# Networking with knobs and knats?
# Towards ubiquitous computing for artists

Jeff Burke, Eitan Mendelowitz, Joseph Kim, Rex Lorenzo
HyperMedia Studio, School of Theater, Film and Television†
University of California, Los Angeles
{jeff, eitan, joseph, rex}@hypermedia.ucla.edu

**Abstract**

Ubiquitous computing technology enables new forms of creative expression, but the exploration of ubicomp in the arts and entertainment communities requires unique design considerations. Specifically, this paper suggests that the *myth of interface independence* and the assumption that *authoring comes for free* are particularly problematic for artistic applications, and summarizes guidelines and implementation details for a middleware-level control system and scripting language for artists under development at the UCLA HyperMedia Studio.

**Ubiquitous computing as a catalyst for new artistic expression**

Ubiquitous computing technologies provide a fascinating new arena for creative expression that extends beyond more immersive gaming environments or new methods of accessing traditional content. One of ubicomp's most intriguing qualities is how it can create connections across modal and geographic boundaries. The technologies of ubicomp allow artists to create *systems of relationships* between sensed events in the physical world, digitally controlled elements of an experience, and purely virtual components.

The UCLA HyperMedia Studio investigates the impact of new technologies on traditional production in theater, film, and television, as well as the new forms of expression that they inspire and enable. [7] For the past several years, we have developed installation and performance projects with students and faculty from film and television, theater, electrical engineering, and computer science. Based on this experience, and after incorporating the results of this work into several courses, we have concluded that one of the most significant roadblocks to the exploration of ubiquitous computing in the arts is the lack of appropriate middleware.

Artists exploring digital technology find negotiating the multitude of protocols, conventions, and metaphors for the physical world daunting. By impeding experimentation, this prevents artists from tackling the more relevant problem of expressing creative desires algorithmically in order to implement them on digital devices. The tools' basic affordances remain largely inaccessible, in contrast to those of traditional materials: paint, paper, speech, the body, and for some, even Adobe Photoshop. For the author without an intuitive feel for a ubicomp device or system's intrinsic capabilities, it remains a provocative novelty and not a useful tool.

While high-level multimedia programming environments like Macromedia Director, Cycling 74's Max/MSP, and even Microsoft Visual Basic, are accessible to technically savvy artists, their inconsistent interfaces to the external world make them difficult environments for ubicomp experimentation. We would like to enable artists to explore ubicomp with at least the ease with which they use existing screen-based tools.

**Art as a laboratory for ubiquitous computing**

At the HyperMedia Studio, we use creative projects to drive technological development. One current multi-year research collaboration, *The Iliad Project*, is the "driver application" for a control system that tries to answer the challenge described above. The project aims to create the architecture for a new performance work that draws its context dynamically from the audience and the surrounding city, wherever it is performed. For this piece, we must integrate radio-frequency identification (RFID) systems, a database of audience information, media control and routing, stock and live media, as well as large-scale theatrical lighting and sound control.

† This work is funded in part by the National Science Foundation grant IIS-0207869.

Participants start their experience through a survey process on a website, which populates a database used throughout the performance process. In the venue itself, which contains gallery and performance spaces, RFID antennas read tags in the participants' tickets, linking their physical presence to the database records. Media and text are adapted based on the particular audience members in attendance, in ways that serve the purposes of the story. As the experience progresses, the control system updates the database with sensor data and images collected from the performance, which are later re-incorporated into the piece.

The control software being designed to support our creative efforts, including *The Iliad Project,* poses challenges that parallel many current topics in ubicomp research, but with an atypical set of conditions. In some cases, difficult requirements present in home/office scenarios are relaxed, while in other cases, additional robustness is necessary. Furthermore, in our application domain, aesthetic needs motivate the use of technology. This leads to healthy and necessary questioning of the very assumptions driving technological choices. When successful, conscientious artistic production results in a *working environment* that has evolved out of an effort to understand critically the role of each component and its effect on the viewer/participant.

With these two ideas in mind, that artistic projects provide a useful and unusual testbed for ubiquitous computing, and that ubicomp inspires and enables new artistic expression, we move forward to describe our considerations and design goals in developing middleware for artistic and entertainment applications.

**Considerations and design goals for ubicomp in art and entertainment**

In our exploration of ubicomp devices, software, and techniques, we must confront and ultimately reject two popular ideas, ones helpful for task-based application development, but which do not apply to systems that interconnect devices for evocative responsive environments.

The first is the *myth of interface independence*. While the transparency of interfaces to the end user is an exciting possibility and a central idea of ubicomp, it is a mistake to confuse transparency with irrelevance. Weiser's seminal paper on ubiquitous computing is exciting because it describes the "disappearing computer," not the irrelevant interface. [8] He states, "Whenever people learn something sufficiently well, they cease to be aware of it." Put another way, the affordances of the device, object, or space are so clear or so well learned that they can be used without a conscious parsing of the interface. Today, this is often inaccurately translated to mean "device neutrality," and used to require that device specifics be hidden from application developers. This allows applications to run across many platforms, but ignores the fact that the particular affordances of an interface or environment are always part of the user experience and are crucial to their ability to "disappear." This is (arguably) unnecessary for the Personal Information Manager (PIM), but these nuances are central to art and entertainment experiences. The tendency towards device abstraction, though valuable, often leads to "lowest-common-denominator interfaces," like the basic Java UI libraries. [2] It also seems perplexingly problematic in ubicomp, where the distinctions between interface and content are often less clear than in traditional delivery systems.

