HyperPlace: Extending Web Platform to Build Online Places for Multiuser Interaction and Collaboration

Thesis Draft for the degree of

Master of Science in Information and Computer Science, UC Irvine

by

Marvin Park

> *The creation of something new is not accomplished by the intellect but by the play instinct acting from inner necessity. The creative mind plays with the objects it loves.*
>
> \- Carl Jung

ABSTRACT

This paper introduces a new form of networked, participatory medium called *hyperplace*. It is a persistent online place for multiple users to interact, play, and collaborate in real time. Each hyperplace has its own rules, narrative, structure, objects, and characters all created and owned by individual participants. The structure of hyperplace network is similar to that of the WWW in that they are topologically connected to each other, forming a decentralized network. Hypertext and hyperplace networks also share common ideologies, such as freedom of expression, distributed ownership, and democratized access. In order to demonstrate the feasibility and evaluate the performance of such networks, I designed and developed the *HyperPlace* platform—a web-based hyperplace player and a distributed object management system where users can build and utilize hyperplaces. This platform lowered the technical entry barriers of authoring new interactive content, and thereby facilitating creative play. This paper evaluates the prototype and explores more possible contents and applications that can be developed on this hyperplace-authoring platform.

INTRODUCTION

Recently, users are gaining more control over the contents on Web 2.0 sites. Their creative participation, which fuels the ecology of Web 2.0, is sometimes driven by self-satisfaction of themselves, but is mostly motivated by the presence of other people. The users feel more rewarded and motivated when they receive immediate feedbacks for their participations. Therefore, Web 2.0 platforms not only enable but also encourage online collaboration among users by making Web pages more interactive to both user input and server response. Catalyzed by the recent advancement of interactive Web technologies such as AJAX or Flash, the paradigm of Web services is rapidly shifting from the passive "browse" mode to the active "play" mode[1].

However, this collaborative content creation is not new to the realm of virtual worlds. In 1990, Pavel Curtis created "LambdaMOO[2]" MUD (Multiuser Dungeon) with the similar philosophy of player participation even before the development of the Web platform. While it was a text-based virtual world service, players could create and control every detail of artifacts in their own world without graphical limitations. This concept of user-created world was adopted to its 3D descendant, "Second Life[3]" (2003, Linden Lab). Second Life has created a new possibility of virtual worlds as a new media

[1] Yet, Tim Berners-Lee's original vision was of a two-way web that could be written as well as read via the browser. [xxx]

[2] http://www.moo.mud.org/

[3] http://www.secondlife.com/whatis/

platform; however, its technical entry barrier is still too high for content creators. In addition to the fact that it is actually a closed "virtual environment" hosting service rather than an open platform, its content creation process requires sophisticated skills and knowledge to deal with 3-D object modeling and proprietary programming language, as discussed later in chapter 3.

In order to encourage a player's participation in a better way and harness their collective creativity, virtual world platforms need to lower these technical entry barriers of authoring content. I was therefore motivated to design a new player-driven virtual world platform, HyperPlace, on which players can express themselves as easily as they do on the Web. Thus, the HyperPlace was originally designed as a virtual world platform where players can create and manage their own virtual places, objects, and characters that constitute a part of the synthetic world.

Since it is basically a virtual world system, I began modifying the source code of the LambdaMOO server as an initial approach to construct the HyperPlace platform. I found that LambdaMOO was a very versatile object language interpreter platform. However, it was designed as a centralized system, and thus inevitably unscalable. I desired to build a scalable and completely decentralized architecture, the criteria of which were not satisfied by the traditional game (or MUD) engines.

After several revisions, a decentralized system architecture was designed (Figure 1). It actually was a parallel extension of the Web platform, which is a quintessential distributed system. I later realized that the original concept of the Web was an "interactive space" [39] similar to the HyperPlace. The pursuit of the similar vision naturally led to a similarity in the system architecture. The HyperPlace platform is implemented as an interactive media platform that renders generic multiuser space. Although it started as a virtual world platform, its possibility is not confined to a specific application. Chapter 1 defines a new type of media, hyperplace, which runs on the HyperPlace platform.

In many aspects, the HyperPlace is similar to the VRML (Virtual Reality Markup Language) platform. They both render navigational space using web browsers. However, VRML is just a standard file format describing a static 3-D model world; therefore, it cannot build a multiuser environment all by itself. VRML worlds are created in users' local computers and are not shared. It corresponds to the HPML file format (illustrated in Chapter 3-3) of the HyperPlace platform. It lacks service protocols and server components to utilize the representational technology. This is why VRML is not widely used and the VRML Consortium's initial vision of building a spatial extension of the Web is not yet realized [4]. In this regard, the HyperPlace attempted to provide at least a minimum package of tools, including a file format for representation, network protocols, player application, server components, and an IDE (Integrated Development Environment). This relatively comprehensive media platform is built to encourage and facilitate a creative and collaborative play through easy authoring, decentralized control, and distributed ownership.

In this paper, "hyperplace" refers to a type of interactive media, as defined in chapter 1, and "HyperPlace" denotes the implemented prototype platform where hyperplace runs.

[4] http://www.3d-test.com/interviews/mediamachines_1.htm

CHAPTER 1.

CONCEPT OF HYPERPLACE

The original World Wide Web is designed and implemented in order to store, retrieve, interpret, render, and browse hypertexts [xvii] [xxx]. Hypertext is a network of texts interconnected with hyperlinks. It is a logical extension of hypermedia—a nonlinear medium of information composed of text, graphics, video, audio, and hyperlinks[5] [6]. Thus, our existing Web is a hypermedia platform.

Similarly, hyperplace can be defined as an interconnected online place. However, it is not a straightforward extension of hypermedia or hypertext. While the coverage of both media may overlap and a few hybrid contents may exist within the boundaries of both media, they are different from a user's perspective. Hypermedia (or hypertext) is a medium that conveys *information* in third person and hyperplace is a persistent virtual place in which participants can *experience* in first person. This chapter develops the concept of a hyperplace from different points of view in order to highlight its peculiarity as an experiential media.

1-1. Disembedded Place

The term "computer" is no longer termed as an electronic calculator. Its unmatched versatility, which has reshaped our society, industry, and culture, has now associated it with numerous other concepts in our linguistic structure. Likewise, the Internet is no longer referred to as a global communication network that connects computers. It sometimes connotes the entire mode of life after the information revolution. Yet, our language has not evolved to capture the *deconstructive nature* of computers and the Internet, which have taken the places of traditional media by decomposing them through digital technology. For example, they have separated news from physical media such as paper or distribution channel. They also have separated music from CDs or tapes. This separation ultimately changed the way we produce and consume information and content.

I term this separation (performed using IT technology) of digital entities from physical media as "disembedment." The disembedment process begins with the process of associating the physical object or phenomenon with the corresponding digital entities. Through this artificial association, which is sometimes called "design" or "programming," the associated part of the physical world is ready to go online and instantly get connected anywhere. A typical disembedment process is as follows: programming (software designer) -> encoding (computer program) -> transmission (the Internet) -> decoding (computer program) -> representation (end user). Thus, the Internet is actually nothing more than a vehicle that transports encoded data packets at the speed of electrons. In fact, the real magic of disembedment takes place at both ends of the processes performed by human brains for thought. Nothing is separated here in practice. Only patterns are discovered and replicated. Therefore, the concept of separation in the process of disembedment is used metaphorically and differs from the Cartesian dualism, which separates inseparable unity of mind and body (or software and hardware).

By considering this metaphorical concept of disembedment, hyperplace can be simply defined as "*a place disembedded from the physical place*." This does not signify that a hyperplace should be a replica of physical place. As news can exist without newspaper, it emphasizes structural similarity with tangible place and physical absence of it.

Then, what is the abstract "place" separated from the physical world[7]? A human geographer Yi-Fu

[5] By convention, the term "hypertext" is often used where the term "hypermedia" might seem appropriate.

[6] http://en.wikipedia.org/wiki/Hypermedia

[7] Dictionary definition of place, "a location in space" is not separated from the physical world.

Tuan defined place as "the center of felt value where biological needs, such as those for food, water, rest, and protection, are satisfied." [i] The concept of place is often derived from more axiomatic and abstract concept, "space." Places are located in spaces, but not all spaces are places. Space is often associated with openness, freedom, anonymity, and threat, while a place is more about security, stability, character, nuance, history, and identity. [i] [iv] What begins as undifferentiated space becomes place as we get to know it better and endow it with value. [i] Places are imbued with social meanings, but the concept of space has nothing to do with experience. [ii]

Most qualities (or felt value) that identifies a place come from embodied *experience* of living in a physical place. *Experience* is a keyword to understand the various modes through which a person knows and constructs a reality. [i] Hyperplace provides a disembedded experience of being in a place with participants, and the sense of "being" somewhere is reinforced by the illusion of moving through space. [iii]

1-2. Mediated Collaborative Place

When people want to do something together, either work or play, they usually flock together in a place. Thus, collaboration[8], collaborative work or play, literally takes "place." The place for collaboration does not need to be a location of physical space. For example, Google docs[9], a web-based office application, is a good example of a mediated collaborative (work)place where every online co-authors can see who is online and who is editing an open document in real-time. In the case of online collaboration, the simultaneous presence of more than one person creates tension and a sense of belonging, which eventually forms the "sense of place." In terms of user experience, a non-delayed response is critical to acknowledge the presence of other people. For this reason, a telephone conversation is known to create a sense of place that comes from an embodied experience of face-to-face communication [iv]. This also explains why occasional exchange of emails or wiki-based collaboration seldom creates such a sense of place while we feel the sense of place with every chat session in a chat room or an instant messenger window.

The presence of other people is a decisive factor that regulates our behavior in a collaborative place, as Steve Harrison and Paul Dourish put it, “the sense of other people's presence and signs of their activity allow us to structure our own activity, seamlessly integrating communication and collaboration ongoingly and unproblematically. Similarly, spatially-organized collaborative environments present views of other people and their actions within the same environment which represents activity and holds the artifacts of work. [ii]”

Since it is a real-time interactive medium, hyperplace can be defined as a mediated collaborative place, where timely interaction forms the "sense of place" and simultaneous presence of people defines the mode of activities. This concept of collaborative place is akin to Giddens' "*locale*", as the place where interaction occurs [v]. According to Giddens, locale is also applicable to multiple scales of interaction. At a much larger scale, locale may refer to groups of people and how they interact with one another across the landscape. Combined both with the *co-presence* of actors and with the communication established between them, the properties of a *locale*, give a "contextuality" to the interactions that occur in it.

1-3. Building Block of Virtual Worlds

Virtual words, in general, are not games even though some of them, such as Ultima Online (Origin Systems, 1997), EverQuest (Sony Online Entertainment, 1999), and World of Warcraft (Blizzard

[8] The term "collaboration" usually refers to a team work or cooperative process for business project, but, in this context, it is also used as collaborative play as well.

[9] http://docs.google.com/

Entertainment, 2004), are popular massive multi-player online games. Virtual worlds are a medium through which many services (games included) might be delivered. [vi] As a first-person experiential medium, virtual worlds are closely related to the notion of hyperplace. For example, Second Life users are called "residents", which underscores how a "sense of place" is fundamental to Second Life. [vi] Richard A. Bartle, a co-writer of the first MUD, defined "virtual worlds" simply as "places" like following:

“Virtual worlds may simulate abstractions of reality; they may be operated as a service; creating them may be an art; people may visit them to play games. Ultimately, though, they are just a set of locations. Places. People go to places, do things there, and they go home.” [vii]

From the perspective of virtual world designers, hyperplace can be used as a representational unit for a virtual world platform just like what hypertext is for the Web platform. In this sense, a hyperplace is defined as a building block of virtual worlds. However, simply connecting hyperplaces does not automatically constitute a virtual world, which requires a minimum global infrastructure, such as currency and economic/social systems, for example. Hyperplace is designed as a network of individually owned, programmable online places, and therefore, it does not include any default global settings for a specific purpose. Each hyperplace does not necessarily share common rules or policies with neighboring hyperplaces. Hence, a virtual world programmer needs to program his/her own global settings and objects to build a coherent network of hyperplaces. (See chapter 3-5)

CHAPTER 2.

IDEOLOGIES OF THE INTERNET, WEB 2.0, AND HYPERPLACE

According to Brian Winston's model of the "diffusion of technology," a technical prototype is accepted as an invention by the operation of a transforming agency called supervening social necessities [viii]. I assume that this transformation is a resonant process between the ideology (or a comprehensive vision) behind the invention and the unfulfilled desires (or needs) of society. This does not necessarily mean that the ideology of an invention is an accurate reflection of the inventor's vision, which propelled ideation of his/her scientific competence. In some cases, these ideologies are neither clear nor intended at the moment of invention, but become manifest by the possibilities they create after being deployed in a society. My assumption also implies that a technological invention would not solely fulfill social necessities but would also amplify desire for those necessities by encouraging specific modes of activity. For example, cell phones might have increased the desire for remote communication. This amplification of desire is the reason why I called the transformation a resonant process.

