procedures for the handling of capabilities. Capabilities are supported naturally by Joule, but an additional layer of protocol support can make them more useful by providing a standard mechanism for tagging them as to what they provide access to, requesting them from servers, and so on. When an object enters a new host, the object and the host engage in a dialog that lets the object inform the host of the capabilities it requires in order to function and the host to inform the object of what it is willing to allow. The Capability Management

Model provides a structure for this dialog. It also defines conventions for the subdivision of capabilities, so that the security rules imposed on any particular object in a particular place can be as fine-grained as is necessary to provide the appropriate mix of restrictions and permissions.

G7: Media Extension Standard — Information being passed around in Cyberspace can take a variety of forms, from text to audio to video to who knows what. Each of these forms needs to be encapsulated and labeled in order for its recipient to know what to do with it. This standard defines the formats for this encapsulation and labeling, so that audio, video, etc., can be shipped around as data in Cyberspace. A key requirement for this protocol is extensibility — it must provide mechanisms for the introduction of new media formats in the future, as standards and technology evolve. The intent is to create something similar to the Internet MIME (Multipurpose Internet Media Extensions) standard, but without MIME's strong orientation towards information packaged in email.

Superstructure

The superstructure level provides standards for the core services that are required to make the Cyberspace environment viable as a marketplace and as a society. At this level are found the service frameworks themselves, as well as standard components for virtual world construction.

S1: Directory Framework — This framework specifies object classes, protocols and procedures for directory services, so that entities in Cyberspace that wish to make themselves known to others can do so. Such publication of your identity may be to offer a service to the network at large, or to state interest in receiving communications on some topic, or any other purpose which you desire. In addition to supporting the publication of directory information, this framework will, of course, also provide protocols for queries of these directories, so that they can be searched for services or other objects according to various criteria.

This framework fills many of the same functions as the ISO X.500 directory services standard. However, X.500 itself is not suitable for our purposes here because of its lack of support for commercial services (for example, it is not designed to support publication of advertising), its overemphasis on email and other forms of heavy-weight messaging (rather than real-time object-to-object transactions), its excessive centralization, its relentlessly hierarchical bias towards administrative and organizational structures that are external to the network rather than inside the network, and its generally user-hostile ugliness.

S2: Minimal World Object Set — Habitat gave us a useful set of metaphors for structuring a virtual world and navigating within it. While we cannot impose a particular user interface metaphor (such as Habitat's), many of the structural components of the Habitat world are generic and will be invaluable in creating places and services. These components will be embodied in a set of object classes that will be specified by this standard. These objects provide a minimal framework for persistent places in the virtual environment, allowing servers to be interconnected and users to migrate between them. These objects will embody concepts such as places, containership, location within a place or container, connections between locations, documents, users, and so on. The various service frameworks at this level will also contribute classes to this object set.

S3: Financial Framework — This framework specifies object classes, protocols and procedures for financial activity of all sorts. In particular, it will provide for digital money

in a variety of forms, including cash. It is the framework that will be used by banks, brokers, stock exchanges, insurance companies, and other financial services organizations to conduct their business in Cyberspace. It is the framework that will be used by all inhabitants of Cyberspace to engage in any form of financial transaction.

In addition to enabling commerce between system users directly, it will also include mechanisms to enable objects to purchase services from each other and to purchase resources that they need in order to function, such as communications bandwidth, disk space, memory capacity, processor time, and so on. Agoric resource management, abstractly speaking, is fundamental to the Joule computational model, but to function as a practical mechanism in a global environment it needs actual monetary underpinnings. This framework will provide those.

S4: Credentials Framework — This framework specifies object classes, protocols and procedures for dealing with credentials and other forms of reputation information. It will make heavy use of the certificates defined by the G5 standard. It is the framework that will be used by credit bureaus, employment agencies, consumer advocates, product evaluation organizations, and other groups which deal in reputation information. They will use it to buy, sell, transmit, and otherwise structure their products.

Since the electronic environment naturally fosters a high level of anonymity and remoteness in transactions, mechanisms of accountability other than the usual ones available in the physical world must be applied. The key to making this work is the concept of reputation. Reputation is the binding of identity with history, so that information about the past behavior of an entity can be considered in the decision to proceed with future relationships with that entity. This results in incentives for objects to behave so that their histories describe behavior that matches what a potential party to a transaction with them wants to see. Generally such desirable behavior consists of things like being reliable and trustworthy, paying bills on time, producing a high-quality product, etc.

The binding of history to identity is accomplished by credentials, which are implemented in this framework using the certificates of the G5 standard. Of course, such credentials are only as reliable as the entities which certified them. The services which do this meta-certification, however, are themselves subject to reputation incentives. There will, no doubt, be credential services which certify the reputations of other credential services. These will be few in number and subject to intense scrutiny and very extreme incentives for reliability and trustworthiness.

This credentials framework will be used by all the inhabitants of Cyberspace to both check the reputation of and provide credentials to the other inhabitants with whom they deal from day to day.

