



Hypergrid: Architecture and Protocol for Virtual World Interoperability

Massive multiuser online (MMO) environments that simulate large virtual spaces for many users have prompted the desire to create an even larger, highly scalable environment in a federated manner. In a federation of virtual environments, users should be able to visit different environments belonging to different authorities while preserving their identity; they should also be able to join a global, federated social network. The Hypergrid is an architecture and protocol for securely decentralizing multiuser virtual environments. It establishes an open federation of multiuser applications that can exchange user agents and assets and can generally interoperate on several basic services.

Cristina Videira Lopes
University of California, Irvine

Massive multiuser online (MMO) games such as Linden Lab's *Second Life* provide shared virtual spaces, in which thousands of users can interact with one another, with virtual objects, and with artificial intelligence (AI) agents. These environments require considerable server-side infrastructure, controlled in each case by a single organization. Centralized control of virtual worlds enables the development of walled-garden environments with high internal consistency. However, several problems arise from centralization of authority. First, groups of individuals and organizations wanting their own virtual worlds face the binary choice of either operating separate walled gardens (thus making them difficult to share across the

group) or jointly operating one single virtual world for the group (thereby losing control of their own share in that virtual world).

In this article, we present the Hypergrid, an architecture and protocol for securely decentralizing multiuser virtual environments at all scales. The Hypergrid establishes an open federation of multiuser applications that can exchange user agents and assets, and can generally interoperate on several basic services. It supports the teleporting of user agents between worlds in different administrative domains while preserving user identity, as well as the user's 3D visual representation and connections to certain home-world services. We designed and implemented the Hypergrid in the *OpenSimulator*

Multiuser 3D Simulation and Gaming Environments

The field of multiuser 3D simulation and gaming environments is divided into two architectural camps: peer-to-peer (P2P) and client-server.

In P2P systems, the program that the user drives is both the simulator and the user interface. An additional network layer lets several peers join in one logical simulation, and physical simulation of different parts of the scene occurs in the different peers. Examples of P2P multiuser virtual environments include MiMaze,¹ High-Level Architecture standards,² TeCo3D,³ Croquet,^{4,5} Miramar,⁶ and Unity 3D Basic (<http://unity3d.com>). P2P virtual environments are naturally federated, in the sense that each user-driven peer represents exactly one user and has full authority over the user agent's state and over parts of that virtual world.

Massive multiuser virtual worlds follow a client-server architecture. Their internal architectures vary considerably, but they all share one authoritative server side, to which interactive rendering clients connect. Besides the well-known commercial massive multiuser online (MMO) games such as Second Life, Eve Online, and World of Warcraft, examples of publicly documented server-side systems and prototypes include RING,⁷ Project Darkstar (now RedDwarf),⁸ Meru,⁹ and OpenSimulator.

In many ways, client-server architectures do well where P2P architectures do poorly. First, client-server architectures naturally support persistent, sharable virtual environments that exist beyond the user agents that visit them. Second, they provide many more options for scalability because the server side can be fueled with many high-end servers and appropriate bandwidth for acceptable quality of service.

In other ways, client-server architectures do poorly where P2P architectures do well: client-server systems, such as the Web, promote walled-garden environments, some of which end up dominating specific application areas. As people and organizations see value in interconnecting, additional pieces of architecture become necessary to enable those walled gardens to interoperate. This has been happening on the Web for a while. The Hypergrid is another step in that direction.

The closest system to the spirit of the Hypergrid is Open Wonderland (<http://code.google.com/p/openwonderland/wiki/>

OpenWonderland). Open Wonderland is a virtual world client implemented in Java that connects to Darkstar-based virtual world servers,⁸ also written in Java. Like the Hypergrid, Open Wonderland supports a federation of virtual worlds. However, that federation has the following architectural differences. First, the client itself keeps the user agent information; the client is its own authority and keeps that state throughout the session. Second, Open Wonderland relies entirely on the Java programming language for dynamically loading code as the user moves from one world to another. These are interesting variations that simplify the interoperability architecture at the expense of narrowing down the implementation technologies and tightly coupling the servers with the clients.