As a newly introduced networked media platform, the historical growth of the Web is a perfect role model for the HyperPlace platform. Based on the assumptions above, I will explore how the early Internet sowed the seeds of open communication in the late 20th century, and I will juxtapose this ideology with the paradox of the recent Web 2.0 movement, which has inherited the traits of its predecessor but is actually planting seeds of centralized ownership in the early 21st century. The ideologies of the HyperPlace will become manifest by this juxtaposition. Metaphorically, the interaction between the new ideology of a technological invention and those ideologies already in existence in a society is analogous to a process of cultivation; it takes time for a new ideology (seed) to take root in society (soil), and its development depends on the complex chemical interactions it has with the soil in which it is growing.

2-1. Ideologies of the Internet before Web 2.0

2-1-1. ARPANET: The seeds of uncensored communication

To explore the evolution of computer networks, we first need to clarify the term "Internet." The Internet can be divided into the commonly used "the Internet" and numerous "internets" since several network protocols are used to communicate between network terminals. The Internet is made possible by using RFC 768 (UDP)[10], 793 (TCP)[11] and 1180 (TCP/IP)[12] as global standard protocols to build network applications. This IP network inherited its decentralized structure from its revolutionary predecessor ARPANET (Advanced Research Projects Agency Network), which was developed by ARPA of the United States Department of Defense.

One of the most common semi-mythical notions about ARPANET is that one of its most important design goals was to be resistant to nuclear (or any missile) attack by using a fault-tolerant routing structure [ix]. The most distinguishing feature of this protocol was the deployment of packet-switching instead of traditional circuit-switching[13]. With packet switching, a system could use one communication link to communicate with more than one machine by disassembling data into

[10] http://www.faqs.org/rfcs/rfc768.html

[11] http://www.faqs.org/rfcs/rfc793.html

[12] http://www.faqs.org/rfcs/rfc1180.html

[13] http://en.wikipedia.org/wiki/Circuit_switching

datagraphs, then gathering these datagraphs as packets. Not only could the link be shared, but each packet could be routed independently of other packets [x].

This decentralized packet-switching was the very basis of the "end-to-end" (E2E) principle, which defined the role of a network thereafter [xi]. This E2E principle is still bolstering the open structure of the Internet and bearing fruit in the form of peer-to-peer (P2P) information/data exchange networks. Simply put, information processing units (network application programs) exist only at the ends of the network and the network should provide only transmission service as a blackbox.

What is the ideology behind the E2E principle? It could be interpreted as the birth of a completely uncensored communication network which ignores the context of communication: Who is sending, who is receiving, what is being transmitted and for what reason. It is unclear whether this was intended from the beginning. This may have been an unintended consequence of a technical limitation; in this protocol, only the sending/receiving terminals can decode a series of wrapped packets regardless their traveling routes.

2-1-2. Web 1.0: The seeds of democratic access and self-publishing of information

There is no common terminology to refer to the World Wide Web (WWW) prior to Web 2.0; Web 2.0 was a term coined to refer to a business trend rather than an official technical update of the Web platform. While the borderline is blurred (or nonexistent), I will refer to the WWW proposed by Tim Berners-Lee in March 1989 as Web 1.0 when discussing its original design philosophy.

The original purpose of Web 1.0 was to improve the information management system of the CERN research center [xii]. All the information systems at that time, including Usenet news groups, local file systems and even help systems, used tree-like hierarchical structures. CERN's complex research project had showed that a hierarchical information architecture could not model the real world.

Our perception of the real world, as modeled in the human brain, consists of neurons (nodes) that are connected to other neurons, thus forming a decentralized neural network. This concept of a neuron-like network of information had never been implemented before Berners-Lee proposed his revolutionary hypertext information architecture. All the information systems in place at the time adopted a tree-like hierarchical structure for Usenet news groups, local file systems, and even help systems. This structure might have been more acceptable to a society where access to a higher level of information was restricted to the upper echelons. Although he had only intended to solve the problem of ambiguous categorization in large-scale information systems [xii], it actually served a greater purpose. It democratized access to information by providing a direct path to any information node by technically ignoring the context of access: Wherever you are and wherever you want to go.

Another contribution of Web 1.0 was that it *exposed* the information from local data storage to a global network. Hypertext not only democratized information, but also provided a unified interface to represent and publish information regardless its internal format or location. Tim Berners-Lee wrote in his proposal that, "the method of storage must not place its own restraints on the information. [xii]" He also wrote in another book that, "the fundamental principle behind the Web was that once someone somewhere made available a document, database, graphic, sound, video, or screen at some stage in an interactive dialogue, it should be accessible (subject to authorization, of course) by anyone, with any type of computer, in any country. [xiii]" This ideology, summarized in a single sentence, became a cornerstone of today's global heterogeneous information network.

2-2. Ideologies of Web 2.0

Web 2.0 is a term which describes a trend in the use of World Wide Web technology and web design that aims to enhance creativity, information sharing, and, most notably, collaboration among users [xiv]. These concepts have led to the development and evolution of web-based communities and hosted services, such as social-networking sites, wikis, blogs, and folksonomies[14]. The term became noable and widespread shortly after the first O"Reilly Media Web 2.0 conference in 2004[15].

2-2-1. Web 2.0: The seeds of user-created media culture

In a podcast interview for IBM, Tim Berners-Lee dismissed the term Web 2.0, saying, “Web 2.0 is of course a piece of jargon, nobody even knows what it means.” He goes on to note that “it means using the standards which have been produced by all these people working on Web 1.0. [xv]”

There is evidence that supports his claims. So-called Web 2.0 sites are characterizing themselves with a list of concepts such as the network effect, social networking, personal media or user created content, but there is hardly anything truly new in Web 2.0. The network effect as business phenomenon existed even before Web 1.0[16]. All the other ideas were first implemented around 1995 with the beginning of the first dot-com bubble. The first social network service Match.com started in 1995[17]; the Web has always been social without social networking services. The first blog was pioneered in 1994 by Justin Hall[18] and the first Wiki was launched in 1995[19]. Book review pages of Amazon.com have been filled with user-created feedbacks since 1995.

Thus, it is safe to say that Web 2.0 is just part of an ongoing development process and that it cannot be divided cleanly from its predecessor. But, we can see a clear trend for so-called Web 2.0 technologies. Most of the important developments around it have been aimed at enabling a community to create, modify, and share content in a way that was previously only available to centralized organizations which bought expensive software packages, paid staff to handle the technical aspects of the site and create content which was published only on that organization’s site.

These new content creation technologies fundamentally changed the mode of production for the Web content [xvi]. Web applications and services have become cheaper and easier to implement, and by allowing the end users access to these applications, a company can effectively outsource the creation and the organization of their content to the end users themselves. Instead of the traditional model of a content provider publishing their own content and the end user consuming it, the new model allows the company's site to act as a centralized portal for users who are both creators and consumers.

For the user, access to these applications empowers them to create and publish content that previously would have required them to purchase desktop software and possess a greater technological skill set. For example, two of the primary means of text-based content production in Web 2.0 are blogs and wikis, which allow the user to create and publish content directly from their browser without any real knowledge of markup language, file transfer or syndication protocols, and all without the need to purchase any software. The use of the web application to replace desktop software is even more significant for the user when it comes to content that is not merely textual. Not only can web pages be created and edited in the browser without purchasing HTML editing software, photographs can be uploaded and manipulated online through the browser without the need for expensive desktop image manipulation applications. A video shot on a standard consumer camcorder can be submitted to a video hosting site, uploaded, encoded, embedded into an HTML page, published, tagged, and

[14] also known as collaborative tagging, social classification, social indexing, and social tagging
[15] http://conferences.oreillynet.com/web2con/
[16] The term "network externalty" was first presented in a paper by Bell employee N. Lytkins in 1917
[17] http://www.match.com/matchus/help/aboutus.aspx
[18] http://en.wikipedia.org/wiki/Justin_Hall
[19] http://c2.com/cgi/wiki?WelcomeVisitors

syndicated across the web, all through the user's browser.

At the end of 2006, Time magazine chose "YOU," the online collaborator, as its "Person of the Year."[20] This epitomized the user-driven paradigm shift of media culture in 21st century, as harnessed by Web 2.0 technology as the content creation platform. But what, ultimately, will this paradigm shift do? What ideology is behind this movement? To answer these questions, we need to look at the value of user-created content from the perspective of Web 2.0 companies.

2-2-2. Web 2.0: The seeds of free labor and centralized ownership

From a business standpoint, the borderline demarcating Web 1.0 and Web 2.0 may be the period between the first dot-com bubble (roughly 1995–2001) and Google's IPO (2004), which coincided with the first Web 2.0 conference (2004). While a lot of the first generation Web 1.0 companies failed due to a lack of a real business model, Google's successful search business model reassured and motivated venture capitalists who needed a new reason to invest in Internet startups. Thus, although it is still being debated, the dominant opinion about Web 2.0 is that it is actually Internet Investment Bubble 2.0. The feeling is that it is based on a fantasy, as if new Internet startups are somehow grounded on a completely updated technical platform [[xvii]].

Unlike Web 1.0, Web 2.0 investors do not need to finance software development or content creation. There are plenty of robust, enterprise-level open source software packages freely available and the content is created by unpaid users. Basically, by providing some bandwidth and disk space, any group of people that can market a site effectively can become a successful Web 2.0 company. The principal success of the business model comes from the ability of the companies to "harness collective intelligence[xviii] [[xix]]," which means to be monolithic in their branding and ownership of that content, while opening up the method of content creation to the community.

While it uses quite robust, scalable storage and servers, the real value of YouTube was not created by the developers of the site, but rather by the people who uploaded videos to the site. However, when YouTube was bought for over 1.6 billion dollars worth of Google stock in 2006, absolutely none of this stock was acquired by the video creators. The value produced by users of Web 2.0 services, such as YouTube, is captured by the investors. From this perspective, every Web 2.0 site's attempt to harness collective intelligence may turn out to be a private approbation of community-created content. This is not fundamentally different from the neoliberal "privatization and commodification of public assets" or "accumulation by dispossession" presented by David Harvey [[xx]]. Seemingly voluntary participation in user collaboration has been exploited; Gabriel Tarde redefined it as "col-labor-ation. [[xxi]]"

Capitalism, rooted in the idea of Marx's primitive capital accumulation theory [[xxii]] and violent deprivation of (salaried) labor, requires centralized control, without which peer producers have no reason to share their income with outside shareholders. Capitalism, therefore, is incompatible with the nature of uncontrollable and decentralized networks. From this perspective, the Web 2.0 business trend is the return of monolithic online services[21] with a mission of destroying the decentralized nature of the Internet. Information production under Web 2.0 creates a landless information proletariat ready to provide alienated free labor [xix] for the new info-landlords of Web 2.0 companies. Thus, Web 2.0 is not to be thought of as a second-generation of either the technical or social development of the Internet, but rather as a second wave of the capitalist enclosure of the "information commons. [[xxiii]]"

[20] http://www.time.com/time/magazine/article/0,9171,1569514,00.html

[21] AOL (America Online) is a quintessential example of the monolithic, centralized, and bundled online service.

2-3. Ideology of the HyperPlace

Commons-based production, such as the Wikipedia project, is a new modality of organizing production: radically decentralized, collaborative, and non-proprietary; based on sharing resources and outputs among widely distributed, loosely connected individuals who cooperate with each other without relying on either market signals or managerial commands [xxiv]. Usenet, email, and early Web 1.0 sites were also rooted in cooperative, decentralized and commons-based systems, owned by everybody and nobody. Meanwhile, privately owned, lucrative Web 2.0 sites are usually equipped with exclusive databases, better visual representations and richer content authoring tools in order to attract more unpaid content creators and audience. If the development of commons-based production platforms requires wealth from venture capital, the great potential of the Internet as a commons may remain unrealized. Yet, many open source projects cited as the key innovations in the development of Web 2.0, such as Linux, Apache, PHP, MySQL, and Python, will become free backbones in building a genuine, decentralized commons.

The HyperPlace platform aims to be anther open-source tool for commons-based production as well as being a decentralized commons where users can build places to flock together and create objects to play with. Commons, in this context, is a particular institutional form of structuring the right to access, use, and control resources, as opposed to private property [xxiv]. Thus, it does not necessarily involve the production and consumption of "information" as web sites do. Rather, the HyperPlace platform is an "experience commons," where users (or players) share the unique moments of interaction. In this regard, the abstract vision of the HyperPlace is almost identical to that of the original Web 1.0. As Tim Berners-Lee puts it: “The idea of the Web 1.0 is all about connecting people. It was designed to be as a collaborative space where people can interact.” [xv] As an open and community-regulated commons [xxiv], it inherits the philosophical underpinnings of the Internet and Web 1.0, including uncensored self-expression, democratized access, and decentralized ownership. Its ideologies do not differ from those of Web 2.0 except for one aspect: It allows the community to own what it creates.

CHAPTER 3

DESIGN AND DEVELOPMENT OF THE HYPERPLACE PLATFORM

The concept and ideologies of the hyperplace can be specified in terms of a web interface and networked interactions. As a concept-proof prototype, the HyperPlace platform was implemented according to this technical specification of the system. This chapter illustrates the requirements, design strategies and technical details of the prototype project.

3-1. Requirements Analysis

A software ***platform*** refers to a software/application framework that allows other software to run[22]. According to this definition, a hyperplace network can be defined as a platform that provides a software framework allowing the execution of user-programmable objects. The ultimate design goal of the HyperPlace, as a hypermedia platform, is to support and encourage its users to express themselves with a maximum degree of freedom and creativity. In this respect, perspective content creators (object programmers) are considered to be the primary target user group for this platform. The following platform-wide design requirements are oriented to meet the needs of this target group at every stage of content authoring projects.