S5: Server Validation Framework — This framework specifies object classes, protocols and procedures for object implementations to submit themselves to testing and validation services. These validation services take object implementations and apply various types of tests to check for various forms of desirable or undesirable behavior. Assuming that an object passes its tests, the validation service produces a credential certificate binding the object implementation to a validation statement that vouches for the object's quality, features, security characteristics, or other attributes. A machine contemplating granting some capability to the object can check it against this certificate, to determine if granting the capability would be a safe thing to do.

Such validation services are a fundamental institutional mechanism for ensuring the integrity and security of users on the network. Although the specific actions which a given

object can perform inside somebody else's system can be regulated in large measure by the selective granting or withholding of capabilities, allowing that object to take some action invariably involves trusting it to some degree, however limited. Capabilities allow trust to be managed at an arbitrarily fine level of granularity. Thus they reduce the problem of deciding whether a particular object can be trusted, since the consequences of granting a particular fine-grained capability can be more readily controlled and, as importantly, understood, than can a more general access right. Alas, capabilities cannot completely solve the trust problem because, in the general case, it is isomorphic to the halting problem and thus undecidable.

However, like the halting problem, the trust problem *is* decidable by various different tests for various limited classes of programs. Validation services can thus divide object implementations into three categories: objects known to be safe by some standard, objects known not to be safe, and objects whose safety cannot be determined. The latter, of course, will for the most part simply be treated as unsafe, although the implementor of an object may treat an "undecidable" result as signal that a different type of testing or analysis should be applied. This validation mechanism reduces the trust problem from one of trusting arbitrary objects to trusting a limited number of validation services.

These services, however, have a very strong reputation stake in providing reliable and trustworthy validations. Even a single documented failure could be enough to put a service out of business. Naturally, one would place most trust in those services which have been in business for a long time. Furthermore, these services will be placed under a spotlight by both their competitors and that inevitable population of users who are ceaselessly nervous and paranoid, thus further increasing the incentive for honesty and reliability.

For even stronger validation, an object could have itself tested by multiple competing validation services. Since each of the services which validated it would have to have been individually crooked or incompetent for the object to have been wrongly certified, having multiple validators strengthens the object's claim to trustworthiness.

S6: Contract Framework — This framework specifies object classes, protocols and procedures for the negotiation and management of contractual relationships of all sorts. It will draw heavily on our experience developing the AMiX online consulting protocols. It will also make extensive use of the S3 financial framework and S4 credentials framework.

A contract is a mutually binding declaration of reciprocal obligations between two or more parties. In addition to the niceties of contract law itself, the negotiation and signing of contracts in an electronic environment requires a set of protocols to ensure that the parties are in fact binding themselves to the same contract, that signatures cannot be repudiated, that obligations which can be objectively tracked by automated means are, and so on.

In addition to enabling contracts in the first place, the contract framework also radically reduces the transaction cost associated with establishing and maintaining a formal, contractual relationship, so that relationships of this kind can be used in a much wider variety of circumstances than is practical outside of Cyberspace. This is important in Cyberspace because many of the kinds of transactions that you would handle in a face-to-face manner in the physical world must be done remotely, often with entities who are only known to you as addresses on the network. In the physical world you would not bother with a contract for a small transaction, since the expense and inconvenience involved would be large, while a good measure of accountability and recourse is nevertheless available to you simply because you can deal directly with the other person if there is a problem. This kind of direct accountability is not available in the electronic environment, but electronic contracts can compensate for this lack by being quick, easy and inexpensive.

S7: Linguistic Framework — This framework specifies object classes, protocols and procedures for the provision and use of language translation services. It will be a key mechanism for the support of international communications and transactions.

Communications between people who speak different languages are always somewhat awkward. However, they can be greatly facilitated by the assistance of knowledgeable interpreters. Telephone companies have long made language interpretation services available to people making international telephone calls. So it should be in Cyberspace with all forms of communications. This framework supports the engagement and use of translation services in any communications channel. It will support both real-time, simultaneous translation, for conversations and high-priority messaging, as well as “batch mode” document conversion services that will accept a document in one language and later return it to you translated into another language.

Since the F3 data encoding standard dictates a common format for handling all the various human languages, users will be able to possess documents in all different languages in a single system. Translation services will thus be able to handle all these different documents readily, and need only deal with the problem of interpreting meaning.

S8: Juridical Framework — This framework specifies object classes, protocols and procedures for the adjudication of disputes between inhabitants of the Cyberspace environment.

Many aspects of contracts in the S6 contract framework are things that can be moderated directly by machines, such as automatic payments associated with delivery of goods. Others, however, such as performance or quality stipulations, require subjective judgment. In such cases, disputes will inevitably arise between people with different standards or expectations. Unfortunately, there will also, no doubt, be instances of fraud or deliberate breach, which will be properly disputed by their victims.

