# Chatscape: A Behavior-Enhanced Graphical Chat Built on a Versatile Client-Server Architecture

by

Matthew W. Lee

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 23, 2001



Author _____

Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by _____

Judith Donath
Assistant Professor of Media Arts And Sciences
Thesis Supervisor

Accepted by _____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

# Chatscape: A Behavior-Enhanced Graphical Chat Built on a Versatile Client-Server Architecture

by

Matthew W. Lee

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 23, 2001

## ABSTRACT

Chatscape is the latest in a series of non-representational chat systems. The system provides users with autonomous behaviors, which form a new channel of online communication. The behaviors allow users to enhance conversation with contextual information, much like body language is used in real-world conversation. These behaviors can act autonomously or at the user's command. The unique configuration of behaviors by each user contributes to their online identity.

A robust, general-purpose server platform entitled SMGServer was developed for several Sociable Media Group chat projects, including Chatscape. Its design was heavily influenced by Chatscape, and its successful implementation and execution was pivotal in the development of several chat projects. SMGServer is now a major component in a set of software tools that will be used for future chat projects.

Thesis Supervisor: Judith S. Donath
Title: Assistant Professor of Media Arts And Sciences, MIT Media Laboratory

# Table of Contents

# Acknowledgements

I would like to thank the following people:

Judith Donath, whose guidance, support, and advice during the last three years, especially during this thesis project, have made my time at the Media Lab and MIT truly memorable and enriching.

Hal Abelson, my academic advisor, who has kept me on a good path through Course 6 and to an exciting career.

Dana Spiegel, for his invaluable design suggestions and implementation assistance for Chatscape and SMGServer, and for picking up the slack on other projects while I wrote my thesis.

Fernanda Viegas, for bringing me into Sociable Media to help build Chat Circles, and for all of her design advice over the years.

danah boyd, for her comments and contributions to the design of Chatscape, as well as the design of one of the behaviors for Chatscape.

Wesley Chan, Michael Goertz, and the rest of my close friends at MIT, for helping me enjoy life at MIT as much as possible.

Mom, Dad, Andy, Chris, Grandma, and Grandpa, for providing unwavering support and love during my life and my 5 years at MIT.

# 1. Introduction

Over the last three years, I have been involved with a series of chat projects for the Sociable Media Group at the MIT Media Laboratory. All of these projects have been non-representational graphical chat interfaces, meaning that while the chat programs represent individual users onscreen using graphics, the graphics that are chosen do not look like human users. During the development of these chat interfaces, several research questions have been studied, and a growing collection of technical knowledge has been built to make future development easier.

This thesis will focus on Chatscape, which is the latest in a series of chat systems. Chatscape was developed to explore how behaviors can be added to a non-representational graphical chat environment, and how the usage of behaviors can contribute significant additional information to a conversation. Communication in a face-to-face setting is much more than just the words that are being spoken; it is a performance of body language, facial expressions, tone, and overall behavior. Chatscape is an online chat system that adds some of this performance aspect of conversation into online communication, using a behavior system that speaks a virtual body language. Each Chatscape user can configure their online identity's behavior settings to best describe the type of person that they would like to be. The result is a more vibrant, more communicative space where conversation is more expressive through the use of behaviors.

In addition to Chatscape, this thesis will describe SMGServer, which is the server technology that is used to build most of the chat projects that have been developed in Sociable Media, including Chatscape. SMGServer is used strictly as a fixed foundation for project-specific chat servers, ideally requiring no modification to SMGServer code by developers. The server began as part of an overhaul plan for Chat Circles, the first

graphical chat program developed by Sociable Media. However, several other projects where SMGServer was used factored heavily into SMGServer's design, especially Chatscape. The initial design for SMGServer focused on Chat Circles, while accommodating the anticipated requirements for the other projects. As the other projects came to fruition, the decisions that were made for SMGServer's architecture were proven correct, as only a small amount of architectural changes were necessary to support all of the other projects. Because of its success, SMGServer is now one of many underlying tools that are used by Sociable Media students to build new chat interfaces.