References

1. L. Gautier and C. Diot, "Design and Evaluation of MiMaze, A Multiplayer Game on the Internet," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems (ICMCS 98)*, IEEE CS Press, 1998, pp. 233–236.
2. F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High-Level Architecture*, Prentice Hall, 1999.
3. M. Mauve, "TeCo3D — A 3D Telecooperation Application Based on VRML and Java," *Proc. Multimedia Computing and Networking (MMCN 99)*, SPIE 3654, Int'l Soc. for Optics and Photonics, 1999, pp. 240–251.
4. D.A. Smith et al., *Croquet User Manual*, tech. report, Open Croquet, 2005; www.opencroquet.org.
5. D.P. Reed, "Designing Croquet's TeaTime: A Real-Time, Temporal Environment for Active Object Cooperation," *Proc. 20th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 05)*, ACM Press, 2005, p. 7.
6. J.D. Miller and C. Pickering, "From One to Many: Transforming Miramar into a Collaboration Space," *Proc. 5th Int'l Conf. Creating, Connecting and Collaborating through Computing*, IEEE CS Press, 2007, pp. 109–116.
7. T.A. Funkhouser, "RING: A Client-Server System for Multiuser Virtual Environments," *Proc. Symp. Interactive 3D Graphics (I3D 95)*, ACM Press, 1995, pp. 85–ff.
8. J. Waldo, "Scaling in Games and Virtual Worlds," *Comm. ACM*, vol. 51, no. 8, 2008, pp. 38–44.
9. D. Horn et al., "Scaling Virtual Worlds with a Physical Metaphor," *IEEE Pervasive Computing*, vol. 8, no. 3, 2009, pp. 50–54.

project (<http://opensimulator.org>). A second, independent implementation is now available in the SimianGrid (<http://code.google.com/p/openmetaverse/wiki/SimianGrid>). The SimianGrid is an alternative back end to OpenSimulator based on PHP and Apache. The Hypergrid is already deployed in several OpenSimulator-based virtual worlds.

Here, we focus on the design of the Hypergrid for worlds based in OpenSimulator that

are accessible via Second Life viewers as the user-driven clients. However, the Hypergrid can also support arbitrary Web-based multiuser applications — a critical capability, as an ever-growing number of Web-based viewers for these virtual worlds are being developed. (The "Multiuser 3D Simulation and Gaming Environments" sidebar describes two main types of 3D simulation and gaming architectures: peer-to-peer (P2P) and client-server.)

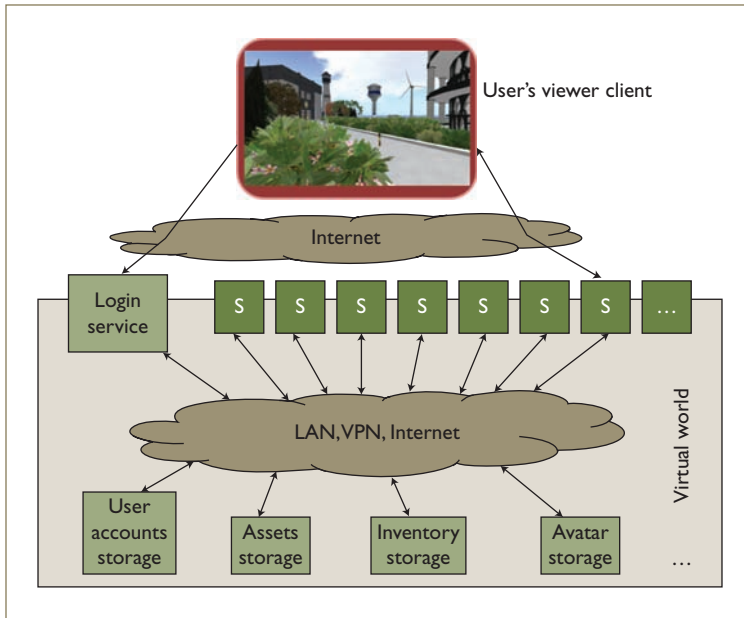


Figure 1. Main architectural components of an OpenSimulator-based virtual world. These worlds can be as small as one single simulator (S) or as large as thousands of simulators that share persistent resources. User-driven clients first authenticate with the world's login service, and then exchange data with specific simulators. (VPN: virtual private network.)

OpenSimulator

The OpenSimulator project began in early 2007 as an open source server side to the Second Life client. A simulator is the basic unit of virtual space containing one or more *regions*, which are 3D spaces of $256 \text{ m} \times 256 \text{ m} \times \infty$. Simulators can be interconnected to form larger, continuous spaces that share persistent resources, known as *grids*. In this article we treat “grid” as synonymous with “virtual world.”

Architectural Style and Components

OpenSimulator worlds follow a client-server architecture similar to that of the Web: user-driven clients merely render the application state, which remains on the server side. Figure 1 depicts the overall client-server architecture of OpenSimulator-based virtual worlds.