In the physical world, of course, these matters are often handled by the public court system. However, independent arbitration and dispute resolution services are increasingly becoming popular among businesses who wish to be able to settle matters between themselves more quickly and inexpensively than the public system allows. In Cyberspace there may be a role to be played by the public legal system. However, the role of independent adjudicators will be much more significant. This is because of the nature of the environment. First, it is transnational in scope, and will be operational before the public institutions have had time to adapt, which in the international realm happens very slowly. Second, accountability for many types of transactions will only be feasible within the electronic environment itself, since it may not be possible to connect a Cyberspace entity to an accountable person or organization outside of Cyberspace. Furthermore, such accountability will be feasible only to the degree that the entities effected have previously made irrevocable binding commitments to subject themselves to it. Consequently, any legal sanctions that might be enforced against an entity must occur within an entirely voluntary contractual framework, which necessarily means that it will be a private system.

The Cyberspace Juridical Framework provides the means for such a private system to function. It specifies the protocols for engaging adjudication services to resolve disputes, including mechanisms to give them access to the evidence required to render a judgment and to capabilities enabling them to enforce that judgment. The S6 contract framework will work with the juridical framework. It will include, for example, provisions for contracts to bind themselves to particular adjudication processes in the case of disputes.

Projects

The three levels of protocols and standards just described do not, by themselves, constitute a system. They merely define an abstract ideal that potentially could exist. Thus, as part of the Global Cyberspace effort, there is specific development work needed to actually bring the system we envision into existence — nobody is going to adopt a set of standards such as we have just described in a vacuum. Potential adopters of these standards need to be shown that they are useful and that they work. To this end, we have identified a series of key projects aimed at proving and deploying the technology and its related institutions.

Cyberspace Standards Organization (CSO) — An important institution will be an international standards body to coordinate, publish, and promote the various Cyberspace standards. We envision this as a relatively informal, unofficial, non-governmental body, modeled on the lines of the Internet Architecture Board. The IAB is the organization which oversees the Internet standards, presiding, if that is the word, over the somewhat anarchic process which has resulted in the most successful family of protocol standards in the history of the computer industry. We believe that this is a better model for deploying these standards quickly and successfully than the more plodding, deliberate and official process employed by organizations like the ISO. It may well be, in fact, that we will want to establish the CSO as part of the Internet architecture effort. This might be an effective strategy for capturing the attention and consideration of many of the people and companies whose support will be required.

Cyberspace Regulation Project — Simply by virtue of being radical changes to the existing order of the world telecommunications establishment, some of the components we have identified have definite political overtones. We have tried as much as is feasible to stay out of the political arena with this vision, but to some degree it is unavoidable. The Cyberspace Regulation Project, however, must be acknowledged up front to be a deliberately political undertaking. Hopefully it will be the only seriously political element that we will have call for.

Some of what we feel are key components of the Global Cyberspace Infrastructure float in uncertain regulatory waters. These include open competition in telecommunications services; unfettered freedom of exchange of information, goods and services; cryptographic privacy and digital money; and private jurisprudence. No doubt an astute reader will be able to find many other additions to this list in the protocols and features described above.

The mission of the Cyberspace Regulation Project will be to track trends in legislation and regulation in these critical areas, to identify policies that help or hinder the Cyberspace effort, and to conduct a campaign of public information and legislative lobbying to promote beneficial policies and to oppose harmful ones. In the United States there are already several organizations which have adopted large portions of this mission. These include the Electronic Frontier Foundation, Computer Professionals for Social Responsibility, and the Digital Privacy & Security Working Group, among others. However, there is much less activity in other countries and in transnational jurisdictions. Such activities will need to be supported and encouraged.

Joule Programming System — Agorics, Inc., the creators and developers of Joule, are pushing ahead with the implementation of a practical Joule system. However, there is much work to be done and it will require our additional support in order for us to obtain all the necessary ingredients of a full Joule programming system in a sufficiently timely fashion for Joule to be effective for us.

Implementations of protocols, models, and standards — The various technical standards must be implemented in order to be credible. A fairly extensive development effort will be required. Implementations will both prove the standards (and help us debug the inevitable mistakes in our conceptual models and specifications) and provide an actual technological base for the Cyberspace network itself. These implementations will ultimately be salable products in their own right.

Reference Backend — An important piece of software to develop will be a backend (server) that realizes the G1 world object model and S2 minimal world object set standards. This will be a reference implementation, providing a model server architecture upon which many other commercial server products can be based. It will also enable us to begin developing new services which go beyond the Infrastructure standards themselves and actually begin making use of what we have created. An important initial service, which we think will be both lucrative as a product and an important element in the evolution of Cyberspace society, will be a fully distributed and user extensible version of a system like Habitat.

Exemplar Frontend — Along with the Reference Backend, we will want a frontend (client) that makes use of it. In addition to being a full fledged product in its own right, a key purpose of this implementation is to demonstrate the capabilities of the Cyberspace technology. It is intended to be a high-profile, flagship product that will show off the various features of the Cyberspace protocols in an exciting and aesthetically pleasing way. The idea is to open people's eyes as to the possibilities and to inspire other developers to try to top it. A good analogy is the role that the original MacPaint and MacWrite applications played in the early days of the Macintosh computer. They were not the ultimate examples of those kinds of applications; in fact, they seem rather primitive and unsophisticated by today's standards. However, at the time they were amazing to behold and set a high standard for other Macintosh software to aspire to. This is the sort of role we would like the Exemplar Frontend to play with respect to Cyberspace.