3-1-1. Accessibility

Content creators are usually motivated by anticipated positive feedback about their work. From their point of view, a good platform is one which target audience can readily access. There is no generally accepted or absolute measures of software accessibility, but subjective experiential accessibility is associated with quantitative variables such as required time and cost to get the software to a readily usable state.

In this respect, the most accessible platform can be defined as a platform that meets the following criteria;

- It runs on average configuration of PC hardware and network bandwidth.
- It must not be bound to specific hardware, OS or commercial application software.
- It should be released as free software.
- Networked contents running on the platform should be instantly delivered to the audience.

3-1-2. Ease of Learning

Authoring of interactive content often requires programming skills and knowledge of a proprietary programming language, including JavaScript for DHML, ActionScript for Adobe Flash[23], Lingo for Macromedia Director[24] and HyperTalk for HyperCard [xxv]. Some virtual world platforms also allow the programmability of internal objects with players. LambdaMOO is actually a multi-user interpreter environment that reads and executes objects written in MOO programming language, which was

[22] http://en.wikipedia.org/wiki/Platform_(computing)

[23] http://www.adobe.com/devnet/actionscript

[24] http://www.adobe.com/devnet/director/

developed by Stephen F. White and Pavel Curtis (1990)[25]. Second Life players can build and control their own objects and avatars with Linden Script Language (LSL)[26].

For some content creators, authoring interactive content is a great challenge since it requires programming experience and skills. Even skilled programmers are often discouraged from learning a new programming language when it is only used for specific purpose. There are two possible ways to lower this technical barrier to building new interactive content and thereby, to shorten the learning curve.

The first and most obvious way is to replace textual programming language with a more intuitive visual programming environment or natural language. In this regard, the "Etoy" visual programming environment[27] has been pre-installed on OLPC XO-1 children's laptops[28]. The resemblance to HyperTalk phrases and English sentences (for example, "*put the value of card field one of this stack*") may alleviate the stress of learning unfamiliar syntax [xxv].

Another approach is to use a general-purpose programming language with a relatively large user base. In addition to familiarity, programmers can take advantage of existing code and abundant referential resources that are freely available on the Web. This reusing and sharing of components (or modules) would not only lower the entry barrier for novice programmers but would also increase the overall productivity of content authoring.

3-1-3. Decentralization

However easy it may be, users will still be reluctant to express their original idea when their production is filtered by censorship and restricted by authoritative guidelines. As described in chapter 2, a fundamental ideology of hyperplace is to decentralize ownership of community-created value and protect the natural ecology of production and distribution of contents from commercial exploitation. To enable this distributed ownership, each piece of content needs to be physically (or digitally) owned by its creator and protected by an ownership management system [[xxvi]].

This requirement can be naturally fulfilled by physically decentralizing the content repository. This diffusion of storage implies that a hyperplace platform needs to run on a loosely connected heterogeneous network, which requires predefined shared protocols to allow them to operate together.

3-1-4. Real-time Interactivity

In order to function as a collaborative place as discussed in chapter 1-2, the HyperPlace platform should be implemented to allow real-time communication. As will be discussed in chapter 3-2, this technically implies that the connection between a hyperplace client and server should be persistent.

3-2. System Architecture

Considering the resemblance with the Internet (or Web) of the major requirements described above, it is natural to build a hyperplace platform inheriting legacy web platform. This implies that the HyperPlace will be implemented under a traditional client-server paradigm and legacy web applications such as web browsers and web servers will be used. The Web is one of the most accessible software platforms in that it is hardware independent, OS independent and basically free. While a few gigantic hub nodes are seemingly reshaping the entire connectivity of the network, the Web is essentially a decentralized hypertext network. Thus, the philosophy and architecture of the Web satisfies the requirements of accessibility and decentralization. However, this approach limits the type of hyperplace contents to web-friendly formats such as DHTML, Java, Flash or Shockwave. Another limitation of the native web platform is the lack of real-time interactivity, since it is based on

[25] http://www.ccs.neu.edu/home/ivan/moo/lm_toc.html
[26] http://wiki.secondlife.com/wiki/LSL_Portal_Guidelines
[27] http://wiki.laptop.org/go/Etoys
[28] http://laptop.org/laptop

connectionless HTTP protocol. A hyperplace platform ought to be built on a bidirectional real-time network in order to better represent and simulate multi-user interactive space.

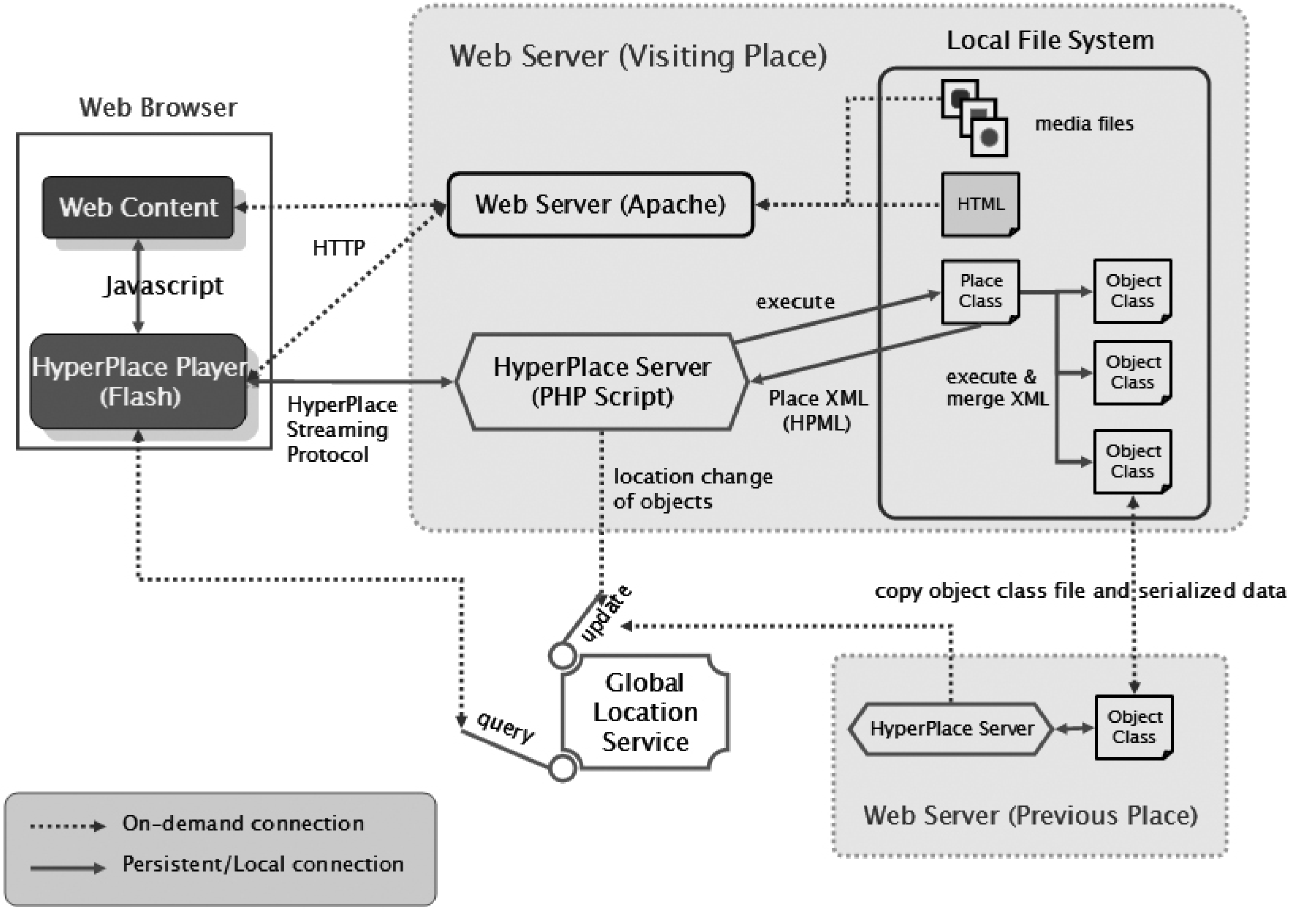


Figure 1: System Architecture of the HyperPlace Platform

AJAX (Asynchronous Javascript and XML) was first considered as a technique to emulate real-time communication on the web. Although AJAX had been one of the most frequently used web development techniques to implement data-driven dynamic web pages, its unidirectional socket connection[29] is inappropriate to build real-time environment. Since AJAX is a remote procedure (method) call implemented over HTTP, clients (web browsers) must initiate every communication cycle. To emulate server-initiated messaging, clients must periodically pull new data from a server. There is an inevitable delay between data pulls, and this method becomes highly inefficient when large numbers of clients are connected to a server. Suppose that a server randomly broadcasts message randomly and the client pulls the message from the server every 0.5 second. If 1,000 clients are connected, the server should respond to 120,000 (= 2 * 1,000 * 60) queries per a minute, regardless of the existence of available messages. If the server can directly broadcast a message to clients, it only then needs to send it to 1,000 clients when a new message is ready.

To avoid the unnecessary system overload, a genuine (not emulated) full-duplex communication channel should be established and maintained between web browsers and web servers. Figure 1 illustrates the overall system architecture of the HyperPlace platform that implemented this persistent socket connection over web architecture.

[29] Only clients can open a TCP socket to connect to servers, which means servers can not reach disconnected clients.

3-2-1. Hyperplace Player

On the web browser side, a persistent socket connection was implemented using Adobe Flash[30] web plug-in. Flash, a de facto standard plug-in software for multimedia web applications, was chosen because it can be embedded in all major web browsers[31] without sacrificing accessibility[32]. Besides maintaining network connection with a server, hyperplace players also interface user input and render hyperplace content. In other words, every hyperplace-hosting web page should include this component in its source code and page layout. It retrieves and parses a hyperplace document written in XML format (described in section 3-3), and requests additional resource files to render an interactive place. This process resembles the way in which HTML is retrieved and rendered by a web browser.

The HyperPlace player web control provides Javascript interface methods to a hosting web page such as *openPlace*, *login* and *logout* so that it can be controlled within a standard web programming environment. Javascript functions in a hosting web page can also be called by the HyperPlace player through Flash's ActionScript API[33]. Thus, two-way communication between web content and hyperplace content is possible at the local function-call level.

3-2-2. Hyperplace Server

On the web server side, hyperplace servers listen to a socket port[34] in order to accept incoming connections from hyperplace players. Once a persistent socket connection is established, it continuously sends run-time simulation data of hyperplaces to the client with Hyperplace Streaming Protocol (HPSP), which will be described in section 3-4. The HyperPlace server is entirely written in PHP script language[35] (running in standalone mode) and is tested to be compatible with Microsoft Windows XP, Mac OS X and Linux without code modification. Since PHP is a widely used web script language, web server administrators do not need to install and configure a separate server software package to host hyperplaces on their web servers. Only a PHP-enabled web server[36] is necessary to host hyperplaces.

A hyperplace server running on a single machine can simultaneously simulate and transfer multiple hyperplaces. This process is similar to how multiple web pages are handled by a web server and web browsers.

3-2-3. Global Location Service

Hyperplace players and servers can access a specific object in the entire hyperplace network with a unique ID. However, unlike hypertext documents, objects in a hyperplace frequently move across the hyperplace network, as will be explained in section 3-5. Given the delay between indexing, the indexing techniques employed by web search engines are inappropriate as a tool for tracking the real-time location of dynamic objects.

Global Location Service (GLS) is a globally-accessible yellow page that maintains an association array to match objects' names with their current physical location. When an object moves to another hyperplace, its hosting server notifies this movement to the GLS, which in turn updates the object's location.

GLS is the only centralized server component in the entire hyperplace network, but it is a background helper service, like the Internet's domain name service, and most users and content creators do not need to be aware of its existence.

[30] Flash Control 9.0 format, compiled by Adobe Flex Builder 3.0

[31] Flash format is compatible with Microsoft Internet Explorer, Mozilla Firefox, Apple Safari and Opera web browser.

[32] http://www.adobe.com/products/flashplayer/productinfo/systemreqs/

[33] ExternalInterface.call function was used for calling Javascript functions from ActionScript

[34] Any port other than web ports (80, 8080) and conventional TCP application ports such as 21 (FTP) or 25 (SMTP).

[35] PHP version 5.2.5 was used to develop and test the HyperPlace server components.

[36] For example, Apache web server was used for the prototype.