In fact, artists are accustomed to precisely and meticulously arranging details in their work: consider the hundreds of individual lights used in a performance on Broadway. In some cases, even the response curves of individual dimmer channels are modified to achieve a particular type of transition. When the computer disappears, and lighting control of a large entertainment venue exists as part of a ubiquitous network, no matter how smart the interface is, it would be an odd designer who settles for only being able to say "lights should be on in this scene," instead of being able to directly adjust the intensity of the second light on the third pipe over the stage. In this example, there are two levels of interface in the same application: the task-oriented input interface for the designer-as-experience-author, which might be device-agnostic, and the interface necessary to address directly individual devices in a very specific manner.

Can there be middleware that consistently and simply exposes the specific functionality of devices on a network to artists, even if it is part of a larger, task-oriented infrastructure with other, more abstract interfaces? Or, will this type of control remain in the domain of specialized scripting languages and inflexible, hermetically sealed control systems interconnected by proprietary interfaces?

The second idea that we challenge is an implicit one, that *authoring comes for free*. Both the vision and technical specifics of modern ubicomp research rarely treat the challenge of creating authoring tools as the rich and interesting problem that it seems to be. In many cases, including augmented reality (AR), there are fantastic toolkits for experimentation, but they tend to be domain-specific, at least when viewed in light of a broad ubiquitous computing vision. Our concern is with how artists can collaborate to author experiences in instrumented environments. For the foreseeable future, these collaborators must have some technical expertise among them, but it should be focused on understanding technological affordances, and developing the relationships and media of an experience, rather than on the details of device interfaces or networking. At this time, though, it seems that the choices of authoring systems either isolate the author completely from devices or remain too low-level for non-programmers. To take the previous example further, what if the lighting designer wanted to author an interactive relationship between the intensity of a set of lights and the position or movement of an actor, using pervasive computing devices in the soundstage or theater? [4] How could a non-programmer most effectively configure this capability of a ubicomp environment?

**Kolo: An arena for experimentation**

To begin to answer some of these questions, we are developing "Kolo," a TCP/IP-based control system and associated scripting language to interconnect sensors, actuators, virtual components, and other digitally controlled elements of an experience. Below, we describe more specific requirements, design goals, and implementation details as an example of ubicomp software for art and entertainment.

*Requirements*

(1) We need a system that supports our real-world application by the fixed deadline of an opening night. We cannot rely on future technologies, but can try to anticipate how we would incorporate them. Artistic work provides a highly controlled but demanding application domain, which is an already recognized need:

> "[C]reating reliable and robust systems that support some activity on a continuous basis is difficult. Consequently, a good portion of reported ubicomp applications work remains at the level of demonstrational prototypes that are not designed to be robust... [T]he requirement is for real use of a system, deployed in an authentic setting…By pushing on the deployment of more living laboratories for ubicomp research, the science and practice of HCI evaluation will mature." [1]

(2) For now, we consider systems that support heavily designed environments: installation artworks, performances, media shoots, and so on. Though we need them to be flexible and resilient, our delivery requirements are less stringent than many broad ubicomp visions, like MIT's Oxygen project. [5] If our control system cannot provide services while a user is deep sea diving off the coast of Australia, or in a car on Los Angeles' 405 freeway, we will not be too disappointed. However, we are interested in the capability of networking remote spaces and exploring the use of embedded devices.

(3) Our interest in practical implementation divides equally between traditional and non-traditional input and output devices. In fact, in keeping with the ubicomp vision, we find alternatives to the mouse, keyboard, and screen most intriguing. These include localization systems, speech recognition, multi-channel sound input and output, lighting control, motors and actuators, and other more unusual devices.

*Design goals*

(1) The high-level object model, scripting language for authors, and the device interface API for developers, must be simple, consistent, and expressive. Every component on the network should be addressable through a common interface, trading off complexity for consistency.

(2) The underlying control system must be stable, extensible, and as consistent with the high-level metaphors of the system as possible. Its installation and bootstrap process must be straightforward.

(3) There are "soft real-time" performance requirements. The perception of the environment's responsiveness by the audience/participants, authors, directors, designers, and/or performers is crucial.

(4) In media-making and performance, grouping and synchronization (*e.g.*, the simultaneous fade-out of lighting, sound, and projection) are necessary. To support this, the system should have the ability to simultaneously control groups of unlike objects that have common attributes.

*Object model*

The *Kolo network object model* (*knom*) provides both developers and artists a consistent method for representing real-world objects and information in the system. By convention, network objects (*knobs*), represent all physical and virtual entities. All knobs are organized into a tree structure and can be referenced by a pathname starting from the root. A knob has only one parent but can have any number of network objects as children, including ones that can have a value—network attributes (*knats).* For example, a lighting controller might expose a light to the network as a knob called "light" with a child knat called "intensity." Similarly, a position tracking system may represent an actor's location with a knob called "achilles" and child attributes "xPosition" and "yPosition." The knob hierarchy can be remapped by the author in scripts.