Logically, a grid comprises one or more simulator services, a common login service, and a collection of common resources such as assets and inventory. Users access the virtual world through a client (or *virtual world viewer*).

In OpenSimulator's software architecture, the connectors to all resource services are plug-ins. This allows for developing various concrete middleware grid services to support the simulators.

Popular configurations include small grids with one or a few simulator servers, all directly connected to a MySQL server on the same LAN, and grids with multiple simulators connected to Apache-server-based resource services over the Internet.

Protocols

Here, we describe the major protocols in OpenSimulator between the viewer, the login service, and the simulator services when both the login and simulators are all within the same administrative trust domain.

So that the viewer software could be reused without changing it, these protocols were heavily influenced by how Second Life is engineered. Although the protocols described here target specific commercial virtual worlds, they're important for three reasons. First, they embody a profound generalization of the well-known user agent concept on the Web. Second, they show how to manage user agent transfers in a distributed system. Third, they're the basis for the Hypergrid protocols described later, which simply add security safeguards for when the interacting components belong to different administrative trust domains.

Login. The login protocol involves the user's viewer client, the login service, and a simulator:

1. The viewer contacts the login service on an HTTP-based (or HTTPS) connection, sending the user's credentials (username and password) and desired virtual place (simulator).
2. The login service verifies the user's credentials. If they're valid, the login process generates a pair of session IDs. (This is a minor detail of Linden Lab's viewer; in other applications, only one session ID would be necessary.)
3. The login service creates a user agent, which includes the session IDs and information about the user's 3D representation.
4. The login service logs the user's session in the grid using a persistent *presence* resource.
5. The login service sends this user agent to the simulator that runs the desired virtual place.
6. The simulator verifies the user's presence with the given session IDs. If verification is successful, the simulator stores the user agent and prepares for initial viewer contact.
7. The login service sends the login reply to the viewer, including the session IDs and the desired simulator's IP end point.

8. The viewer contacts the simulator. The simulator then verifies the existence of the valid user agent, and the simulation proceeds from there on.

Once the user logs in, he or she can access and interact with the resources of the virtual world. The user can also move around to different parts of that virtual world through a process called teleporting.

Intragrid agent transfer. The intragrid agent transfer (*teleport*) protocol involves the viewer, the current simulator, and the target simulator (that is, the simulator to which the user wants to go next). We assume all server-side components are in the same administrative trust domain:

1. The viewer notifies the current simulator about the desired virtual place where the user wants to go next.
2. The current simulator sends a copy of the user agent to the target simulator running the desired virtual place. It also sends an opaque callback address for later use.
3. The target simulator stores the user agent and prepares for initial viewer contact. Preparation includes creating authorization tokens for the user agent to use while visiting that simulator.
4. The current simulator sends information to the viewer about the target simulator, including the target simulator's IP end point. Although the mechanism is quite different, the nature of this step is similar to HTTP's redirect return code.
5. The viewer contacts the target simulator, which verifies the existence of a valid presence for the user.
6. The target simulator invokes the callback to the original simulator, signaling that the viewer has made contact.

The current simulator discards its copy of the user agent, and the hand-off is complete.

User Agents

The term "user agent" is most notably used to identify clients that access Web servers. For example, the HTTP protocol includes a `User-Agent` request header that identifies the software used to issue the Web request: `Mozilla/5.0 (Windows; U; WindowsNT6.1; en-US; rv:1.9.2.13)`

`Gecko/20101203 Firefox/3.6.13`. The Session Initiation Protocol (SIP) uses the term "user agent" to denote the user-driven client software's Internet end points.¹

Here, we've generalized the concept of user agent to include information not only about the software that users drive and their Internet end points but also about the users themselves: the service end points used by a particular user, identifiers of assets related to that user's 3D representation, authorization tokens, and so on.

Because these environments provide simulation of virtual spaces, of which the user's representation is a part, portions of the user's state could change as that user visits different simulators. For example, the user could carry a script that stores the names of all users that this script encounters; such a script is part of the user agent, and it's executed by each simulator that the user visits. That script's state (users' names) is accumulated as the user moves around and the script migrates from one simulator to another. Transfers of user agents between simulators ensure the preservation of the server-side state related to the respective users throughout the session's duration.