3-3. Hyperplace Markup Language

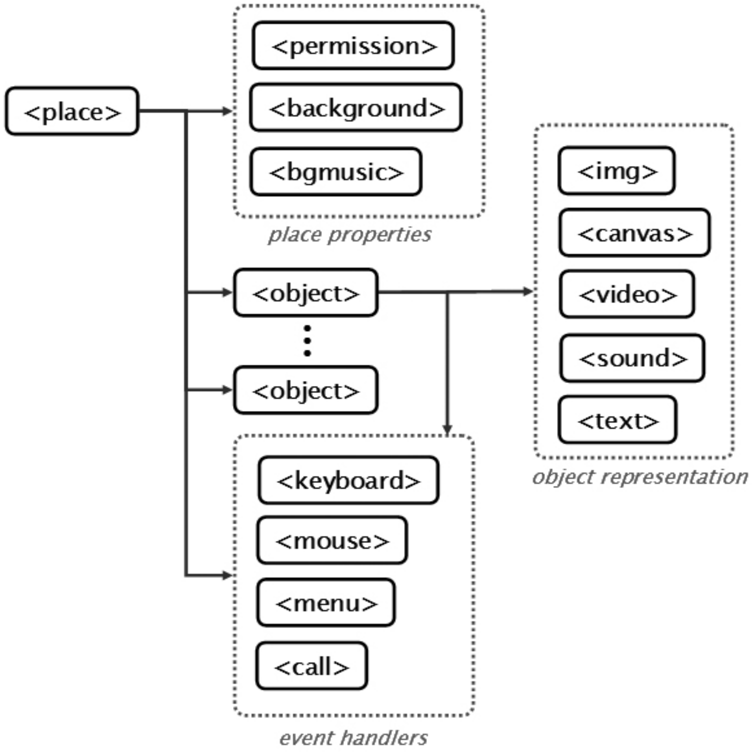


Figure 2: Structure of a Place Document

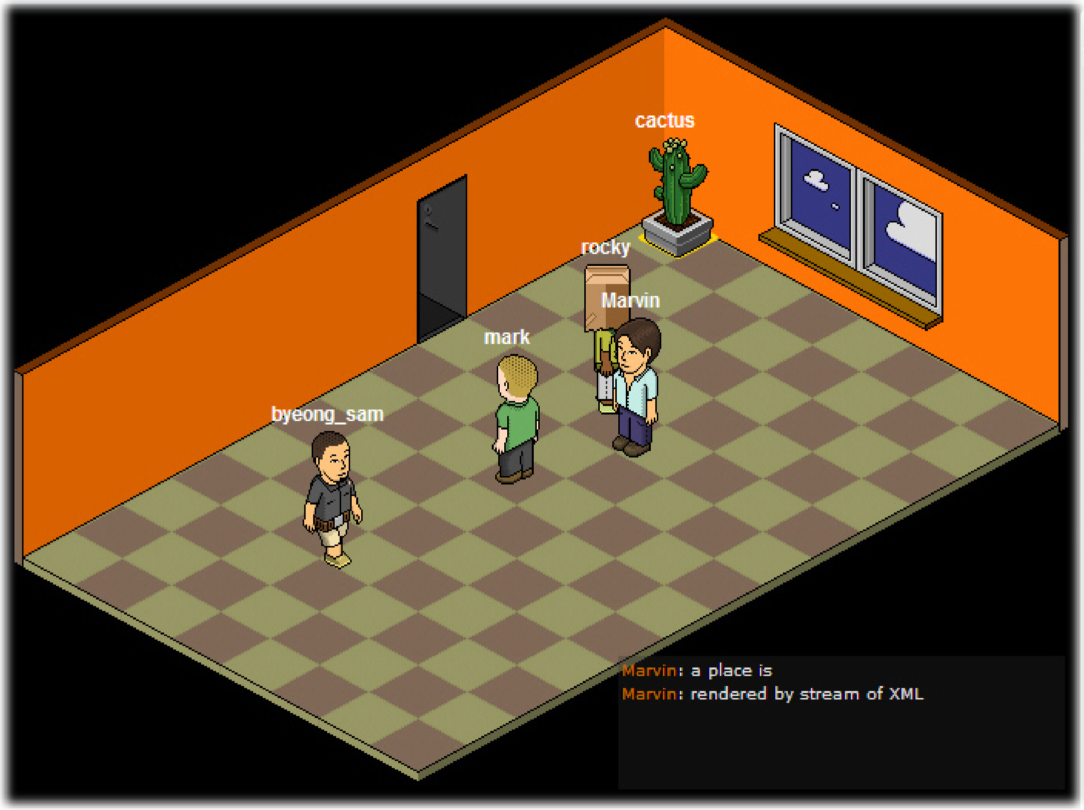
Hyperplace Markup Language (HPML) is an XML document that represents a hyperplace. As depicted in Figure 2, a hyperplace consists of objects, events and properties, such as permission to access. Appendix A provides, for reference, XML markup syntax of HPML documents. Objects may have event handlers that define how they interact with users or other objects. A place object is a root container object that contains all the current objects in the place. An object contains layers of user interface elements including bitmap images, vector graphics[37], formatted text, video and sound. These media resources can be stored in any accessible web server and are linked through HTTP protocol.

The main function of hyperplace players is to retrieve, interpret and render this HPML in order to simulate hyperplaces. Unlike HTML (Hypertext Markup Language), the HPML of a given hyperplace changes in real time as the status of objects change. Thus, a HPML document needs to be continuously streamed to hyperplace players, as opposed to being transferred upon requests. A network protocol to stream HPML is described in section 3-4.

The visual representation of hyperplaces is another key role played by hyperplace players. Objects and display elements may be represented in either a two-dimensional or a three-dimensional coordination system depending on the display scheme of the container place. The Hyperplace implemented a 2.5 dimensional isometric coordination system[38] instead of three dimensional space as is shown in Figure 3. Figure 4 illustrates the source code for the hyperplace in Figure 3.

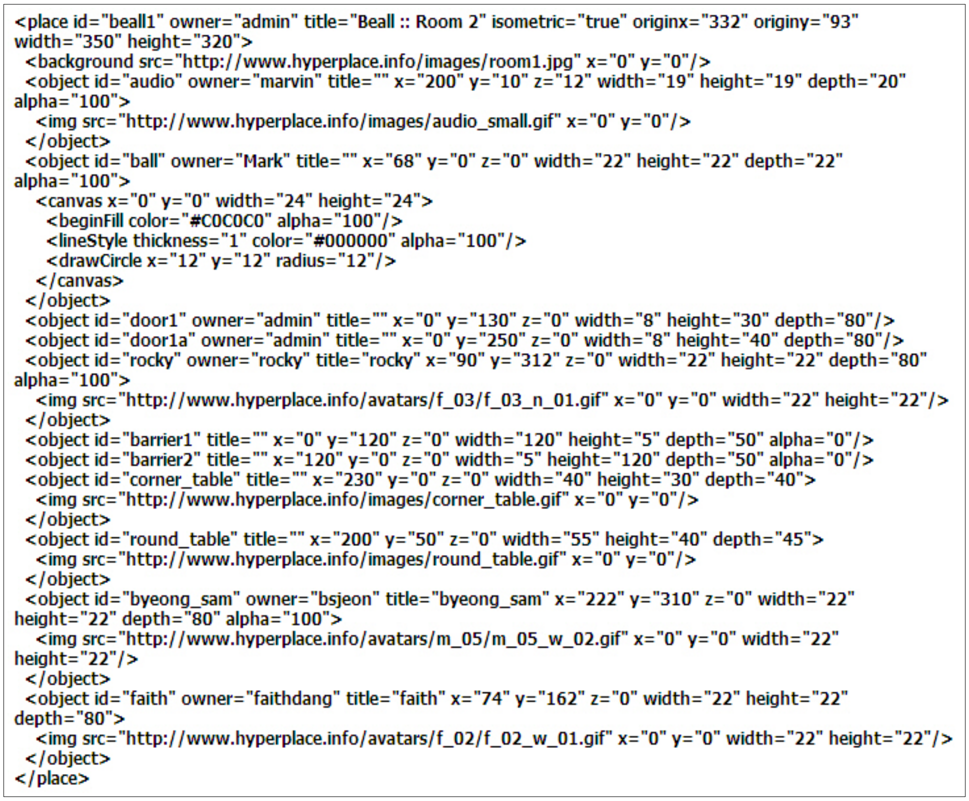
[37] Vector graphic is implemented using drawing APIs similar to ActionScript graphics APIs.

[38] A method of visually representing three-dimensional objects in two dimensions, in which the three coordinate axes appear equally foreshortened and the angles between any two of them are 120°



Marvin: a place is
Marvin: rendered by stream of XML

Figure 3: Isometric representation of a 3D hyperplace



```
<place id="beall1" owner="admin" title="Beall :: Room 2" isometric="true" originx="332" originy="93"
width="350" height="320">
 <background src="http://www.hyperplace.info/images/room1.jpg" x="0" y="0"/>
 <object id="audio" owner="marvin" title="" x="200" y="10" z="12" width="19" height="19" depth="20"
alpha="100">
  <img src="http://www.hyperplace.info/images/audio_small.gif" x="0" y="0"/>
 </object>
 <object id="ball" owner="Mark" title="" x="68" y="0" z="0" width="22" height="22" depth="22"
alpha="100">
  <canvas x="0" y="0" width="24" height="24">
   <beginFill color="#C0C0C0" alpha="100"/>
   <lineStyle thickness="1" color="#000000" alpha="100"/>
   <drawCircle x="12" y="12" radius="12"/>
  </canvas>
 </object>
 <object id="door1" owner="admin" title="" x="0" y="130" z="0" width="8" height="30" depth="80"/>
 <object id="door1a" owner="admin" title="" x="0" y="250" z="0" width="8" height="40" depth="80"/>
 <object id="rocky" owner="rocky" title="rocky" x="90" y="312" z="0" width="22" height="22" depth="80"
alpha="100">
  <img src="http://www.hyperplace.info/avatars/f_03/f_03_n_01.gif" x="0" y="0" width="22" height="22"/>
 </object>
 <object id="barrier1" title="" x="0" y="120" z="0" width="120" height="5" depth="50" alpha="0"/>
 <object id="barrier2" title="" x="120" y="0" z="0" width="5" height="120" depth="50" alpha="0"/>
 <object id="corner_table" title="" x="230" y="0" z="0" width="40" height="30" depth="40">
  <img src="http://www.hyperplace.info/images/corner_table.gif" x="0" y="0"/>
 </object>
 <object id="round_table" title="" x="200" y="50" z="0" width="55" height="40" depth="45">
  <img src="http://www.hyperplace.info/images/round_table.gif" x="0" y="0"/>
 </object>
 <object id="byeong_sam" owner="bsjeon" title="byeong_sam" x="222" y="310" z="0" width="22"
height="22" depth="80" alpha="100">
  <img src="http://www.hyperplace.info/avatars/m_05/m_05_w_02.gif" x="0" y="0" width="22"
height="22"/>
 </object>
 <object id="faith" owner="faithdang" title="faith" x="74" y="162" z="0" width="22" height="22"
depth="80">
  <img src="http://www.hyperplace.info/avatars/f_02/f_02_w_01.gif" x="0" y="0" width="22" height="22"/>
 </object>
</place>
```

Figure 4: Sample HPML source code

3-4. Hyperplace Streaming Protocol

When a user types the URL of a hyperplace or when the avatar object moves to a specific (hyper)place, a hyperplace player connects to the host of that place and awaits an incoming stream of HPML. This process involves the connection, authentication, log-on and streaming of data between a client and a server, all of which is handled by Hyperplace Streaming Protocol (HPSP).

HPSP is a human-readable textual protocol in a simplified format of XML-RPC[39]. Table 1 illustrates HPSP command sets for access control, object control, event control and network control. Detailed descriptions and examples of each command are given in Appendix B.

Category	Commands
Access Control	policy-file-request, login, logout, enter, bye
Object Control	create, delete, view, checkin, checkout
Event Control	event
Network Control	ready, test

<Table 1: HPSP command sets>

3-4-1. Minimizing streaming data traffic

The seamless representation of a hyperplace requires a stream of HPML with a ratio of 20+ frames per a second. The simplest way to implement this streaming is to transfer the entire HPML document whenever an update is ready. However, this brute-force approach is highly inefficient for network usage. Suppose that a HPML document at a given time is 3 Kbytes and an updated HPML document is transferred to a client every 0.05 second. At least 480 Kbits/s (20 x 3 x 8) of network bandwidth would then be consumed by a single client-server connection. ISDN-level (128 Kbits/s) clients can not handle this stream and the server network would become a bottleneck for the entire system while serving multiple connections.

One possible way to reduce the amount of traffic is to compress HPML data just as HTTP 1.1 supports compressed content encoding[40]. When tested, HPML files were, on average, compressed to 20% of their original size by gzip[41] encoding. While this compression/decompression algorithm requires 5-10% additional CPU time[42] for both clients and servers, this time cost is negligible compared to the savings on network bandwidth and download times.

Another efficient heuristic to minimize data traffic is to transfer only the changed segments of the HPML from two consecutive frames. This idea is based on the fact that, in most cases, only a small portion of the document changes at a time. No more than 10% of the original data will typically be transferred using this method. The comparison of two XML documents[43] consumes far less CPU time than the gzip compression algorithm. A core comparison algorithm was implemented in the HyperPlace server and it also keeps track of the last HPML document transferred to each client in order to filter out common lines and distill changed ones.

3-4-2. Handshaking between clients and servers

Even after minimizing the amount of streaming traffic, network congestion time must be considered in order to synchronize server-side simulations and client-side representations. This synchronization becomes particularly critical in a multi-user environment where all players in the same place need to see the "same" world at the same time, even if every client has different network bandwidth. For this reason, a handshaking technique was used in the actual implementation of the HPSP. When a client is finished rendering the previous frame and is ready to receive the next frame, a "ready" signal is sent to

[39] a remote procedure call protocol which uses XML to encode its calls and returns
[40] http://www.w3.org/Protocols/rfc2616/rfc2616.html
[41] Gzip encoding/decoding functions are included in both PHP default package and ActionScript library.
[42] Actual CPU consumption ratio depends on file size, frequency of operation and the CPU type.
[43] Xmldiff open-source algorithm was used for Prototype 1.0

a server, as illustrated in Figure 5. The server transfers the most recent frame of the hyperplace only to clients in "ready" state and internally assumes that they will remain "busy" before they explicitly send the "ready" signal again. Under this handshaking protocol, clients with lower bandwidth represent hyperplaces with lower frame refresh rate in order to maintain synchronization with other clients.

