A DISTRIBUTED CONTROL SYSTEM AND SCRIPTING LANGUAGE FOR "INTERACTIVITY" IN LIVE PERFORMANCE

Eitan Mendelowitz, Jeff Burke
HyperMedia Studio
School of Theater, Film and Television
University of California, Los Angeles USA

Abstract: This paper describes the architecture of a new control system and associated scripting language currently under development in a collaboration between computer scientists, engineers, and artists. The system is designed to facilitate the creation of real-time relationships between people and media elements in live performance and installation artworks. It draws on the experience of the UCLA HyperMedia Studio in producing media-rich artistic works and suggests an approach also useful for prototyping "interactive" and "smart" spaces for entertainment and education.

Key words: Live performance, interactivity, theater, scripting, control, smart rooms, intelligent environments.

1. Introduction

1.1 New Technology and Theater

The next phase of the information age, one that will drive entertainment for years to come, will not unfold on the computer screen or involve the mouse and keyboard. It will happen in public and private spaces without visible computers, under the skin and at the interface between machines and the body. While industry refines existing technology, research within and outside of the university explores artificial cognition, nanotechnology, smart objects and environments, and human/machine interfaces that replace the

R. Nakatsu et al. (eds.), *Entertainment Computing*
© Springer Science+Business Media New York 2003

button click with the spoken word, a gesture, or movement of the body. This new technology is being created to be a part of our everyday life, away from our desks, away from our computers, and away from our televisions. For better or worse, it will be pervasive, ubiquitous and active in every object and element of our lives, including our new entertainment experiences.

The use of this new technology in the performing and media arts is a research area shared by many U.S. and international universities. The Intelligent Stage Project at Arizona State University [6] and research in interactive environments at MIT's Media Lab [11-14] are two examples. At UCLA's HyperMedia Studio in the School of Theater, Film and Television, we developed "interactive systems" for a recent subscription-series production of Ionesco's *Macbett* that enabled control of theatrical lighting and sound based on performer movement and position. [3]

Macbett connected a position tracking system to the extensive lighting and sound control infrastructure of a modern theater, enabling designers to create direct real-time relationships between onstage action and the many parameters of digitally controlled design elements. Once the system was in place, sophisticated and unconventional relationships between performers and the environment became possible. For example, sound could be intensified based on the speed of movement of an actor during a certain section or lighting controlled by the distance from one tracked performer to another, regardless of their position on the stage. Similar relationships can be considered for audience interaction in single- and multi-person experiences.

One of the unique qualities of the digital arena is the ease with which connections can be made between components. Because the components (or their controllers) share a common digital representation of information, they are ultimately separated only by conventions and protocols. Where these can be bridged, digital technology allows artists to set up systems of relationships between the physical world (as it can be measured by technology), digitally controlled elements of the experience, and purely "virtual" components.

For many experimenting with "interactive experiences," the most time-consuming step is the creation of those initial bridges across conventions and protocols. Especially for artists, experimentation with connection-making is most limited by the software available, not the sensors for input or display technologies for output. Relationships between viewer-participant or performer action and interactive works are enabled by software systems that connect or "glue together" different components of the interactive system. Our past works have used custom software developed in a variety of programming languages and authoring environments: Macromedia Director, Microsoft Visual Basic, Cycling 74's Max/MSP, and C/C++.

Though the control applications for each work were fairly flexible, each used a slightly different internal approach and presented a different configuration interface. To facilitate future works and encourage experimentation, we are developing a common control system and scripting language for our work at the HyperMedia Studio. The two are designed to provide a consistent method for non-programmers to script interactive relationships across media boundaries, allowing databases to affect stage lighting, sensors to control video playback, participant proximity to vary sound playback, and so on. We believe the approach will have applications outside of performance, in single or multi-user interactive spaces for education and entertainment.

2. Hypermedia Control System

As mentioned above, the new control system is designed to provide a consistent method for non-programmers to script interactive relationships across media boundaries. To that end, the system must: (1) Be flexible and expandable enough to support the many different hardware interfaces currently used in live performance and to aid in the integration of non-traditional technologies; (2) allow for run-time control and modification of scripts, so as not to hinder the dynamic nature of theatrical rehearsals and live performance; (3) be usable by technically minded non-programmers.

2.1 Structure

2.1.1 Objects and Attributes

The HyperMedia Control System exists as a collection of objects distributed over a TCP/IP network. Objects on the network may correspond to physical objects (*e.g.*, lights, actors), physical quantities (*e.g.*, light intensity, actor positions), or stored data (*e.g.*, databases, web pages). The network may also include organisational objects, used by the control system for accessing and grouping of objects, and control objects, used by the system to modify other objects.

All objects on the HyperMedia control network have a name and parent, and may have an arbitrary number of children. Some objects have a value and are referred to as attributes. Attributes provide a uniform read/write interface to data in the HyperMedia system. For example, a light object has an intensity attribute as its child that both describes and determines the

light's brightness. Any object may read or subscribe to an attribute's value. Attribute values are either determined by the application implementing the object itself (usually the case for a sensor) or by relationships with other attributes (usually the case for an output like the intensity of a light).