The Hypergrid

The Hypergrid's goal is to provide a relatively seamless user experience as users visit different virtual environments, while ensuring the integrity of all parts. The Hypergrid achieves seamlessness through a single sign-on (SSO) mechanism that preserves user identity throughout the session's duration, and by making certain user services available to the virtual worlds that the user visits. The result is an open but secure federation of virtual environments.

Architectural Components

The Hypergrid consists of a collection of Web services provided by the virtual environments to the rest of the world in addition to their internal services. Figure 2 illustrates the Hypergrid's architecture.

The gatekeeper service. Users can enter a virtual world via two main entry points: the regular login service, which requires a local account in the virtual world, and the gatekeeper service, which is the entry point for users with accounts elsewhere. All user agents from users of other

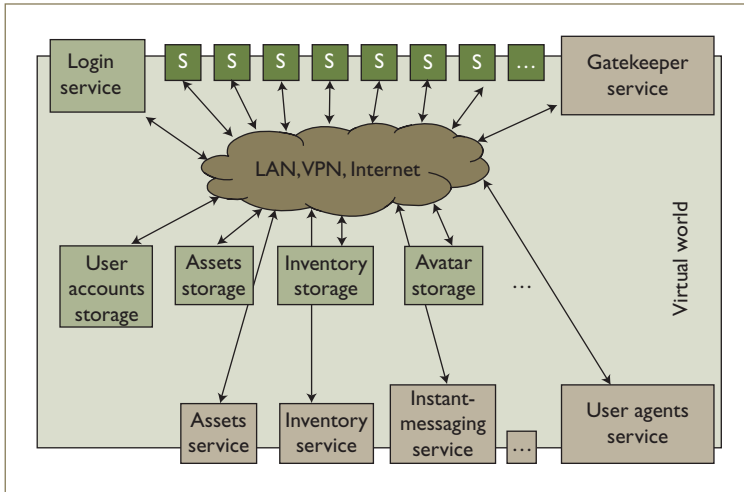


Figure 2. Main architectural components of the Hypergrid, including additional Web services that virtual worlds can provide to become part of the open Hypergrid Federation. The green boxes are internal to each grid, whereas the tan boxes are the Hypergrid's components.

worlds enter through the gatekeeper; one of the gatekeeper's responsibilities is to authenticate such user agents. Attempts at sending user agents directly to the world's simulators will fail because the simulators expect the gatekeeper to have authenticated those agents. The authentication procedure is the basis for the SSO mechanism, which is the core of the Hypergrid.

Additionally, the gatekeeper can filter user agents on the basis of access control rules and policy decisions regarding incoming data.

The user agents service. As explained earlier, the virtual environments considered here use a client-server architecture in which the client simply renders information kept by the server side. This has some important consequences for user identity, and how it is managed throughout the sessions.

The most important consequence is that all authority resides not on user-driven components but on servers, and this includes information pertaining to user identity: users acquire identities by logging in to identity services on the Internet. Those identity services could be part of the virtual worlds, or they could be stand-alone identity services. OpenSimulator worlds include user accounts, and thus can be identity providers. We call the system with which a user acquires his or her identity the *home world* for that user.

Once users acquire an identity with their home world via a login procedure, they can visit other worlds. A visit to another world requires sending user agent information to the target world – more specifically, to its gatekeeper. For security reasons, the only authority that can send user agents to other virtual worlds is the home world's user agents service.

To illustrate the need for this component, consider the following scenario. User 1 is visiting some foreign world Y and wishes to move to another world Z. World Y has a copy of the user agent, so it can simply send it directly to world Z. However, such a direct exchange could compromise the user agent's integrity. A rogue world Y could add malicious data to the user agent, undermining possible trust relations between the user's home world and world Z.

To avoid such vulnerabilities, the Hypergrid establishes the existence of the user agents service – the authoritative driver of all user agents pertaining to each world's local users. One of this service's main responsibilities is to regenerate valid user agents every time users move between worlds. A second main responsibility is to keep track of all user agents and their locations.

Additional user support services. Besides securely preserving user identity across virtual environments, the Hypergrid also provides federated access to certain services that support a better user experience (see Figure 2). We describe one of these services here.

In these virtual worlds' rich visual environments, the user's 3D representation (*avatar*) is important, and its preservation across worlds might be desirable. There are several different ways to represent the avatar, but it always includes assets stored in the user's home world. As such, preserving the avatar upon agent transfers requires providing access to those assets by the world that the user is visiting.