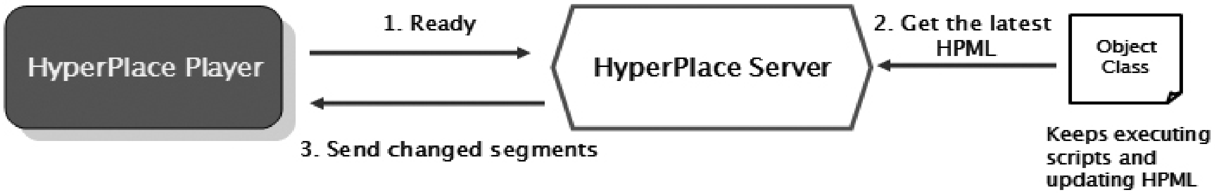
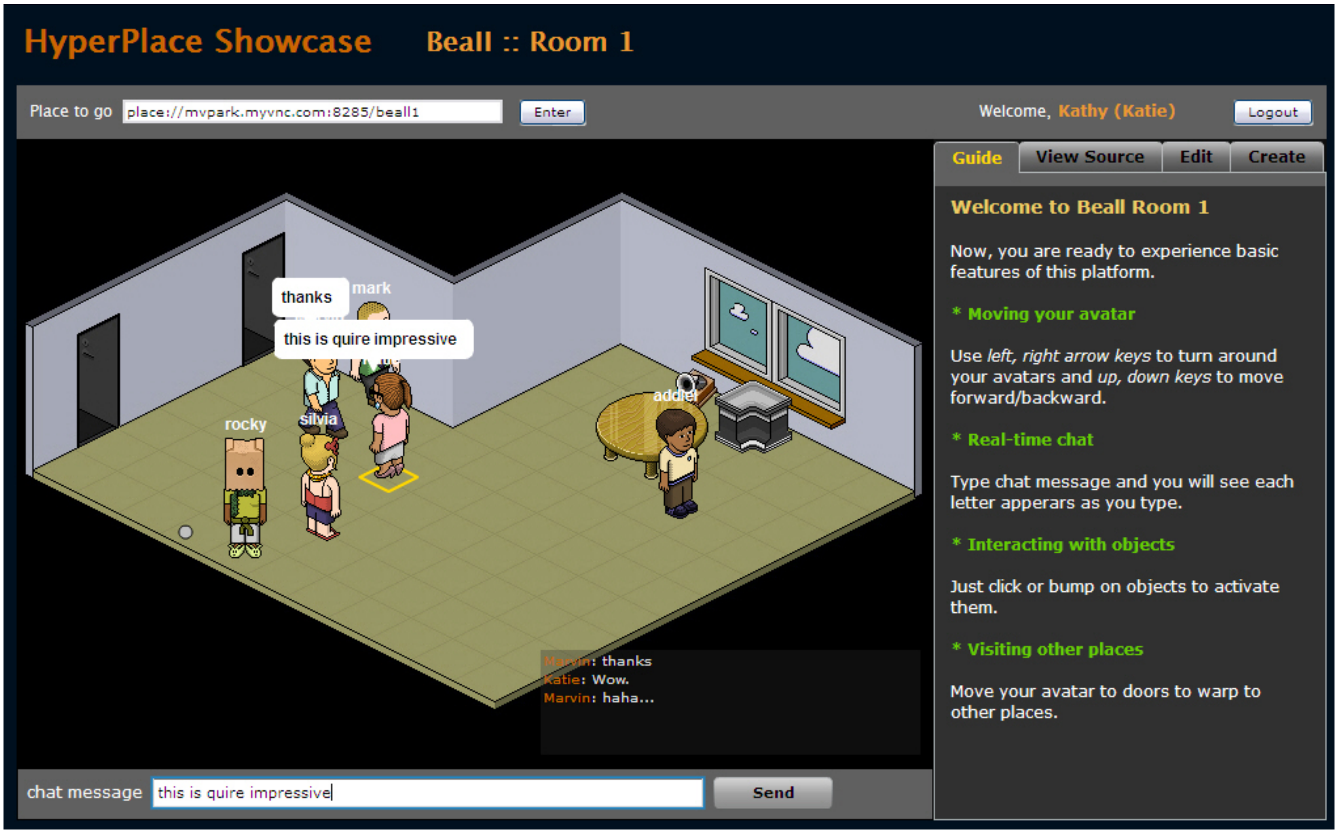


Figure 5: Handshaking between a client and a server

3-4-3. Real-time Conversation

Since the HyperPlace platform affords full-duplex persistent communication in a web browser, it can be used to emulate face-to-face conversation, which is often treated as the “gold standard” for real time interpersonal communication. The HyperPlace players treat a user's single keystroke as an independent event and sends it to connected servers. The HyperPlace servers dispatch these events to corresponding agent objects so that they can add (or delete) a new character to existing chat messages. as depicted in Figure 6. Compared to traditional turn-taking conversation, this "text-as-you-type" messaging increases the fluidity of real-time interaction, making it easier to figure out a speaker's intention before each sentence is complete [xxvii].



place://mvpark.myvnc.com:8285/beall1

Welcome, Kathy (Katie)

Welcome to Beall Room 1

Now, you are ready to experience basic features of this platform.

* Moving your avatar

Use *left, right arrow keys* to turn around your avatars and *up, down keys* to move forward/backward.

* Real-time chat

Type chat message and you will see each letter apperars as you type.

* Interacting with objects

Just click or bump on objects to activate them.

* Visiting other places

Move your avatar to doors to warp to other places.

Marvin: thanks
Katie: Wow.
Marvin: haha...

this is quire impressive

Figure 6: Each character of chat message appears as a user types it

3-5. Dynamics of Hyperplace Objects

Visual representation of a hyperplace is constructed by individual representation of the objects. Each object, the unit of ownership, is created and owned by an individual user. It can be an avatar, an NPC (non-player character) or any hypermedia artifact that can be represented by hyperplace players. This section covers both internal mechanisms of hyperplace objects and external interfaces to edit them.

3-5-1. Object Programming Interface

As previously discussed, an HPML representation of a hyperplace should reflect real-time changes of objects such as motions of user agents. Thus, an HPML representation needs to be dynamically generated by a hyperplace server. This can be implemented using techniques similar to those employed by web servers to dynamically generate HTML data. Such dynamic HTML is usually generated by server-side scripts such as Perl, ASP, PHP or JSP, all of which have a wide range of string manipulation functions and libraries for database and file management.

Among the server-side script languages, PHP was chosen for scripting hyperplaces on consideration of the requirements for the platform discussed in the section 3-1. PHP is one of the most popular web script languages[44] and has sufficient developer communities to share and reuse source code. Moreover, it is free, platform-independent and object-oriented. ASP runs only on Microsoft Windows OS and Perl and JSP are less popular than PHP. Compiler languages such as C/C#/C++/Java were not considered here because it is much easier to implement code mobility using interpreter languages and this will be discussed later in this section.

End-users can program their own places or objects on the HyperPlace platform with PHP classes. Each hyperplace object class has the common interface methods, including "*draw*" that generates HPML tags representing the current state of the corresponding object. A *"place"* class has the same *draw* method, but it collects HPML tags from the objects that belong to the place, by executing one after another, as well as representing the place itself (Figure 7).

The object class library provides content creators with a base class interface for hyperplace programming. A user created object inherits one of the classes from the common object library (shown in Figure 8), which should be installed on every HyperPlace server. For example, users can inherit the "*Agent*" base class to create their own avatar objects, and they can fill in skeleton methods such as "*draw*" or "*handleEvent*". Appendix C illustrates an example code for custom object programming. User created object classes also inherit properties from their base classes, which are usually used for storing HPML properties. Figure 9 illustrates an example of properties and methods that an *"Agent"* class inherits from its ancestor classes.

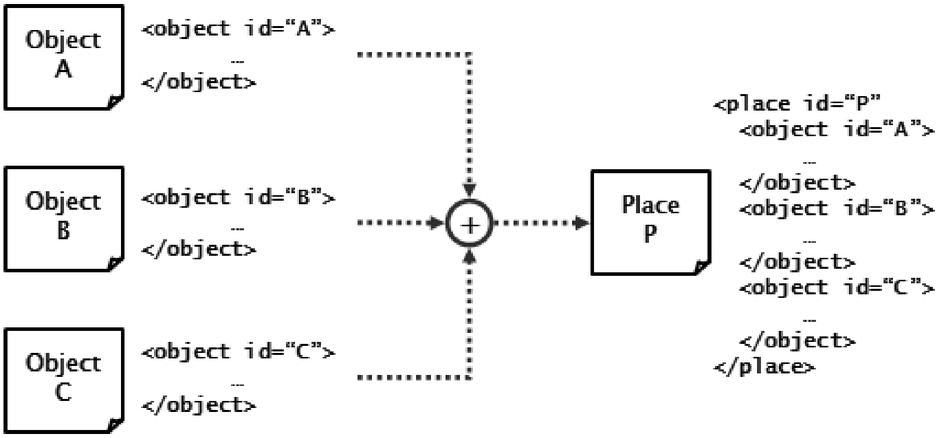


Figure 7: Composition of an HPML from objects

[44] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

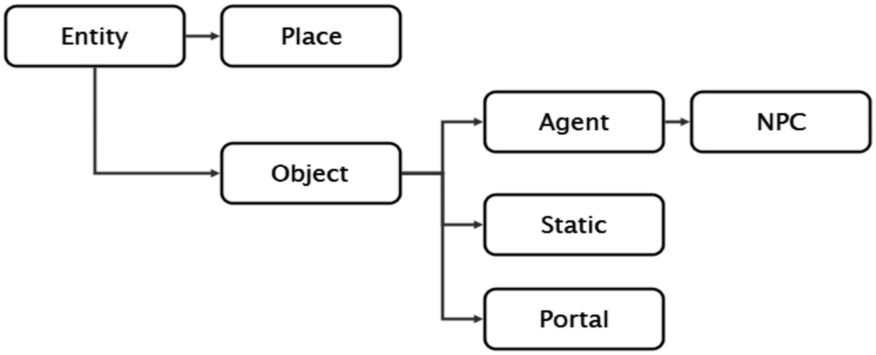


Figure 8: Hierarchy of PHP Object Library

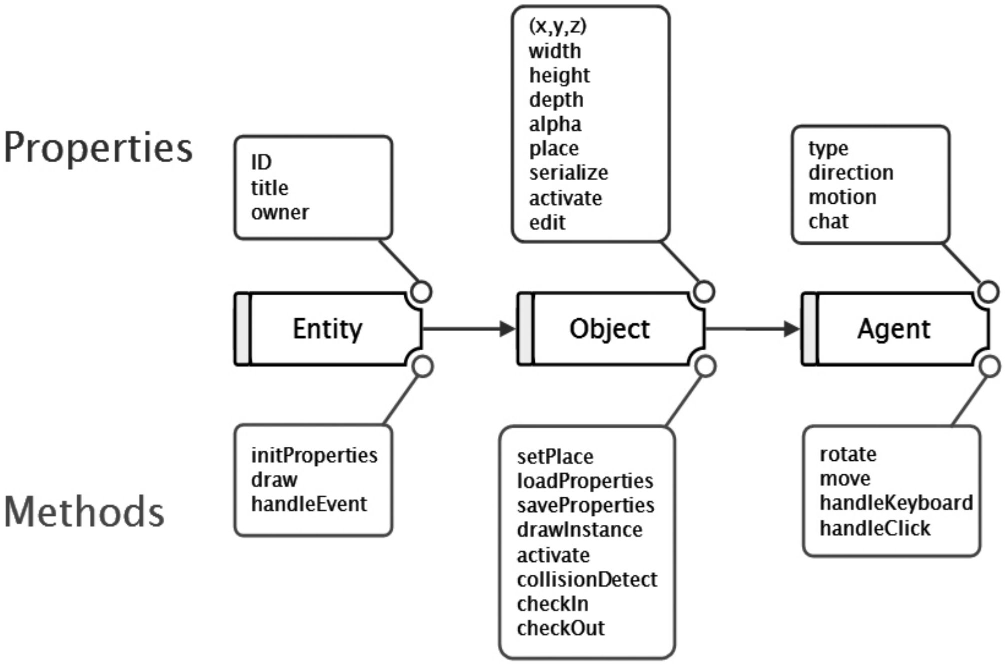


Figure 9: Inherited Object Interface

3-5-2. Simulation of a Persistent World

An interconnected network of hyperplaces forms a persistent world. A *persistent world* is defined as a virtual world that continues to exist even after a user exits the world, and that user-made changes to its state are, to some extent, permanent[45]. To simulate a persistent world, the HyperPlace servers keep executing "*draw*" methods of local place classes in every update period predefined by a configured frame rate. This execution continues regardless of the existence of clients to serve.