We are currently developing libraries to support the creation of objects and the subscription to attribute values. Such libraries will allow the rapid addition of familiar theatrical components (lighting, sound, databases, video, tracking). Additionally, the use of distributed objects and the abstracted network protocols will aid the creation of advanced objects for intelligence sensor fusion (using, for example, Bayesian networks, fuzzy logic systems, and Kalman filters) and graphical user interfaces for the runtime creation, monitoring and control of objects by human operators.

2.1.2 Organisational Objects: Root, the Registry, and Groups

As stated above, all objects on the HyperMedia control network must have a parent. This requirement (by definition) organizes all objects into a tree structure and requires the presence of a special root object exempted from the requirement to have a parent. All objects on the HyperMedia network may be accessed through a unique path from root.

One of root's children is the registry, which keeps a mapping from an object's unique path to its IP address, port, and numeric identifier. The use of both root and registry allows an object to be accessed either by its unique path or by its location in the tree structure (*e.g.* the grandchild of the 3rd child of root).

In live performance and experience design, designers often wish to address a group of dissimilar devices in unison (*e.g.* a bank of lights in different physical locations but focused on the same area). We implement groups, a special type of object that aggregates an arbitrary set of objects. Groups have a special attribute called "members" whose value is a list of member objects. Any operation that can be performed on an object can be performed on a group. The use of groups in this manner is unique to the HyperMedia control system and arises directly from the needs of live performance, which frequently requires synchronized, simultaneous control of multiple elements (*e.g.* light and sound intensity at a particular moment).

2.1.3 Control Objects: Relationships and Arbitrators

Control objects modify attribute values. Relationships are control objects that define a functional mapping between attribute values. For example, a light's intensity attribute can be made inversely proportional to the distance of an actor from the audience by creating a relationship between the two

attributes that correspond to these physical quantities. Relationships provide a powerful way for theatrical designers, directors, and actors to specify dynamic control over media based on real-time sensory input. [4]

Relationships have at least two special attributes: (1) a target and (2) an expression. The target is the "output" destination attribute for the relationship. The expression attribute functionally defines the semantics of the relationship. In the above lighting example, the target would be the light's intensity while the relationship's expression would resemble:

```
1 / (Hamlet.pos.x - Audience.BoundingRect.downstage)
```

In complex environments, multiple relationships may vie for control of a single attribute. [5] When this happens, an arbitrator object examines the competing relationships and determines a single value for the target attribute. Different arbitrators may use different methods (*e.g.* "winner take all," weighted average, maximum value). For example, in addition to the above relationship defining the light's intensity, another relationship might define the same intensity to be directly proportional to the number of audience members in attendance. If the light intensity attribute is assigned to a weighted average arbitrator, the resulting output brightness would retain its relationship to the actor's distance from the audience but be tempered by the audience's size.

The use of arbitrators allows for a consistent and well-defined method of dealing with conflicting relationships; their use frees the (non-programmer) author from the often-complicated task of resolving conflicts. Without arbitrators, script authors would be forced to create ad hoc solutions for each and every constellation of relationships.

Arbitrators in the HyperMedia control system should be distinguished from Metaglue's arbitrators, which act on the level of resources and services. A script informs the system that it wishes to perform a service (*e.g.* show a video) and the system decides the best way to perform the service given the resources available. While this approach is useful for a "smart rooms" application it does not give the script author the control of media elements needed in theatrical production and interactive experience design. The HyperMedia arbitration approach allows the scripting author control not only over services but actual delivery of media while still providing a uniform approach to conflict management.

3. Hypermedia Scriptiong Language

3.1 Related Work

The scripting language incorporates a collection of features drawn from and extending existing control systems for interactive environments, including MPGS [1], Metaglue [5, 10], DAMSEL [9], and others. The core of the Hypermedia scripting language is the creation and modification of real-time relationships between attributes for input (*e.g.* sensors, databases, and the internet) and outputs (*e.g.* lighting, sound, video, and servos) for live performance.

Standard scripting languages "glue" together software components [8]. Similarly, the HyperMedia scripting language connects runtime objects through the creation and modification of relationships. Because few members of the theater community are experienced programmers, this language is designed to be used by technically minded non-programmers.

3.2 Scripting Language Structure

The HyperMedia scripting language is finite state machine based and provides the script author with a single control structure that is computationally powerful (Turing machine equivalent) yet simple to understand. Every script consists of a collection of states with a specially denoted start state. Each state consists of a list of statements followed by a list of transitions. In addition to the finite state machines simplicity and power, states gain representational significance if viewed as analogous to the scenes and acts within a performance. The uniform structure of the finite state machine coupled with the small consistent vocabulary of the scripting language has also been shown in similar languages to aid in learning by non-programmers [7].

Like a standard finite state machine, every script starts by entering its start state. Upon entering a state, the state's statements execute sequentially. Statements support the creation, modification, and destruction of objects, attributes, groups, and relationships and the reassignment of attributes to new or different arbitrators. Scripts can also call other scripts (and wait for them to terminate), or spawn other scripts (and run them in parallel).