Implementations of services — In order to function successfully, the Cyberspace network will require the various services whose frameworks were outlined in the above description of the superstructure level. Not only will implementations of the frameworks be needed, but actual services which use these frameworks must be made operational. Each of these is a business that someone can run. Ideally, we will convince companies which are already in these businesses (e.g., banking, software testing, dispute adjudication, etc.) outside Cyberspace to go into business inside Cyberspace, possibly with our technical assistance at the beginning. Where this is not possible, we will have to arrange for the establishment of these businesses directly, either by starting them ourselves or by encouraging venture capitalists and others to become involved in their creation.

Conclusion

We have descended from the most abstract, high-level system requirements down to a set of specific proposed communications protocols and other technical standards. The result is a roadmap for the designers and implementors of the Global Cyberspace Infrastructure. These protocols, correctly developed, will usher in a new era in digital telecommunications, in both consumer and business markets. They will fill the vast infrastructure gap that now exists between wires and content, between telecommunications providers on the one hand and information service providers on the other, by enabling anyone who wishes to create persistent places in the Cyberspace universe that they can share with others, for fun or for profit (or for both).

What this infrastructure makes possible is a division of labor that is currently beyond the reach of present day telecommunications technology. It will enable service vendors to concentrate on the business they know, providing services, rather than on becoming network operators or software development organizations. It will enable telecommunications vendors to concentrate on the business they know, providing telecommunications, rather than on trying to become all possible things to all people. The result will be to enrich both service vendors and telecommunications vendors, as a new economy blooms on the electronic frontier.

Appendix A: The Distributed Instantiation Object Model

Introduction

This document describes the Electric Communities distributed instantiation object model. This is an alternative approach to the organization of distributed objects. It differs from other approaches, such as those adopted by Sun Microsystem's DOE (Distributed Objects Everywhere) system or IBM's SOM (System Object Model) package, in that it views objects themselves as intrinsically distributed entities, as opposed to non-distributed members of a distributed collection. This view of distributed objects has evolved over a period of almost ten years, beginning with its first, crude realization in the implementation of Lucasfilm's Habitat system. Over the years our conception of this approach to object organization has become both more detailed and sophisticated as well as more deliberate and formal. This document represents a snapshot of the current state of our thinking with respect to various aspects of our model. However, it does not present an application interface schema of the sort that would be incorporated directly into a piece of software utilizing the techniques we outline here. The development of the kernel for such a schema will be one of the tasks Electric Communities will be pursuing during 1994.

A note on terminology

The term "object" has become hopelessly overloaded as a result of sloppiness and overuse. Among the many possible interpretations for this word are two related but definitely distinct meanings that are particularly important for our purposes. The first meaning is the sense in which it is generally used in object oriented programming. When used in this way it signifies a package of data and executable code that encapsulate some concept in a program. We might label this an "OOP object". The second meaning is the sense in which it is used when talking about something like the Habitat world. In this case it signifies a discrete entity that can be manipulated by the user or which has some definite role within the world. This sort of object often has some metaphorical or analogical relationship to a real-world object of some kind. A term for this might be "world object".

These two similar but distinct meanings can lead to considerable confusion, since when we are talking about the Cyberspace software architecture we will have occasion to use both senses of the term in a single piece of writing, perhaps sometimes even within a single sentence. Lest we get hopelessly muddled, some sort of terminological fixup is needed.

We could continue to use the term "object" for these concepts, qualifying it everywhere with the modifiers mentioned. However, this is verbose and clunky, and likely to prolong the confusion in any case. Another possibility would be to use "object" for one of these concepts and make up a new term for the other. This would get rid of the ambiguity but still leave ample room for confusion.

The terminological convention we will adopt, therefore, is to refrain from using the word "object" for either meaning. Instead, for the first meaning, OOP object, we will adopt the term that Joule uses for this concept, which is **server**. For the second meaning, world object, we will appropriate the word **unum**, from the Latin meaning one thing. The word "object" will be reserved for talking about objects in languages like C++ or Smalltalk or physical objects in the real world.

Distributed instantiation

The classic Smalltalk or C++ object model makes a fundamental distinction between a class and an instance. Any object is an instance of some class. A class is a template — each class is described by some sort of class definition (in Smalltalk by an explicit `Class` object, in C++ by a `class` declaration). For any given class there is a single class definition. Instances, on the other hand, may exist in profusion. Each instance is described by the class definition for its class. Class definitions contain information which is invariant across instances. This includes the code that implements the executable part of the class's semantics. Class definitions, in fact, are to a large degree code entities. Instances, on the other hand, contain state information which can vary from instance to instance. Instances are thus for the most part data entities.

However, implicit in this model is an unspoken and largely unconscious assumption that this all takes place in a unitary, homogeneous environment. It also tends to assume that individual objects are simple entities with simple structure and simple function. More complex structures and functions are expected to be obtained by the functional composition of multiple objects. This model is certainly appropriate for what we are now calling servers, wherein the encapsulation is principally an abstraction to aid software engineering. For what we are now calling *una* however, this model fails in a number of ways.