[45] http://en.wikipedia.org/wiki/Persistent_world

3-5-3. Locality of Hyperplace Objects

As depicted in Figure 1, the HyperPlace platform is designed to decentralize server components. For this reason, it does not have a central database or dedicated data storage. Instead, hyperplace object classes are stored in the local file system of the distributed HyperPlace servers. Each (hyper)place corresponds to each folder in a server file system. Naturally, objects belonging to a hyperplace are physically stored in the same folder. While it may not be technically mandatory, this 1:1 correspondence between a logical entity and a physical storage unit helps content creators to more intuitively understand and build a mental model of the system. The metaphor of a (hyper)place is incarnated as a folder and the ownership of an object is embodied as a file.

Besides the understandability aspect, this approach—storing objects from the same hyperplace in one server also reduces overall communication costs due to the locality of interactions in a hyperplace. People in the same place communicate more frequently with each other than they do with people in remote places. Objects in close vicinity often interact/interfere with each other while they seldom influence objects out of their visibility range. Suppose that three avatar objects in the same hyperplace are stored in three separate servers. Then, every user interaction should go across the network to synchronize all the servers and clients. Internal network traffic will be exponentially increased as more distributed objects join the communication. Hence, it is statistically more efficient to keep logically neighboring objects physically close. If a local object A needs to communicate with a remote object B for a prolonged period of time, it is more efficient to move A to B's place rather than allowing them to communicate remotely.

3-5-4. Mobility of Hyperplace Objects

A hyperplace object can instantly move from one place to another either by user control or by internal logic. Hence, it is a form of mobile agent; defined as "*program code and the associated internal state which can move between computers in a network* [xxviii]." In this sense, the HyperPlace platform can be categorized (and utilized) as a mobile agent framework implemented on the popular web environment. Like other mobile agents, hyperplace objects maintain their internal state between migrations. This is implemented using *serialization/unserialization* methods built-in default PHP library[46]. Before a hyperplace object moves to another place, it stores its internal properties to a data file[47]. Migration of an object is simply accomplished by moving a code file and a data file to a folder corresponding to the new hyperplace[48]. When the HyperPlace server receives an incoming object, it loads the object class to its memory and unserializes saved properties in order to restore the last state of the object.

The HyperPlace platform could implement this mobility of hyperplace objects by fully utilizing the flexibility of PHP as an interpreter language. Compiler languages require complete class definitions at compile time. To include a new class, the entire application should be terminated and recompiled. Meanwhile, a PHP application can be programmed so that it dynamically loads classes during runtime without stopping its execution. For this dynamic class loading, *eval* method was used to transform an arbitrary data string into an executable program code[49]. A PHP magic method *"__autoload"* was overridden to actually load dynamically requested class files[50].

This *code mobility* has been studied as a new paradigm for programming in large-scale distributed settings like the Internet [xxix]. Viewed from a software engineering perspective, this persistence of memory in a changing environment can be used to develop an AI program that shows more autonomous behaviors[51]. For example, a hyperplace object may accumulate historical data acquired

[46] http://www.php.net/serialize

[47] Base object class in the hyperplace object class library has methods saveProperties and loadProperties to serialize and unserialize properties of an object.

[48] After migration, hyperplace servers keep object files in a backup folder with timestamps for backup purpose.

[49] http://www.php.net/manual/en/function.eval.php

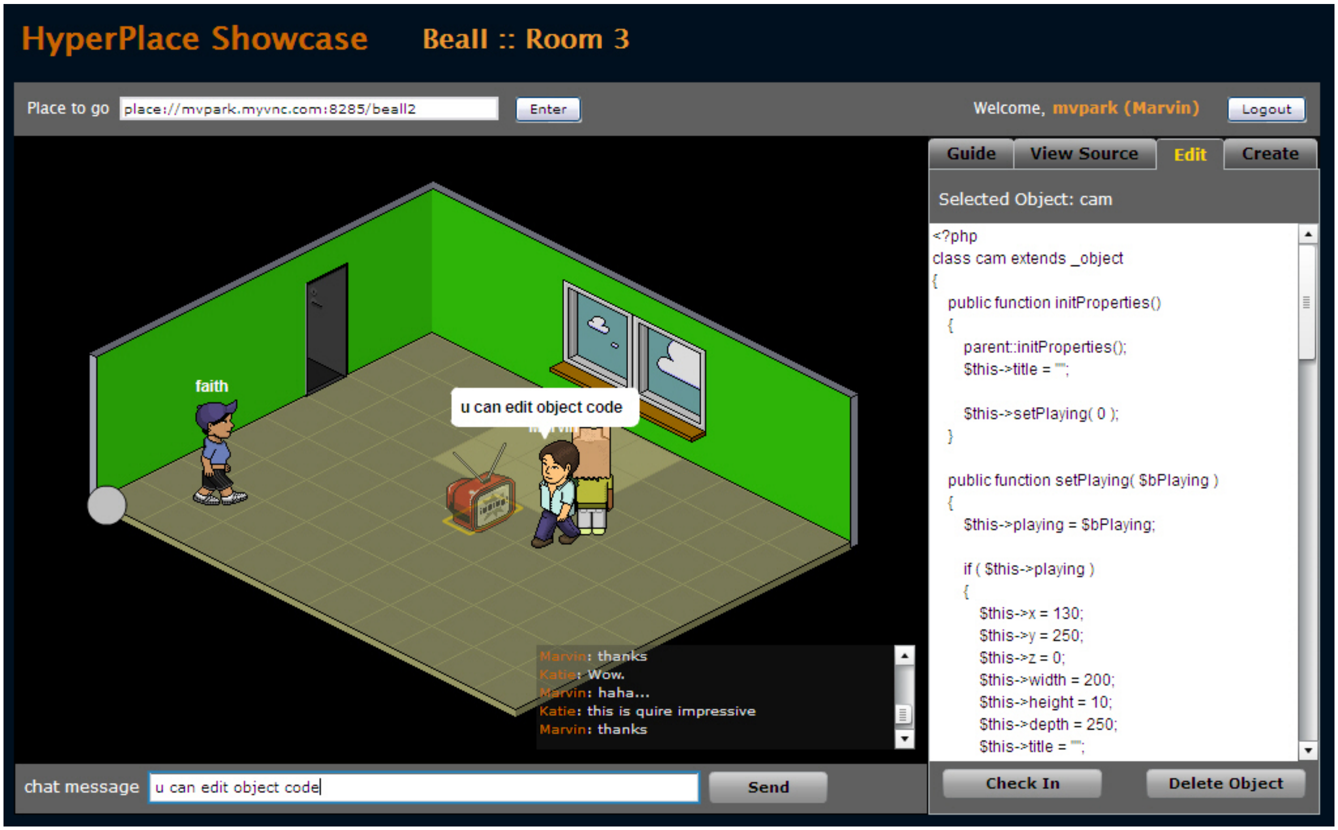
[50] http://www.php.net/__autoload

[51] http://en.wikipedia.org/wiki/Autonomous_agent

from the servers it has visited and other objects it has interacted with before. If the object's behavior is determined by the data it has collected, it may be programmed to evolve and adapt itself to its surroundings.

3-5-5. Real-time programmability

The HyperPlace platform was designed and implemented as an integrated development environment (IDE) for content creators, as well as an interactive media platform for players. Hyperplace object classes can be accessible and modifiable by their owners while simulation of the hyperplace continues. As following Figure 10 shows, the owners of an object can view and check out its code with an IDE interface of the HyperPlace players. When checked out, the object is in frozen state and it can not be checked out again; it stops executing its code until it is checked in. As soon as the owners check in the modified object classes, the objects resume execution and the modifications are applied immediately.



Place to go: place://mvpark.myvnc.com:8285/beall2

Selected Object: cam

```
<?php
class cam extends _object
{
   public function initProperties()
   {
      parent::initProperties();
      $this->title = "";

      $this->setPlaying( 0 );
   }

   public function setPlaying( $bPlaying )
   {
      $this->playing = $bPlaying;

      if ( $this->playing )
      {
         $this->x = 130;
         $this->y = 250;
         $this->z = 0;
         $this->width = 200;
         $this->height = 10;
         $this->depth = 250;
         $this->title = "";
```

Marvin: thanks
Katie: Wow.
Marvin: haha...
Katie: this is quire impressive
Marvin: thanks

chat message: u can edit object code

Figure 10: Users can access and modify object code during run-time

This real-time programmability requires more than dynamic class loading. A loaded class needs to be "unloaded" from the system memory and "reloaded" to apply changes. However, dynamic unloading of already loaded classes is not supported by PHP language. For instance, a class ABC can be included and executed during runtime, but another class with the same name (ABC) is not allowed to be loaded again. The HyperPlace platform emulated this dynamic reloading of classes by coupling a "*prototype*" class with an "*instance*" class. A prototype class is a normal class source file that users can access and modify. A coupled instance class is a time-stamped hidden file which the HyperPlace servers actually load to their memories. Contents of both classes are internally synchronized to be identical but they have different names. For example, a prototype class ABC may have an instance class ABC.11326 ,at a given time, where the attached number stands for the last modified time of the prototype class. Since every modification of a prototype class changes the name of the instance class, it will be reloaded to the server system. The HyperPlace server maintains an array that associates prototype classes with corresponding instance classes so that object programmers can access the running instance classes with the names of prototype classes.

CHAPTER 4.

EVALUATION OF THE PROTOTYPE

The requirements of the platform (section 3-1) are the basic criteria for evaluating the implemented prototype. This chapter evaluates the prototype based on the requirements and identifies technical problems found during implementation and exhibition. However, overall system performance was not measured yet since there is no quantitative measures or criteria for this platform. This should be further considered for the larger scale deployment of the system.

4-1. Requirement Satisfaction

The following analysis shows that the prototype meets roughly 88% of the initial requirements. This analysis also guides what needs to be done for next revisions to fully satisfy platform requirements.

4-1-1. Accessibility: 100%

The implemented prototype of the HyperPlace platform runs on a legacy web platform without proprietary software installation. Both the client and server components were tested to run on average configuration of PC hardware (Intel Pentium 1Ghz-level CPU, 1MB RAM) without delay of performance. As mentioned in section 3-2-2, they both run on all major operating systems (Microsoft Windows, Apple Mac OS X, and Linux). With the data compression and handshaking technique (discussed in chapter 3-4), average streaming rate of data does not exceed the capacity of the bandwidth[52]. Thus, the prototype satisfies the conditions to meet the “accessibility” requirements suggested in section 3-1-1.

4-1-2. Ease of Learning:70%

Current implementation of the content authoring relies on the text-based programming of HPML and PHP objects. The base PHP object library minimizes user programmers' responsibility so that they need to modify only input and output string handlers. Nevertheless, it would be controversial whether object programming in PHP language is an easy way for normal users who are unfamiliar with programming. While PHP has larger user group than other web script languages and abundant online resources are available for PHP programmers, it would be better to provide visual tools with novice programmers. However, in so far as the representational method of hyperplace is textual HPML and hyperplace object has access to the open Web resources, the textual programming method provides more flexibility and versatility than the indirect visual programming tools. Thus, visual tools should be developed as an auxiliary tool to support codeless, instant creation of objects.

4-1-3. Decentralization: 80%

Each HyperPlace servers are identical in terms of system architecture. They all have the same base object library and execute the same main server code. Only mobile objects move from a server to another. In this sense, it is basically a “decentralized” and “distributed” system just like Web. However, the cross-reference of objects between servers requires the Global Location Service (GLS) as introduced in section 3-2-3. It functions as a real-time DNS (Domain Name Service) for distributed objects. By objects' real-time mobility, it is difficult to duplicate and synchronize data entries in one GLS to another server. For this reason, the prototype has only one GLS server. This also need to be decentralized and distributed over the network in the future.

[52] Difference-based data compression is a heuristic algorithm, so the actual volume of traffic may vary according to content.

4-1-4. Real-time Interactivity:100%

Unlike HTTP, HPSP is a connection-keeping protocol on which full-duplex communication can be set up. The prototype fully implemented the specification of the protocol, and thus satisfies the requirement of “real-time interaction” including real-time chat and event handling (see section 3-2 and 3-4).

4-2. Technical Problems

While the prototype showed the overall feasibility of the system, the shortcomings of the current approach also became apparent throughout it. Particularly, following technical problems are exceptional ones that conflict with the design philosophy of the platform. It is the goal of next revision of the HyperPlace to solve these problems without sacrificing ideologies of the platform.

4-2-1. Security

The current real-time programming environment of the HyperPlace can not distinguish a malicious hackers' (or crackers') code from an innocent content creators' code. Since user programmable mobile objects have the right to access the resources of the operating systems they are running on, they may be used to spread spywares[53] or viruses. Their versatility and mobility will certainly worsen the situation if used for malicious purposes.

In order to protect the system from the attack of malicious objects, the platform must have a mechanism that verifies access level of each object and limits individual operations. For example, the security sandbox[54] model used for Java and Flash platform may be applied to the HyperPlace. However, the sandbox typically provides a tightly-controlled set of resources for guest programs to run in, such as disk and memory. Network access, the ability to inspect the host system or read from input devices are usually disallowed or heavily restricted. In this sense, enhancing security through virtualization (abstraction of computer resources) significantly limits freedom and functionality of the object programs.