After executing a given state's statements the script remains in that state until one of the states transitions are triggered. Each transition has a condition and a state. When a transition's condition is satisfied the script enters the transition's state. Transitions are dependent on attribute values. Upon completing execution of a state's statements the script subscribes to all

attributes present in the transition conditions. Transitions can be made dependent on the time attribute and time can be incorporated into relationships to allow for synchronization and time-based triggers.

Similar to Brooks' subsumption architecture [2], the scripting language implements multiple augmented finite state machines running in parallel (either on a single machine or distributed over the network). Communication between different finite state machines is done through subscriptions to attributes; like all HyperMedia objects, scripts may have attributes whose values can be set and monitored. Unlike the subsumption architecture there is no inherent hierarchy in this scripting language. All finite state machines may have equal control over attributes values.

A departure from traditional finite state machines is the use of state parameters. Every state can have a list of parameters. Upon entering a state, values are bound to the members of the parameter list. The use of parameters allows scripts to be compact and reusable.

Additionally, the HyperMedia scripting language will allow for the runtime creation, observation, and modification of scripts. The ability to make changes at runtime is crucial for the dynamic nature of live theatrical productions and the efficient use of rehearsal time involving many people. In addition, runtime adaptable environments allow for incremental development and testing of scripts without compilation, both of which are extremely helpful to non-programmers.

The following is an example of a script that sets a light's intensity proportionally to the size of the audience and then sets the intensity to also be inversely proportional to an actor's distance from the audience only after the actor has moved within one meter of the audience.

```
(script controlLight
 (state lightAudience () ; start state
   (statements
     light.intensity = audience.size ; adds relationship)
   (transitions
     (((hamlet.pos.x- audience.boundingRect.downstage) < 1)
 (lightActor))))
 (state lightActor ()
   (statements
     light.intensity=
       1/(hamlet.pos.x- audience.boundingRect.downstage)
     ; adds relationship))))
```

4. Conclusion

By restraining the complexity of the HyperMedia control system and scripting language we will allow technically minded non-programmers to construct meaningful real-time relationships across media boundaries for live performance and experience design. In the future, intermediary objects can provide more complex input to output mappings, arbitration schemes, and access to persistent storage allowing new features to be incorporated with the existing HyperMedia architecture.

5. References

[1] E. Bertino, E. Ferrari and M. Stolf "MPGS: An interactive tool for the specification and generation of multimedia presentations," *IEEE Trans. on Knowledge and Data Engineering* vol. 12, no. 1, pp. 102-125, 2000.

[2] R. Brooks. "A Robust Layered Control System for a Mobile Robot." *IEEE Journal of Robotics and Automation.* RA-2:14-23, 1986a.

[3] J. A. Burke, Adam Shive, and Fabian Wagmister. "*Macbett*: A Case Study of Performance & Technology for Dynamic Theater Spaces." *IEEE Multimedia Technology and Applications Conference*, Irvine, California: November 7-9, 2001.

[4] J. A. Burke "Dynamic control of performance environments by online analysis of performer movement," M.S. Thesis, *Dept. of Electrical Engineering,* University of California, Los Angeles, 2001.

[5] M. Coen, B. Phillips, N. Warshawsky, L. Weisman, S. Peters and P. Finin "Meeting the computational needs of intelligent environments: The Metaglue system," *Proc. of MANSE '99,* 1999.

[6] R.E. Lovell "Computer Intelligence in Theater," http://www.intelligentstage.com/, 2001.

[7] E. Mendelowitz "The Emergence Engine: A behavior based agent development environment for artists," *Proc. Twelfth Conf. on Innovative Applications of Artificial Intelligence (IAAI)* pp. 973-978, 2000.

[8] J.K. Ousterhout. "Scripting: higher level programming for the 21st century." *IEEE Computer*, 31(3), 23-30, 1998.

[9] P. Pazandak and J. Srivastava "Interactive multi-user multimedia environments on the Internet: an overview of DAMSEL and its implementation," *Proc. of the Third IEEE Intl. Conf. on Multimedia Computing and Systems* pp. 287-90, 1996.

[10] B. Phillips "Metaglue: A programming language for multi-agent systems," M.Eng., *Dept. of Electrical Engineering and Computer Science*, Massachussetts Institute of Technology, 1999.

[11] C.S. Pinhanez "Representation and Recognition of Action in Interactive Spaces," PhD. Thesis, *Media Arts and Sciences,* Massachussetts Institute of Technology, 1999.

[12] F. Sparacino, C. Wren, G. Davenport and A. Pentland "Augmented Performance in Dance and Theater," *Proc. Intl. Dance and Technology 99 (IDAT99)* 1999.

[13] F. Sparacino, G. Davenport and A.x. Pentland "Media in performance: Interactive spaces for dance, theater, circus, and museum exhibits," *IBM Systems Journal* vol. 39, no. 3/4, pp. 479-510, 2000.

[14] C.R. Wren, et al. "Perceptive spaces for performance and entertainment," *Applied Artificial Intelligence* vol. 11, no. 4, pp. 267-284, 1999.