# 2. Goals

The largest research area of Chatscape is a study of online identity. The online world has a different set of rules and structures for expressing identity, since real-world concepts like clothing, body language, speech patterns, and other such things do not have direct analogues in the online world. Despite the different ways of representing identity that the online world must use, influences on your personal identity can come from similar places, such as your own influences and the influences of others. Chatscape is a system where the identity profile you develop for yourself can also be adjusted by other users. One of the research goals for this project was to see how online chat is affected by this model of identity. Furthermore, I wanted to see if this model would make online chat more interesting or entertaining.

SMGServer can be seen as a separate project that is closely tied to Chatscape from a technological point of view. Although a completely custom chat server could have been developed for Chatscape, it made more sense to develop SMGServer as a server platform for multiple projects. Also, a new chat server platform was needed for other projects before it was needed for Chatscape. Therefore, the goal of SMGServer was to provide a general-purpose chat server platform to multiple projects, while allowing a great deal of customizability so that the general-purpose nature of SMGServer would not be a hindrance.

# 3. Previous Research

Several previous works influenced the research directions for Chatscape and influenced design decisions for SMGServer. This section will highlight some of the more prominent pieces of work that are referenced in this thesis.

## 3.1. Chat Circles


**Figure 1: Chat Circles 2.**

Chat Circles [Viegas 99] is an abstract graphical chat room project designed by Fernanda Viegas. Shown in Figure 1, Chat Circles features a single chat room for all conversations, instead of using multiple separate rooms or “channels” that other chat

systems use to separate conversations by topic. The chat room is a large 2D virtual space, in which small colored circles represent users; each user can move their circle by dragging it with their mouse. When users chat, their circles expand, and their chat messages are drawn inside their circles. After a short period of time, their circles contract back to a small size. The concept of "hearing range" is introduced to filter out separate conversations; a user can only read chat messages from other users that are within a certain radius from the user. This feature makes people intuitively arrange themselves into separate conversation clusters, much like people at a party.

Chat Circles forms the foundation for Chatscape, both visually and technically. The Chatscape chat environment is visually very similar to Chat Circles, featuring simple graphical shapes, hearing range, and one single chat room. Chat Circles is designed to be a very simple, featureless graphical chat system that can serve as a base for more complex chat systems. Chat Circles is a direct antecedent to Chatscape, both conceptually and technically; Chatscape seeks to apply additional conversational methods into graphical chat, resulting in a more complex environment.

## 3.2. Body Chat

Body Chat [Vilhjálmsson 98] is a graphical chat system developed at the MIT Media Lab that uses 3-dimensional human-like avatars to represent users in the system. Users use standard text chat to communicate with each other in the system. Body Chat features a behavior system that is used to manipulate each avatar in response to the current conversation, using hints derived from the text chat messages. Most of these behaviors are used to maintain the human-like appearance of the avatar; for example, behaviors are used to indicate attention, facial expressions, blinking, and other "maintenance" tasks that are intended to contribute additional information and context to

the conversation. Additionally, each user is in control of a few high-level aspects of their avatar's behavior, such as conversational availability.

Chatscape features similar groupings of behaviors as Body Chat, such as behaviors that are largely autonomous, versus aspects of behavior that are controlled by the user. However, Body Chat is used here as an example of how the selection of behaviors can be restricted by the choice of avatars in a chat system, and therefore the resulting behavior sets may not be able to contribute much worthwhile information to a conversation. The choice of a human-like avatar brings with it all sorts of expectations which must be satisfied by maintenance behaviors that make the avatar's appearance acceptable and believable. If the behaviors are poorly done, or not appropriate in their operation based on the ongoing conversation, they can actually detract from the value of the conversational experience. Chatscape, on the other hand, uses abstract user representations that are mostly free of predefined expectations, removing a huge burden from the selection and operation of the behaviors available to users.