Una exist in a distributed, heterogeneous environment. Considering this environment as a whole, the model just stated is essentially correct. However, a programmer or a piece of software never encounters the environment as a whole but only pieces of it represented in particular computers. The big architectural question is how to divide up the world so that it can be distributed across multiple machines. This is an important problem because the easy or obvious solutions tend not to scale well, but scalability is a critical requirement for Cyberspace.

The approach we will use is derived from the original Habitat model: the world is divided into discrete locations called **regions**. Each region represents a place in the world. For now, you can think of a region like a room, though we have worked out a generalization that also lets us support continuous open space. (This generalization is outside the scope of this document, however, and we don't need to get into it in any case in order to understand the basic principles that we are trying to work out here.)

The model of a single computer, running the code that implements a server, communicating over a network to a different computer, running the code that implements a different server, is easy to visualize. However, it should be pointed out that there is essentially no difference between this situation and that of two mutually untrusting but communicating processes running on single, time-shared CPU. On the other hand, a closely-coupled multiprocessor might be the "machine" that runs the code for a single server. We will refer to the entity which runs a server, whether it is one machine or many, one process or several, as a **machine**. In general, when we talk about communication between machines we mean any type of communication across a trust boundary, and when we talk about the actions of an individual server we mean the processing and communications which stay within a particular trust boundary.

Each region has a number of machines which are said to be **involved** with it. One of these is considered to be the **host** for the region; the remainder, if there are any, are considered to be **participants**. Each region has exactly one host and zero or more participants. In general, there can also be an upper limit on the number of participants allowed for a particular region, but rather than being a fundamental semantic issue, this

upper limit is determined by pragmatic factors such as communications bandwidth and host computational capacity.

Each region contains a collection of unum. A representation of this collection is found at each of the machines involved with the region. We say that a machine involved with the region containing an unum is involved with the unum as well. Alternatively, we say that an unum has a **presence** in that machine. As with the region itself, one of the machines involved with each unum is considered to be the host for that unum, while the rest of the machines are considered to be participants. Note that the list of involved machines is the same for every unum in a region, but that the particular machine from this list that is the host for a particular unum may vary from unum to unum. An important corollary to this is that the host for a region is not necessarily the host for any particular unum in it, though in practice it often will be.

Like objects in Smalltalk or C++, each unum is an instance of some class. Considering the distributed environment as a whole, each unum has a single, invariant class definition that describes it. However, this class definition might not be found in its entirety at any of the machines involved. Instead, each machine possesses a subset of the class definition suited to its platform architecture and its particular role (host or participant) with respect to the unum. This subset is called a **partial definition**. Thus, for a given unum, we find differences between the partial definitions possessed by the host and participant machines, as well as differences among the participant machines themselves if they are implemented on different platform architectures. Similarly, if the unum moves from a region hosted by one machine to a region hosted by another, the partial definitions possessed by the two hosts may also differ.

A partial definition for an unum class is made up of a collection of resources called **parts**. Each part can be executable code, literal data, or a mixture (i.e., a quasiliteral). Which parts go into a particular partial definition depends on the role and platform of the machine involved. Further differentiation between machines is possible too. A particular machine might wish to retain parts for platforms or roles other than its own so that these will be available for transmission to other machines that it might come in contact with. On the other hand, it might lack certain parts that are used infrequently, obtaining them from some remote archive only if they become needed.

Just as the class definition for an unum may vary across machines, so may the data structure describing the state of an unum instance. Each machine keeps track of two types of state information for each of the unum that have a presence on it, the **joint state** and the **local state**. The joint state is the same (semantically the same, though not necessarily bit-for-bit identical) at all machines involved with the unum and typically represents aspects of an unum related to its world semantics. The local state is known only to the particular machine in which it is found and can vary widely from machine to machine. It typically represents information related to screen display or other housekeeping functions, though in the case of a host it may also contain internal state that is withheld from ordinary users (e.g., for "black-box" objects).

Any machine may manipulate, in any way it chooses, the local state of any unum that has a presence on it. However, the host for an unum is the ultimate arbiter of manipulations to the joint state (this is what we mean when we sometimes informally refer to the host as being the "reality server" for the unum). More specifically, we can't stop a machine from inappropriately altering its copy of the joint state, but, if it does, the copies of this state at the other involved machines will be unaffected. As a result, the offending machine becomes desynchronized from everybody else. Since this can only hurt the machine who does this, in a sense we don't care. All we require (and all we are actually capable of

requiring) is that “official” changes to the joint state be mediated by the host. The host maintains the “actual” state and is responsible for making sure that all the other machines know if this state changes. This responsibility may be fulfilled in one of three ways, depending on the nature of the state change:

- Common knowledge — if some event occurs which is commonly known to effect the joint state in some deterministic way, each machine involved may simply note the effect in its copy of the joint state, confident that the other machines will have done the same, without necessarily requiring an explicit message from the host to inform everyone of it.
- Explicit notification — the host transmits a message to all the machines involved informing them of a change. Such changes typically result from the processing of requests directed to the host by the various individual involved machines.
- Delegation — the host issues a capability to a participant machine that enables that participant to change the joint state in some way. This participant broadcasts a message to all the other machines involved informing them of a change. The host and the other participants choose to accept (or reject) this change by validating that the sender indeed has the requisite capability to do what they say they did. This is an unusual mode but may be useful, in particular, when a state change involves a large number of bits. It may be easier and more efficient to simply broadcast the bits from their point of origin rather than having to relay them through the host.