Any knob (and therefore any knat) may subscribe to a knat's value and take actions as the value changes. A knat that subscribes to one or more values and changes its own value in response is called a *relationship*. Relationships can range from simple arithmetic expressions (e.g., a sum calculation) to any complex function that a computer can implement. An author may choose to allow only one relationship to change a particular knat's value (and often, then, to cause a corresponding side effect in the environment). This privileged relationship is called an *arbitrator*. Competing knobs attempting to set an arbitrated knat's value are redirected to the arbitrator, which, for example, might calculate the knat's value as the mean of all incoming values.

The *knom* is simple, flexible, and allows for the representation of most real-world information as well as for complex interconnections between network objects. It does not define the nature of relationships' mappings; this is handled by individual knats implemented in Java or in the scripting language described below.

*Implementation*

The Kolo API is implemented in "pure" Java, allowing knobs to be placed on the network from any platform that supports Java and TCP/IP. The API facilitates creation of knobs, knats, and the relationships between them. Additionally, it defines standard ways of allowing developers to add new devices to the system. To add a new device, the developer needs only to extend a few classes that deal with knob and/or knat creation. If a knat is to provide an interface to a sensor or actuator, the developer must define the external source of its value and/or the action to take when its value is changed. Communication between objects is via a simple UDP-based protocol for data, and the Simple Object Access Protocol (SOAP) over TCP for control. The latter will facilitate use of the web service description language (WSDL) in the future to describe knob and knat functionality. Currently, we implement limited device discovery, rely on network hardware for security, and implement fault-tolerance on an application-specific basis.

*Scripting language*

While we expect developers to write device interfaces, "applications" will be constructed primarily by technically savvy artists. As an initial step towards other authoring methods, we have developed a scripting language with a small consistent vocabulary that can be learned quickly, based on previous work by one of the authors in scripting languages for artists. [6] The language allows control over Kolo's core functionality - creation and destruction of knobs, knats, subscriptions, relationships, and arbitrators. It is embedded in Scheme and run using a Java-based Scheme interpreter. This provides many capabilities including run-time creation and modification of scripts, supporting a flexible scripting process that compliments the artistic process. Because of our emphasis on soft real-time applications, the scripting language's implementation makes heavy use of Scheme macros, compiling the high-level scripts to lower-level Scheme and Java byte code in order to mitigate the run-time performance issues usually associated with interpreted systems.

## Conclusion

"Today, pervasive computing is more art than science."
– Banavar et. al, IBM TJ Watson Research Center. [2]

The possible uses of ubiquitous computing technology for creative expression are as broad as the technical challenges of the field itself. We feel that these applications provide a unique opportunity for highly controlled, real-world tests of ubicomp devices, software, and techniques, and that the technology allows artists to incorporate into their work systems that are responsive to both viewer-participants and external conditions. A lack of suitable middleware and authoring tools limits experimentation in this arena, which we are addressing with small steps in our own work through the creation of Kolo. As technology disappears into the environment, we hope that, oddly enough, its useful specifics can be made more accessible to artists.

## Acknowledgements

The authors thank Fabian Wagmister, director of the HyperMedia Studio, for shaping the vision of what software supporting artistic expression should really do, and Jared Stein, writer and co-architect of The Iliad Project, for countless ideas and contributions that inspired many technological decisions.

## References

[1] Abowd, G. D., Mynatt, E. D., Rodden, T. "The Human Experience," IEEE Pervasive Computing Jan.-March, 2002, 48-57.
[2] Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., and Zukowski, D. "Challenges: An Application Model for Pervasive Computing," Proc. of the Sixth Annual Intl. Conf. on Mobile Computing and Networks (Mobicom 2000), Boston, MA, , 2000, 266-274.
[3] Burke, J. A., Shive, A., and Wagmister, F. "Macbett: A Case Study of Performance & Technology for Dynamic Theater Spaces." IEEE Multimedia Tech. and Applications Conf., Irvine, California: November 7-9, 2001.
[4] Burke, J. A. "Dynamic performance spaces for theater production." Theatre Design & Technology (U.S. Institute of Theatre Technology), 2002, 38(1):26-35.
[5] Dertouzos, M. L. "The future of computing," Scientific American, August 1999.
[6] Mendelowitz, E. "The Emergence Engine: A behavior based agent development environment for artists," Proc. Twelfth Conf. on Innovative Applications of Artificial Intelligence (IAAI), 2000, 973-978.
[7] Wagmister, F. and Burke, J. A. "Networked multi-sensory experiences: Beyond browsers on the web and in the museum," Museums and the Web Conference, Boston, MA, April 2002, Pittsburgh, PA: Archive & Museum Informatics, 205-216. http://www.archimuse.com/mw2002/papers/wagmister/wagmister.html
[8] Weiser, M. "The Computer for the 21st Century." Scientific American, 1991, 265(3):94-104.