Serving assets on the Hypergrid isn't the same as serving assets within one world, because asset exchanges between worlds involve different administrative and trust domains, and hence require additional filters and safeguards. Hypergrid asset servers should perform authorization of requests and could perform metadata adjustments. For example, the current implementation of the Hypergrid asset service in OpenSimulator adds universal resource identifiers to create

information pertaining to exported assets. Thus, if John Smith created an asset in world A, the exported information would include a URL for user John Smith's profile.

Besides assets, the Hypergrid enables the open-ended collection of user support services related to the user's resources, social network, and communication. Social networking in the Hypergrid is a global, federated facility: users can have friends in other grids and can communicate with them. Therefore, each world can expose services that support those global social connections in a manner that shares similarities with Diaspora (<https://joindiaspora.com>).

Single Sign-On

The Hypergrid SSO mechanism lets users log in only once to their home world and securely use their identities to visit other worlds in the federation without being prompted for credentials or confirmation. The Hypergrid SSO mechanism is based on the protocols described earlier but extends them to deal with components in different administrative trust domains. In these protocols, parts in bold denote the new protocol elements that the Hypergrid has added.

SSO login. For the sake of simplicity, the protocol explained here assumes users always log into their home world. In OpenSimulator, the Hypergrid login procedure is more general, letting users log in directly to any grid. The simplification made here doesn't change in any way the main security safeguards on agent transfers that the Hypergrid adds.

Here, the user agents service, the login service, and the initial simulator are all within the same administrative trust domain. The sequence of events is essentially identical to the one described earlier in the "Login" section; the main difference is the collection and storage of additional information for the user agent: that is, in step 3, the login service creates a user agent comprising the session IDs, information about the user's 3D representation, and a collection of URLs representing the user's services, including the user agents service; and, in the latter part of step 4, the login service sends the user agent's information to the user agents service; this information includes the user's client IP address, as reported by the initial login request's TCP stack.

Hypergrid agent transfers. Here, we describe a teleport protocol involving the viewer, the current simulator, the user agents service of the user, the grid's gatekeeper service where the user wants to go next, and the target simulator. There are three trust domains: the viewer and the user agents service, the gatekeeper and the target simulator in its grid, and the current simulator. This protocol is based on the teleport protocol described in the "Intragrid agent transfer" section:

1. (same, except the target virtual place is in another grid and is identified by that grid's gatekeeper address)
2. The current simulator sends a snapshot of the user agent to the user agents service of that user, along with information about where the user wants to go next. It also sends an opaque callback address for later use.
- 2.1 The gatekeeper and user agents service interactions ensue. The security precautions regarding these interactions are as follows. First, the user agents service generates a unique service key for the desired grid, adds it to the user agent data, and launches the agent at the desired location's gatekeeper service. The unique service key consists of the destination's gatekeeper URL, to which a unique random token is added (for instance, <http://hg.osgrid.org/?cap=9876543210>). Second, the user agents service updates the user's traveling data with the new destination and service key (for example, a user agent with session ID 1, IP address 70.45.12.64, going to hg.osgrid.org, with service key <http://hg.osgrid.org/?cap=9876543210>). Third, the user agents service might filter data from the user agent it received from the departing simulator, and then launch the agent at the destination gatekeeper. The destination gatekeeper service performs verification against fake agents (impersonations). The data used for this is the provided service token and the reported user agents service URL.
- 2.2 If all verifications succeed, the gatekeeper logs the user session in its grid using a persistent presence resource, and launches the user agent at the desired local simulator.
3. (same)
4. (same)
5. The viewer contacts the target simulator, which verifies the existence of a valid presence

for the user. Additionally, the target simulator contacts the user's reported user agents service URL for verification of the user's client IP address. This prevents other kinds of impersonations.

Finally, steps 6 and 7 are the same as in the "Intragrid agent transfer" section.

Security

Virtual worlds, especially those built on client-server architectures, operate within broad margins of mutual trust. To a large extent, this trust is determined by current technological limitations about what can be protected. Nevertheless, neither users nor virtual worlds should be allowed to go beyond those margins of trust. Hypergrid security must ensure the integrity, availability, and confidentiality of resources intended to be integral, available, and confidential. Two particularly important types of resources that need protection are the user agents themselves and the virtual worlds' assets.

User Agent Integrity

Whatever abuse might occur in one world should be limited to that world only, and shouldn't compromise the integrity of the user's agents sent to other virtual worlds. This is the main purpose of the user agents service in the Hypergrid architecture.