Messaging

In the Smalltalk/C++ model, since an object is just an object, there is basically only one form of message passing: from one object to another object (an object can also send a message to itself, but while compilers might treat this as a special case, it is not a special case from the point of view of the language semantics). Distributed instantiation is more complicated, however. In distributed instantiation, the analog to a Smalltalk/C++ object is the unum rather than the server; consequently the analog to messages between objects would be messages between una. However, due to the distributed nature of una this is not a reasonable abstraction to be talking about, since there exists no unitary entity to either send or receive messages. Messages can only be sent and received by servers, which are the elements that make up una. New complications arise because a number of distinctions now become relevant which previously were not part of the model or could be ignored:

- Sender is a host vs. sender is a participant
- Receiver is a host vs. receiver is a participant
- Sender and receiver are on the same machine vs. different machines
- Sender and receiver are elements of the same unum vs. different una

There are thus fifteen cases that we need to consider, which are summarized in the following table:

Case	From	To
1	host(A, X)	host(A, X)
2	host(B, X)	host(A, X)
3	host(B, Y)	host(A, X)
4	part(A, X)	host(A, X)
5	part(A, Y)	host(A, X)
6	part(B, X)	host(A, X)
7	part(B, Y)	host(A, X)
8	host(A, X)	part(A, X)
9	host(A, Y)	part(A, X)
10	host(B, X)	part(A, X)
11	host(B, Y)	part(A, X)
12	part(A, X)	part(A, X)
13	part(A, Y)	part(A, X)
14	part(B, X)	part(A, X)
15	part(B, Y)	part(A, X)

where A and B are una and X and Y are machines. We denote the participant presence of «unum» on «machine» as `part(«unum», «machine»)` and the host for «unum» on «machine» as `host(«unum», «machine»)`. Note that there are fifteen cases rather than sixteen because the logically orthogonal case of a message from `host(A, Y)` to `host(A, X)` cannot occur since a given unum only has one host and thus this case is impossible.

We denote the transmission of a message as `send[«msg», «sender», «receiver»]`, E.g., `send[M, part(A, X), host(A, Y)]` indicates the sending of a message M from the participant presence of unum A on machine X to its host on machine Y.

Since una are not unitary objects that can send and receive messages, we can't speak of an unum's message interface in the ordinary sense. We *can* talk about host interfaces and

participant interfaces. Even these, however, need to be qualified, since the interface between, for example, $\text{part}(A, X)$ and $\text{host}(A, X)$ is different from the interface between $\text{part}(B, X)$ and $\text{host}(A, X)$, which is in turn different from the interface between $\text{host}(B, X)$ and $\text{host}(A, X)$. Thus we can't talk about *the* host interface for A. Instead, we end up with a series of "sub-interfaces" that have a lot of overlap but which are distinct. Each of the fifteen cases has its own role to play in implementing the extended cyberspace, and each has its own set of relevant trust and security issues. These security matters will be considered in more detail in a section of their own, below.

In the course of maintaining a consistent representation of the joint state of an unum among all the involved machines, it will sometimes be necessary for a host or participant to broadcast messages to all participants, or to all participants save one (the latter case being when one particular participant is handled differently, e.g., when it is the participant that requested some operation from the host, the result of which must be broadcast to everyone else). These cases are sufficiently important that we will explicitly add them to our repertoire of message patterns:

Case	From	To
16	$\text{host}(A, X)$	$\text{part}(A, *)$
17	$\text{host}(B, X)$	$\text{part}(A, *)$
18	$\text{part}(A, X)$	$\text{part}(A, *)$
19	$\text{part}(B, X)$	$\text{part}(A, *)$
20	$\text{host}(A, X)$	$\text{part}(A, *-Y)$
21	$\text{host}(B, X)$	$\text{part}(A, *-Y)$
22	$\text{part}(A, X)$	$\text{part}(A, *-Y)$
23	$\text{part}(B, X)$	$\text{part}(A, *-Y)$

where $\text{part}(\langle\text{unum}\rangle, *)$ denotes the participant presence of $\langle\text{unum}\rangle$ on all machines involved with it and $\text{part}(\langle\text{unum}\rangle, *- \langle\text{machine}\rangle)$ denotes the participant presence of $\langle\text{unum}\rangle$ on all machines involved except $\langle\text{machine}\rangle$.

Since these message operations may be composed from the message operations in cases 1 through 15, they do not introduce new trust or security issues. However, they do introduce new protocol and performance issues since they imply, in effect, adopting multicast messaging as a low-level primitive (and depending on the mechanism used to exclude non-addressed receivers from seeing multicast messages, there may actually be some security issues after all, at least at the implementation level).

Host Authority

We stated above that the host is the final arbiter of the joint state of an unum. However, this glosses over some key issues regarding state changes and who has the right to make them, which we will now delve into in greater detail.