4-2-2. Load Balancing

Another technical challenge is to load balance distributed hyperplace servers where all the objects on the network can instantly move to and crowd in a specific server at a given time while all the other servers remain idle. This difficulty of load balancing is counterbalanced by the advantages of local communication (discussed in section 3-5-3).

When a crowded server crashes or fails to respond to clients due to heavy volume of internal operations, the owners of the objects can not access their own objects since the object files are currently stored in the local file system of the crashed server. While previous servers keep the last status of the objects before they move to the current server, modifications made in the crashed server are lost. Hence, it is important for servers to balance load and maintain idle process time to serve incoming requests promptly.

In order to balance load and maintain idle time, each HyperPlace server should keep monitoring their status and limit the entrance of incoming objects when their system load exceeds a certain percentage of the maximum capacity. Theoretically, this policy works well for an evenly distributed network, which seldom exists in reality. It may significantly limit the mobility of the hyperplace objects because it will probably block the entrance of “the most visited places.”

Therefore, an efficient distributed hosting (hosting a hyperplace by multiple servers) method should be developed to balance load among servers.

[53] Spyware is computer software that is installed surreptitiously on a personal computer to intercept or take partial control over the user's interaction with the computer, without the user's informed consent.

[54] http://en.wikipedia.org/wiki/Sandbox_(computer_security)

CHAPTER 5.

APPLICATIONS OF THE HYPERPLACE PLATFORM

As mentioned in chapter 1 and chapter 3, hyperplace media and the HyperPlace platform naturally constitute an online collaboration site. They can also be used for building a virtual world. This chapter covers further considerations to be taken while building such applications on the HyperPlace. As hypertext and the Web does not limit the range of applications, it is not necessary to confine the type or genre of the possible contents on a generic media platform. Nevertheless, a few additional applications are introduced here in order to highlight the uniqueness of the HyperPlace platform compared to legacy media platforms.

5-1. CSCW/CSCP Environment

One of the most primitive forms of hyperplace constitutes a multi-user chat environment with controllable avatars. Such real-time group communication environment can be utilized as an online community space without significant modification. In order for this basic hyperplace to be used as a CSCW environment, data storage and tools for workflow management must to be added. It is possible to create a hyperplace object that stores and transfers data (or documents). Such objects may be visualized as documents, books, folders, or a container box. In this case, the hyperplace also functions as a GUI of the CSCW system. For instance, moving a document into a box may actually move the corresponding document file to a specific location. Typical CSCW workflow also requires a non-visual information management system to control schedules, resources, and members. These invisible entities can be implemented by modifying the "place" object, where the global logic of the hyperplace is programmed. (See chapter 3-5). It is also possible to extend an existing Web-based CSCW system. If a CSCW system is already implemented on the Web, hyperplace can integrate with it through a Javascript interface (See section3-2-1). In addition to textual collaboration systems, hyperplace visualizes activities related to documents. Studies on CSCW environment have shown that by visualizing collaborator behavior, *social translucency* is enhanced and usability of the system improves. [[xxx]] [[xxxi]]

Another social application of such a collaborative place is an environment for CSCP (Computer Supported Cooperative Play). Collaboration platforms usually serve co-workers, not co-players. It is for this reason that the term CSCP is used less frequently than CSCW. Collaboration requires players to have cooperative minds rather than competitive attitudes, a principle employed in most game narratives. From this aspect, CSCP is different from most of the multiuser online games. It is more closely aligned to Olderber's idea of the “third place” [xxxii] in which an individual's individuality and personality are celebrated.

In CSCW, as an extension of real life, work role is particularly important and therefore self-in-the-world is closely connected to the online embodiment of self. In CSCP, self-in-the-world and the online embodiment of self are less tightly coupled and less constrained by fixed roles and identities such as those of the "workplace". Therefore, CSCP environments support a more fluid and organic sense of self. This is illustrated by the fact that, in CSCP, many instances of role-playing and multiple identities have been reported. [[xxxiii]]

Building CSCP environments does not require prototypical functions. Rather, It is about maintaining a playable mode, a task that demands more flexibility than a format. In this regard, a player-driven hyperplace is a technically appropriate medium to build such online third places.

5-2. Player-driven Virtual Worlds

In chapter 1-3, I defined hyperplace media as the building blocks of virtual worlds. Particularly, hyperplace would be a good medium to build a player-driven virtual world where players can create object, NPCs and events with maximum freedom. Yet, the current prototype of the HyperPlace platform does not have a 3-D rendering engine that presents a realistic replica of the physical world. It is possible to enhance the graphic engine, but run-time manipulation of larger amounts of data is more challenging. By its dynamic programmable nature, prediction of incoming data is technically impossible[55], so it does not preload (pre-install) graphic data like other virtual world client programs. All the data must be downloaded on the fly with minimum waiting time.

However, in building a more persuasive reality, the social aspect of a medium is more important than its representational aspect. This is why players can deeply immerse themselves into purely textual MUD, which has no visual representation but is relatively rich in communication features. Ironically, graphical illusion is the only "fake" part of a virtual world because it can be built only through pre-designed computer simulation. The "reality" of virtual worlds, on the other hand, emerges from dynamic interactions among the human players inside them. The players coordinate and organize communities and compete and cooperate with each other in order to struggle against fearful odds. [xxxiv] These interactions are what really happen in the virtual environments, not simulated ones. Through these social interactions, players reassure their identities reciprocally and reinforce the initial weak sense of individual presence. [xviii]

5-3. Venue for Online Artwork

An increasing number of galleries in Second Life are exhibiting online artwork [xxxv]. Some artists such as DC Spensley, known by his avatar name Dancoyote Antonelli, create artwork and exhibit it in only in Second Life. They even sell their work in exchange for Linden dollars, which can ultimately be exchanged for real currency. For them, Second Life is the venue where they work, install, exhibit, and sell their artwork. As more artists live and work in a digital landscape, the display of online art in a virtual world may develop into a new genre of art.

Artists can take advantage of open connectivity when they craft their artwork on the HyperPlace platform. Since hyperplace objects are written in PHP script language and the output canvas is a Web browser, they can be easily integrated with numerous Web APIs and mash-ups. Most widely used Web mash-ups and APIs include geographical databases, photographs, and social networks. For example, an artist may create an interactive artwork that utilizes relevant photographs or the social network of an audience.

Unlike online venues in a proprietary virtual world like Second Life, exhibition on the HyperPlace is not restricted to its residents. Since these exhibitions run on normal Web browsers, they are sharable and accessible to more people.

[55] While the prediction is impossible, internal cache algorithm is implemented in the HyperPlace player client to minimize data download time.

5-4. Interactive Theater

Drama consists of both characters and story. The concept of interactive drama contains a contradiction in it since characters are played by autonomous online actors and story comes from an author's intention. If the characters do not follow a predefined plot, the story cannot proceed as intended. Thus, interactivity and narrative are not easily reconcilable. Ernst Adams stated that “interactivity is almost the opposite of narrative; narrative flows under the direction of the author, while interactivity depends on the player for motive power. [[xxxvi]])” As this statement implies, the most significant problem of combining interactivity and narrative in a medium is maintaining a coherent narrative structure within a player-controllable environment. This conflict is about freedom vs. control, bottom-up vs. top-down, performance vs. representation, production vs. consumption, and relaxation vs. tension.

However, interactive drama can be defined in a completely different way. For example, it could be defined as a live performance of role played by individual participants. This first-person participation is what users can **experience** through virtual worlds. Suppose that there is a role-playing virtual world and it automatically assigns a different role to each player. When agents in the virtual world are given roles, quests, abilities, costumes, and items, all of which are aligned to characterize them, they will become **characters**. Their individual pursuits of goals can be an open-ended **story** if their quests and resources are designed to foster a dramatic structure (or a dramatic arc), for example, conflict among characters. In this sense, most virtual worlds, including the HyperPlace, can become online theaters for an improvised performance of interactive drama. In some interactive dramas such as Michael Mateas' *Façade*, believable agents act like human characters [[xxxvii]]. The hyperplace “Agent” object can be programmed to act like a believable agent with behavioral AI algorithms [[xxxviii]]. These believable agents are better than human characters when they play some tedious and less-significant characters such as gatekeepers. They can be also used to maintain the coherence of narratives by preventing undesirable unfolding of the story or initiating events necessary to the narrative.

5-5. Multiuser Programming Environment

As described in section 3-5-5, multiple users can simultaneously program objects in the same hyperplace. The programmers can communicate with each other and review each other's source code in real time. This unique setting can be utilized to increase productivity. For example, “pair programming” has been introduced as a way of practicing extreme programming [[xxxix]], a new software development methodology that emphasizes team work and feedback to increase productivity. Pair programming is a software development technique in which two programmers work together at one keyboard. One types in code while the other reviews each line of code as it is typed in. The two programmers switch roles frequently.[[56]] Thus, it is possible for two programmers to practice online pair programming on the HyperPlace platform.

Sometimes, programming is a playful activity and programmers often absorb themselves in the world of logical structure they are building. This highly productive yet enjoyable experience is what Csikszentmihalyi calls “optimal experience” or “moment of flow.” [[xl]] In this sense, another application of a multiuser programming can be named “Programming for Fun.” For example, programmers may cooperate or compete with each other to create a stronger NPC creature that defeats other NPCs (or user agents). In this case, the most enjoyable part is programming something very rapidly in a creative way. This kind of creative fun has been already practiced on MOO platforms [[xli]].

[56] http://en.wikipedia.org/wiki/Pair_programming

CONCLUSION

In this paper, I introduced a networked, participatory medium called hyperplace, and the HyperPlace platform that can be used by both content creators and players to build or utilize hyperplaces. The key ideas of the platform include easy authoring of the virtual place, decentralized ownership, real-time interaction, and mobility of code. The implemented prototype of the HyperPlace platform proves that all the basic ideas of the hyperplace can be implemented on the Web platform without proprietary software installation.

Meanwhile, the shortcomings of the current approach became apparent. System-wide problems include security issue and load balancing as discussed in chapter 4. Its also has the limitation of representing realistic graphics as described in section 5-2, which ultimately limits the type and quality of the visual experience.

Next upgrade of this platform should be directed to tackle these technical challenges. I currently do not have good ideas to solve (or improve) the limitation of the current system design. Yet, this project will be open to community as an open source project with manuals. Web administrators who want to host hyperplaces can freely download the project files and allow other players to come in and build their own places and objects. As the hyperplace network grows and more people try to solve the problems, better solutions would be suggested. It would be ideal if the platform itself can evolve by the collective intelligence and better serve its content.

APPENDIX A. XML SYNTAX OF HPML

```
<place
      id="globally_unique_id"
      owner="[email_address]"
      title="My First Placce (display)"
      width="{pt}"
      height="{pt}"
      bgcolor="#cc6600"
      isometric="{false|true}"
      originx="{pt}"
      originy="{pt}"
      stylesheet="[url]" />
      <permission />
      <background
            src="[image_url]"
            width="200"
            height="100"
            x="0"
            y="0"
            align="{left|center|right}"
            valign="{top|middle|bottom}"
            tile="{no|yes}" />

      <sound
            src="[sound_url]"
            loop="{0|1|2|...|*}" />

      <jscript method="func_name( parameters )" />

      <keyboard
            charcode="{none|all|specific_chracter}"
            shortcut="{none|shift+z}"
            handler="{server|client}"
            menu="menu_id"
            link="{place_url|http_url}"
            linktarget="window_name"
            jscript="js_function( parameters )"
            controller="{all|owner|others}" />

      <mouse
            action="{rollover|rollout|leftdown|leftup|rightclick|dblclick}"
            handler="{server|client}"
            menu="menu_id"
            link="{place_url|http_url}"
```

```
        linktarget="window_name"
        jscript="js_function( parameters )"
        controller="{all|owner|others}" />

<menu
        id="menu_id_unique_in_object">
        <menuitem
                +id="item_id_unique_in_menu"
                +handler="{server|client}"
                +title="take this"
                icon="[url]"
                link="{place://|http://}"
                linktarget="window_name"
                jscript="js_function( parameters )"
                menu="submenu_id" />
</menu>

<object
        id="globally_unique_id"
        owner="[email_address]"
        title="Nick name for (display)"
        bgcolor="[color]"
        width="{*|100}"
        height="{*|100}"
        x="20"
        y="20"
        z="20"
        drag="{fasle|true}"
        alpha="10%">

        <img
                src="[image_url]"
                width="200"
                height="100"
                layer="{0|1|2|...}"
                x="0 (relative)"
                y="0 (relative)"
                z="0 (relative" />

        <canvas
                x="0 (relative)"
                y="0 (relative)"
                z="0 (relative)"
                width="200"
                height="100"
```

```
bgcolor="[color]"
border="{false|true}"
bordercolor="[color]" >
<beginFill
       color="[color]"
       alpha="0...100" />
<clear />
<curveTo
       contrlX="[coord]"
       controlY="[coord]"
       anchorX="[coord]"
       anchorY="[coord]" />
<drawCircle
       x="[coord]"
       y="[coord]"
       radius="30" />

<drawEllipse
       x="[coord]"
       y="[coord]"
       width="30"
       height="20" />

<drawRect
       x="[coord]"
       y="[coord]"
       width="30"
       height="20" />

<drawRoundRect
       x="[coord]"
       y="[coord]"
       width="30"
       height="20"
       ellipseWidth="30"
       ellipseHeight="20" />

<endFill />

<lineStyle
       thickness="{0...255}"
       color="[color]"
       alpha="{0...100}"
       pixelHinting="{false|true}"
       scaleMode="{normal|none|vertical|horizontal}"
```

```
                caps="{none|round|square}"
                joints="{miter|round|bevel}"
                meterLimit="{1...255}" />
         <lineTo
                x="[coord]"
                y="[coord]" />
         <moveTo
                x="[coord]"
                y="[coord]" />
   </canvas>
<video
         src="[video_url]"
         width="200"
         height="100"
         layer="{0|1|2|...}"
         x="0 (relative)"
         y="0 (relative)"
         z="0 (relative"
         repeat="{1|2|...|*}" />
   <sound />
   <keyboard ... />
   <mouse ... />
   <menu ... />
   <call ... />
   <text
         x="0 (relative)"
         y="0 (relative)"
         z="0 (relative)"
         width="200"
         height="100"
         bgcolor="[color]"
         bordercolor="[color]"
         align="{none|left|center|right}"
         color="[color]"
         font="face name"
         size="face size"
         bold="{false|true}"
         italic="{false|true}"
         underline="{false|true}"
         stylesheet="[url]"
         wordwrap="{false|true}"
         target="{_self|_blank|...}"
         url="[url]"
         feature="open_window_feature" >
                <![CDATA[content (HTML tags)]]>
```

```
        </text>
        <chat
            alpha="[percent]"
            stylesheet="[url]" >
                <![CDATA[content (HTML tags)]]>
        </chat>
    </object>
</place>
```