Hypergrid security relies primarily on reliable user agent authentication throughout the federation of virtual worlds. If impersonations were to occur, the Hypergrid wouldn't function. Impersonations could occur if rogue virtual worlds visited by users could send rogue user agents to other worlds and then control those user agents as if they were representing the users.

Fortunately, the verifications that the gatekeeper and the target simulator make against the user agents service ensure that impersonations won't occur.

Confidentiality of Assets

Whatever abuse a user might perform on the world's data should be limited to the exposed data only, and shouldn't compromise the confidentiality of assets that aren't exposed.

Hypergrid-facing asset servers open another door to the world's assets that must be carefully secured. Access to assets via Hypergrid asset

servers should be granted only to authorized asset consumers and not to anonymous clients on the Internet. The only authorized consumers of those resources are the worlds that the home users visit at any point in time. For example, if user U of home world H visits virtual world Z, then Z might need to download U's avatar assets from H to construct an accurate 3D representation of the user. Similarly, if, while visiting Z, U gives an item to another user, Z must broker that transfer, which requires access to the item's assets stored in H.

We're currently adding this authorization mechanism to OpenSimulator's Hypergrid-facing asset server. It works in the following manner. According to the protocol explained in the "Hypergrid agent transfers" section, every time the user agents service sends a user agent to a new virtual world, it issues a unique service key. The target world uses that key as an authorization token to access resources of the user's home world. Subsequent requests to the resources of the user's home world must include that key. Unauthorized requests will be denied access to the asset resources.

The Hypergrid asset servers can establish more restrictive policies on top of this authorization mechanism. For example, they could deny access to certain types of assets.

Although we designed and implemented the Hypergrid for OpenSimulator-based virtual worlds and the dedicated clients currently used to interact with those worlds, we ultimately would like it to be an architecture and protocol for federating virtual environments on the Web itself. Consider the architecture in Figure 2. If we substitute Web servers for the simulators (S), the components in green represent the server-side architecture of many multiuser Web applications. As such, it's straightforward to add the (tan) Hypergrid components, making those applications ready to be federated.

The reason behind this goal is simple: the Web has the critical mass of users, and interest has increased in adding 3D immersion to Web applications. Promising emerging technologies for adding interactive 3D elements to regular Web applications include Flash; Unity 3D; the combination of JavaScript, WebGL, and WebSockets; and server-side streaming. The future

of virtual worlds will likely include various viewers that run on Web browsers – not only the popular 2.5D Flash applications, but well beyond. Therefore, the Hypergrid takes the Web design principles and the server-side of Web applications as design invariants,² carefully staying away from optimizations and simplifications that would compromise applicability to the Web in general. □

Acknowledgments

Melanie Thielker contributed invaluable input to the Hypergrid's design, especially its security. The OpenSimulator community has made the Hypergrid a reality; their feedback and enthusiasm is what matured the Hypergrid from an experiment to a viable interoperability architecture. This work is partially supported by NSF grant IIS-0808783.

References

1. J. Rosenberg et al., "SIP: Session Initiation Protocol," IETF Internet draft, work in progress, June 2002.

2. R.T. Fielding and R.N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Trans. Internet Tech.*, vol. 2, no. 2, 2002, pp. 115–150.

Cristina Videira Lopes is an associate professor with the Department of Informatics in the School of Information and Computer Sciences at the University of California, Irvine. Her research interests include information retrieval for aspect-oriented programming; software engineering for large-scale systems; ubiquitous computing, including lightweight software acoustic modems that can be played and decoded in small portable devices such as cell phones; and massive multiuser online (MMO) virtual worlds and their applications beyond gaming. She's a core contributor to the OpenSimulator project, a server-side virtual world platform. Lopes has a PhD from Northeastern University. She's a senior member of IEEE.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

NEW

Transactions on Computers {EssentialSets} Available:

ESSENTIAL INDUSTRIAL IMPLEMENTATIONS OF FLOATING-POINT UNITS DURING THE LAST DECADE:

VOLUMES 1 & 2

Edited by TC AE Elisardo Antelo, these EssentialSets surveys the industrial design of floating-point units during the last decade. This EssentialSet is broken into two volumes, sold separately.

PDF edition • \$15 each (\$9 members)

Order Online: computer.org/store.

IEEE **computer society** **CSPress**

This article was featured in

computing **now**

ACCESS | DISCOVER | ENGAGE

For access to more content from the IEEE Computer Society,
see computingnow.computer.org.



IEEE

IEEE  **computer society**

Top articles, podcasts, and more.



computingnow.computer.org