Some of the state attributes that an unum might possess are strictly internal to that unum and do not pose any semantic difficulty. For example, a combination lock has a combination which unlocks it. This attribute does not depend on any information that is external to the unum nor is it of direct consequence to the environment in which the unum resides — though, of course, there may be substantial indirect consequences, depending, in this example, on what the lock secures access to! However, other attributes are highly context dependent. The archetypal example of such an attribute is an unum's location within a region. Another, more subtle, but profoundly important, context dependent attribute is the indicator that determines which of the involved machines is the unum's host.

Let's consider location for a moment, since it makes a good proxy for almost all attributes of this sort. An unum's location would appear to be an attribute of the unum itself, rather than of the region that contains it. On the other hand, it would seem to defeat much of the purpose of having the region in the first place if any unum could unilaterally declare where it wished to be. The region, it seems, ought to be the arbiter of the unum's location. On the third hand, simply giving the region total control over the unum's location seems like ceding too much authority in the other direction. An unum's other attributes may figure into the location calculation in a way that only the unum itself can fully understand. For example, an unum might represent a vehicle of some sort with an engine and control mechanism that constrain its movement characteristics, or it might have mass that requires energy to move, or it might be glued to the floor. All of these are special cases that ought to be handled by the unum itself since the region cannot reasonably be expected to model all possible mechanisms for changing an unum's location. On the fourth hand, something external to the unum (say, a magic wand of teleportation) might be able to alter the location in violation of whatever internal rules govern the unum's normal operation, so the region clearly requires some power to override the unum's internal mechanism. It seems that as we analyze the situation, this process of asserting control back and forth between the region and the unum can go on forever with no clean basis for termination.

The first step in resolving this conflict is to distinguish between who proposes a change and who approves it. A reasonable principle is that anybody can propose a change. Since there will already be a control and permission mechanism in place — that is what we are designing here, after all — no harm can come from accepting requests as broadly as possible. Bogus or unacceptable requests will simply be rejected. This provides maximum flexibility in that changes may originate with any actor in the overall system.

The second step is to separate permission from computation. Figuring out what the location should be is different from actually changing the location. The computation of location can be viewed as a service that the entity with ultimate control over location can subcontract to the entity who has the model that ought to be used.

The third step is to observe that, regardless of our preferred formal semantics, the host for a region and the host for an unum must concur on any outcome which affects them both. The two entities are effectively peers because of the nature of the communications channel that links them together. The host for a region can always reject or disconnect any unum that does not work the way it chooses. Similarly, the host for an unum can avoid or disconnect from any region that behaves in a way that it finds unsatisfactory.

The means for dealing with these sorts of context-dependent attributes is thus the following: the location of an unum within a region is part of the unum's joint state. That is, it is an attribute of the unum rather than the region. The unum's host possesses a capability that enables it to change this location attribute (as well as the rest of the joint state). This capability is called **host authority**; possession of this capability is what makes a particular machine the host. Note, however, that all communications among machines involved with the region and the una in it are (logically) routed through the region's host. Thus the host for the *region* possesses the capability by which changes in the unum's location may be communicated to the other effected parties. Thus, although an unum determines the ultimate objective truth concerning its own joint state, the region controls who is allowed to know about this ultimate truth.

This scheme has a couple of notable consequences. First, the region cannot arbitrarily set the unum's location to whatever it wants, since it lacks host authority over the unum. Even though the region's host is the communications nexus between all the involved machines, it

cannot spoof a location change by pretending to everybody else that the unum's location is whatever it wants, since such changes need to be authenticated by the unum's host. Thus we can see that host authority is not so much the right to fiddle with the joint state, but the right to be believed by the other machines involved when you say what the joint state is. If the region wants to change the unum's location it must request the unum's host to do so on its behalf; the unum, of course, can refuse this request if it finds it unreasonable.

Second, the unum cannot behave arbitrarily either, since the region can disconnect it at any time, refusing to pass a location change message to the others if it considers the location change inappropriate.

Consider how this works in practice. Location changes can originate internally or externally. That is, an unum can autonomously move itself or it may be acted on by outside forces. These outside forces, in turn, may take the form of communications from external entities to the region or to the unum. Assuming an unum T with its host on machine X, a region R with its host on machine Y, and an external entity E with a host or participant presence on machine Z, events thus proceed something like this:

Origination

- (1) Region generates and sends request to unum —
`send[Request, host(R, Y), host(T, X)]`
- or-
- (1) External entity generates and sends request to region —
`send[Request, part(E, Z), host(R, Y)]`
- (2) region vetoes or approves
- (3a) if approved, region relays request to unum —
`send[Request, host(R, Y), host(T, X)]`
- (3b) if vetoed, region informs requester —
`send[Veto, host(R, Y), part(E, Z)]`
- or-
- (1) External entity generates and sends request to unum directly —
`send[Request, part(E, Z), host(T, X)]`
- or -
- (1) Unum autonomously generates internal request to itself

Completion:

- (1) Unum acts or doesn't
- (2) unum informs region —
`send[Reply, host(T, X), host(R, Y)]`
- (3) region vetoes, modifies or approves
- (4) region informs unum —
`send[Info, host(R, Y), host(T, X)]`
- (5) unum vetoes or approves region's veto or modifications
- (6) unum informs region —
`send[Info, host(T, X), host(R, Y)]`
- (7a) if both approved, region informs external entities
`send[Info, host(R, Y), part(T, *-X)]`
- (7b) if either disapproved, connection between region and unum is severed.