APPENDIX B. HPSP PROTOCOL SPECIFICATION

HPSP protocol syntax is similar to XML-RPC. Both request commands and responses are formatted with XML tag syntax: <command parameter=”value” />

command	parameters	description	response
ready	<none>	handshaking signal	<none>
test	<none>	test connectivity	<test value=”OK” />
policy-file-request	<none>	Flash sandbox security check	policy file (xml)

event	type=mouse, keyboard, ...
	value=left, right, ...
	target=object_id
		send a client event to an object (or a place)	<none>

create	id=object_id
	template=file
		create a new object	<create id=”object_id” result=”success|failure” reason=”...”>

delete	id=object_id	delete an object	<delete id=”object_id” result=”success|failure” reason=”...”>
view	id=object_id	View object source code	<view id=”object_id”> <![CDATA[content (object source code)]]> </view>

checkin	id=object_id
	value=new_source
		check-in modified object code	<checkin id=”object_id” result=”success|failure” reason=”...”>

checkout	id=object_id	check-out an object code	<checkout id=”object_id” result=”success|failure” reason=”...”>

login	id=user_id
	password=md5hashed
		login as a specific player (and avatar)	<login id=”object_id” result=”success|failure” reason=”...”>

logout	id=user_id	logout	<logout id=”object_id” result=”success|failure” reason=”...”>
enter	place=place_id	enter a specified hyperplace	stream of HPML
bye	<none>	logout and close connection	<none>

APPENDIX C. SAMPLE HYPERPLACE OBJECT CODE

Sample 1. audio object

Following code is for an audio object that players can turn on by clicking it. When turned on, it bounces on the floor and plays an MP3 file.

```
<?php
class audio extends _object
{
    public function initProperties()
    {
        parent::initProperties();

        $this->x = 200;
        $this->y = 10;
        $this->z = 0;
        $this->width = 19;
        $this->height = 19;
        $this->depth = 20;

        $this->title = "";
        $this->img = array();
        $this->img[ 0 ] = getImgPath( "audio_off.gif" );
        $this->img[ 1 ] = getImgPath( "audio_on.gif" );

        $this->audio = getSoundPath( "reflection.mp3" );

        $this->playing = 0;
        $this->direction = 1;
    }

    public function draw()
    {

        $xml = singleTag( "img", "src", $this->img[ $this->playing ], "x", 0, "y", 0 );

        if ( $this->playing )
        {
            $xml .= singleTag( "sound", "src", $this->audio, "loop", "1" );

            $this->z += $this->direction;
            if ( ( $this->z > 20 ) || ( $this->z < 0 ) ) $this->direction *= -1;
        }

        return $xml;
    }

    function handleEvent( $eventType, $eventValue )
    {
        if ( $eventType == "click" )
        {
            if ( $this->playing ) $this->playing = 0;
            else $this->playing = 1;
        }

        return true;
    }
}
?>
```

Sample 2. a networked object

Following code constitutes a networked ball object that randomly moves in a place. It receives a color value from another networked artifact, called gravitable made by Mark Roland, and reflects it to the color of the ball.

```
<?php
class ball extends _npc
{
    public function initProperties()
    {
        parent::initProperties();

        $this->x = 250;
        $this->y = 150;
        $this->z = 0;
        $this->width = 40;
        $this->height = 40;
        $this->depth = 40;
        $this->title = "";

        $this->radius = 20;
        $this->direction = 4;
        $this->change = 1;

        $this->color = "#C0C0C0";
        $this->lastUpdate = 0;
    }
    public function draw()
    {
        $curTime = getSysTime();
        if ( $curTime - $this->lastUpdate >= 10 )
        {
$this->color =
file_get_contents( "http://www.markroland.com/gravitable/demo/gravitable_hex.php" );
            $this->lastUpdate = $curTime;
        }

        $tag = openTag( "canvas", "x", 0, "y", 0, "width", $this->radius * 2, "height",
$this->radius * 2 );
        $tag .= singleTag( "beginFill", "color", $this->color, "alpha", 100 );
        $tag .= singleTag( "lineStyle", "thickness", 1, "color", "#000000", "alpha",
100 );
        $tag .= singleTag( "drawCircle", "x", $this->radius, "y", $this->radius, "radius",
$this->radius );
        $tag .= closeTag( "canvas" );

        return $tag;
    }

    public function simulateMove()
    {
        if ( $this->radius > 15 ) $this->change = -1;
        if ( $this->radius < 5 ) $this->change = 1;
        $this->radius += $this->change;

        $this->move( true );
    }
}
?>
```

i Yi-Fu Tuan, Steven Hoelscher, "Space and Place: The Perspective of Experience", University of Minnesota Press, pp. 4-10 (February, 2001)

ii Steve Harrison & Paul Dourish, "Re-place-ing space: The roles of place and space in collaborative systems", In Proceedings of CSCW 96, ACM Press, Cambridge, MA, pp. 67-76 (1996)

iii Jen Clodius, “Concepts of Space and Place in a Virtual Community”, http://www.dragonmud.org/people/jen/space.html (1994)

iv Ruth M. Rettie, "Presence and Embodiment in Mobile Phone Communication", PsychNology Journal, Volume 3, Number 1, pp. 16 – 34 (2005)

v Anthony Giddens, "The Constitution of Society: Outline of the Theory of Structuration", University of California Press (March, 1986), pp.118-24; 132-5; 164-5; 366-8

vi Tom Boellstorff, Coming of Age in Second Life: An Anthropologist Explores the Virtually Human, Princeton University Press, pp. 91-96 (April 21, 2008)

vii Richard Bartle, "Designing Virtual Worlds", New Riders Games, pp. 473-476 (July, 2003)

viii Brian Winston, “Media, Technology and Society: A History - From the Printing Press to the Superhighway”, Routledge, pp. 3-15 (April 1998)

ix Katie Hafner, Matthew Lyon, “where wizards stay up late: the origins of the internet”, Simon & Schuster, pp. 58-70 (January 21, 1998)

x Robert M. Metcalfe, David R. Boggs, "Ethernet: distributed packet switching for local computer networks", Communications of the ACM archive, Volume 19 , Issue 7, pp. 395-404 (July 1976)

xi Saltzer, J. H., Reed, D. P., Clark. D. D., “End-To-End Arguments in System Design”, ACM ransactions on Computer Systems, vol 2(4) (1984)

xii Tim Berners-Lee, “Information Management: A Proposal”, CERN, http://www.nic.funet.fi/index/FUNET/history/internet/w3c/proposal.html (March 1989)

xiii Tim Berners-Lee, "Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor", HarperOne; 1st edition, p. 37 (September, 1999)

xiv Tim O"Reilly, "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software", http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (September 30, 2005)

xv Tim Berners-Lee, IBM developerWorks Interviews: "Originator of the Web and director of the World Wide Web Consortium talks about where we"ve come, and about the challenges and opportunities ahead", http://www-128.ibm.com/developerworks/podcast/dwi/cm-int082206.txt (recorded July 28, 2006)

xvi Don Tapscott, Anthony D. Williams, "Wikinomics: How Mass Collaboration Changes Everything", Portfolio Hardcover; Expanded edition, pp. 67-71 (April, 2008)

xvii Todd Dagres, David Hornik, “Is Web 2.0 Another Bubble?”, http://online.wsj.com/public/article/SB116679843912957776-fF7CtrdMDTE4n1h5Ju5pv0HKhgM_20071227.html (December 27, 2006)

xviii Tim O"Reilly, "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software", http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, (September, 2005)

xix Tiziana Terranova, "Free labor: producing culture for the digital economy", Social Text, 63, Vol. 18, No. 2, pp. 33-57 (2000)

xx David Harvey, “A Brief History of Neoliberalism”, Oxford University Press, pp. 154, 178-179 (September, 2005)

xxi *Vincent-Antonin Lepinay*, “Economy of the germ: Capital between accumulation and hybrid in Psychologie Economique, Economy and Society”, Volume 36, Issue 4, pp 526 – 548 (November 2007)

xxii Karl Marx, “Capital: Volume 1: A Critique of Political Economy”, Chapter 26, Penguin Classics (May 1992)

xxiii Dmytri Kleiner, Brian Wyrick, “InfoEnclosure 2.0”, Mute Magazine, http://www.metamute.org/en/InfoEnclosure-2.0 (January 2007)

xxiv Yochai Benkler, “The Wealth of Networks: How Social Production Transforms Markets and Freedom”, Yale University Press, pp. 60-63 (May 16, 2006)

xxv Apple Computer Inc., "Hypercard Script Language Guide: The Hypertalk Language", Addison-Wesley (C); 2nd edition (December 1990)

xxvi Michael Stini, Martin Mauve, Frank H.P. Fitzek, "Digital Ownership: From Content Consumers to Owners and Traders," IEEE MultiMedia, vol. 13, no. 4, pp. 1-6 (Oct. 2006)

xxvii Jonathan Schull, Mike Axelrod, Larry Quinsland, "Multichat: Persistent, Text-As-You-Type Messaging in a Web Browser for Fluid Multi-Person Interaction and Collaboration," hicss, p. 60, Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06) Track 3 (2006)

xxviii Lingnau A., Drobnik O., Domel P., "An HTTPbased Infrastructure for Mobile Agents", WWW Journal - 4th Intern . WWW Conf. Proc. , Boston, MA (December 1995)

xxix Fuggetta, A., Picco, G.P., Vigna, G., "Understanding code mobility", Software Engineering, Volume 24, Issue

5, pp. 342 - 361 (May 1998)

xxxMark S. Ackerman, Brian Starr, "Social Activity Indicators: Interface Components for CSCW Systems", Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'95), New York: ACM, 159-168 (1995)

xxxiThomas Erickson, Wendy A. Kellogg, "Social Translucence: Using Minimalist Visualizations
of Social Activity to Support Collective Interaction", Designing Information Spaces: The Social Navigation Approach, Springer; 1 edition, pp. 17-43 (January, 2003)

xxxiiRay Oldenburg, "The Great Good Place: Cafes, Coffee Shops, Bookstores, Bars, Hair Salons, and Other Hangouts at the Heart of a Community", Da Capo Press; 3rd edition (August, 1999)

xxxiiiGreg Wadley, Martin Gibbs, Kevin Hew, Connor Graham, "Computer Supported Cooperative Play, Third Places and Online Videogames", Proceedings of the Thirteenth Australian Conference on Computer Human Interaction, University of Queensland, pp 238-241 (2003)

xxxivRaph Koster, "Theory of Fun for Game Design", Paraglyph; 1 edition (November, 2004)

xxxvCaroline McCaw, "Art and (Second) Life: Over the hills and far away?", digital arts and culture conference, issue 11 (2007)

xxxviErnst Adams, Three Problems for Interactive Storytellers. The Designer's Notebook http://www.gamasutra.com/features/designers_notebook/19991229.htm (1999)

xxxviiMichael Mateas, Andrew Stern, "Façade: An Experiment in Building a Fully-Realized Interactive Drama", In Game Developer's Conference: Game Design Track (2003)

xxxviiiMichael Mateas, "An Oz-Centric Review of Interactive Drama and Believable Agents", AI Today: Recent Trends and Developments, Lecture Notes in Artificial Intelligence no. 1600, Springer-Verlag, Berlin, pp. 297-328 (1999)

xxxixKent Beck, Cynthia Andres, "Extreme Programming Explained: Embrace Change" Addison-Wesley Professional; 2 edition, pp. 100-102 (November, 2004)

xl Mihaly Csikszentmihalyi, "Flow: The Psychology of Optimal Experience", Harper Perennial, pp. 1-22 (February, 1991)

xli Amy Bruckman, "Programming for Fun: MUDs as a Context for Collaborative Learning", ftp://ftp.media.mit.edu/pub/asb/papers/necc94.ps (1994)