In principle, messages can flow back and forth between the unum host and the region host an arbitrary number of times, as vetoes or modifications are offered. That is, steps (3) through (6) can repeat indefinitely. In effect, the region and the unum negotiate on the

outcome, with connection breaking as the final result if the negotiations fail to converge on a mutually agreeable settlement. In practice, such negotiation must take place between machines in the context of a predefined protocol, so endless message exchange is not a realistic possibility — in real systems the conditions which force communications termination will likely have broader scope than is formally necessary in the abstract model. In addition, the need to support real-time or pseudo-real-time interaction also limits the amount of back and forth correspondence which is practically feasible.

From the above model we can see that one of the trickiest aspects of this interaction is the transfer of host authority from one machine to another. Note that the difficult thing to transfer is not the right to change the joint state — any machine can change its copy of the joint state. Rather, the difficult thing to transfer is the credibility associated with being an unum's host — every machine involved must cease to consider the old host as the final arbiter and start believing the new host. This entails some subtle authentication handshaking.

More on capabilities

As we mentioned, there are four attributes which together distinguish the fifteen basic message passing cases. Another way of thinking about the message protocols of host and participant servers is to think of there actually being singular host and participant interfaces, but with the set of features accessible through these interfaces being regulated by capabilities that can be possessed by the respective servers. In essence, two servers can make various provable assertions to each other about the state of their mutual relationship, and these assertions can then regulate the interface these servers present to each other. These capabilities collectively enable the kind of multi-faceted message passing model described above.

A `Machine` capability is granted to each server by the particular machine in which it resides. Each machine in the network grants a distinct `Machine` capability. This capability allows a receiver server to securely verify that the sender server is or is not at the same machine as it is.

A `Host` capability is possessed by host servers and can be used by them to prove to participant servers that they are in fact hosts. Similarly, a `Participant` capability is possessed by participant servers and can be used to prove this fact to hosts (and to each other). Note that `Host` and `Participant` capabilities are mutually exclusive.

An `Instance` capability is possessed in common by all presences of a particular unum instance and can be used by them to prove to each other that they are presences of the same unum. A distinct `Instance` capability exists for each unum instance.

It may also be useful to define a `Class` capability that is shared in common by all servers that are presences of instances of a particular class, regardless of whether they are hosts or participants. This does not affect the message passing case rules but may be important for the operation of certain kinds of una.

Appendix B: Glossary

In this document, we use certain words in ways that are either more specialized than common practice would dictate or which emphasize particular shades of meaning more than is the norm. Eventually we would like to develop new terminology for the cases where our usage differs in a confusing way from conventional usage. For the time being, however, we provide this glossary to explain our particular spin on the meanings of certain important terms. In some cases we have already introduced new terms, and these are explained here as well.

Backend — A server which implements the host semantics of a region in the distributed instantiation object model.

Capability — A token, key or address which enables an object to have access to particular resources or privileges.

Client — An object which makes use of a service provided by a server.

Credential — An authenticatable document in which a certifying agency associates historical information with an identity.

Cyberspace — The world inside the global telecommunications network.

Distributed instantiation — A model of distributed object computation in which object instances are themselves distributed entities.

EDI — Acronym for “Electronic Data Interchange”, usually applied to one of a number of archaic protocols for contractually pre-arranged communication of specialized information between computers belonging to various business entities.

EFT — Acronym for “Electronic Funds Transfer”, a specialized form of EDI between banks.

FINE — Acronym for “Future Interactive Network Environment”, the name of Electric Communities’ earlier cyberspace R&D efforts on behalf of Fujitsu.

Frontend — A client which provides a user interface for access to a backend.

GCI — Acronym for “Global Cyberspace Infrastructure”,

Host — In the distributed instantiation object model, a computational entity responsible for the state of an unum.

Object — A computational entity which realizes a part of some conceptual entity in a software system by sending and receiving messages to and from other objects.

OOP object — A package of data and executable code that encapsulate some concept in a program; in distributed instantiation terminology, a server.

Region — An unum which acts as a communications nexus and common point of reference for a set of una interacting together.

Server — (1) An object which provides a service to a client through some interface. (2) An OOP object in Joule.

Service — A useful function or group of related functions provided to objects and/or users in the Cyberspace world via a server.

Service framework — A set of object classes, protocols and procedural conventions which collectively enable a particular category of service transaction to be possible in the Cyberspace network.

Unum — A world object in the distributed instantiation object model. Derived from the latin meaning “one thing”. Plural form is “una”.

Traversable — A property of a network such that objects may freely move over (i.e., traverse) it from place to place.

World object — A discrete entity that can be manipulated by the user or which has some definite role within the world, typically with some metaphorical or analogical relationship to a real-world object of some kind; in distributed instantiation terminology, an unum.