## 3.3. Comic Chat

Comic Chat [Kurlander 96] is a graphical chat interface that displays conversations as ongoing comic strips, styled after the Sunday funny pages in a newspaper. Users select one of a handful of cartoon characters to represent themselves, and as they chat in a chatroom, their character is added to each panel as necessary. Comic Chat also allows users to specify an emotion from an emotion wheel that is displayed by their character when they chat; while this is a small amount of additional information, it helps contextualize the current conversation quite well. While Chatscape does not provide functionality as explicit as an emotion wheel, it serves to demonstrate how simple behaviors can greatly enhance a conversation.

Another interesting aspect of Comic Chat is that while each user can select their character and emotion while they chat, it is ultimately up to the system to choose how their character is displayed in the comic panels. Through intelligent analysis of conversation patterns and the usage of a few simple rules, Comic Chat lays out the cast of chat room participants in the most aesthetically pleasing manner. This type of arrangement is not used in Chatscape, but there are a few aspects of Chatscape that remove total control over one's appearance from a user.

# 4. Overview of Chatscape

Chatscape is a client-server graphical chat system, meaning that there is a client application that users download and use to access the Chatscape system, as well as a separate server application that each user's client connects to over the network. The client displays information, including the chat conversation, as graphics in a window. However, Chatscape is not the first chat system of this type.

Chatscape is the most recent chat system in a series of chat systems based upon Chat Circles. Throughout this lineage of chat systems, several different issues pertaining to online chat have been explored, including the behavior system in Chatscape. Chat Circles was followed closely by Talking in Circles, an audio chat project; later came TeleAction; a collaborative tele-direction project, and finally Chatscape.

Chat Circles was meant to be an extremely simple abstract graphical chatroom; one that could be quickly grasped by users. Details were kept to a minimum in Chat Circles for several reasons; not only was the system intended to explore a few select issues, but it was also designed as a "base" project for other projects to build upon in the future.

Talking in Circles [Rodenstein 00], Roy Rodenstein's MAS masters thesis project, took Chat Circles into the audio chat realm by adding realtime voice communications. A screenshot is shown in Figure 2. Text chat messages were replaced by a realtime audio amplitude "throbber" in each user's representation,

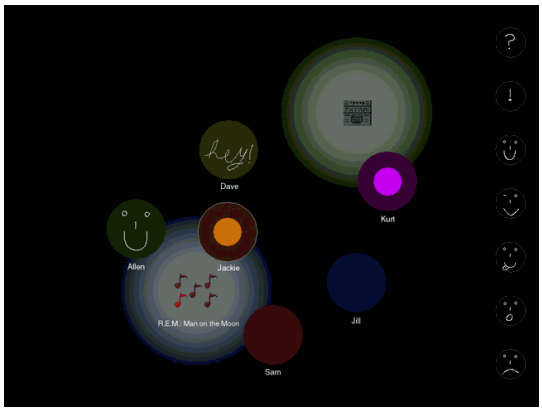


**Figure 2: Talking in Circles**

allowing a user to quickly identify another user's voice based on visual activity from their circle. Additionally, the system allowed each user to draw on their own circle, which was a freeform way to temporarily "mark up" one's identity in the space, providing a second mode of communication. Additionally, Talking in Circles featured areas in the chat space that played various audio clips or live audio feeds, which could give conversations a pleasing background or a focus of attention.

The TeleAction system [TeleActor 01] was developed to explore the idea of allowing a group of people online to collaboratively control a scarce resource, in this case a remote human "actor" outfitted with a video camera and other equipment to receive directions from the online crowd. The video from the actor is displayed as a "stage" in the graphical chat environment, and serves as a work area for group decision-making. As the crowd votes on what goal the person should perform next, a voting system collects the votes and relays the winning goal to the actor. While the graphical representations in TeleAction are just as simple as in Chat Circles, the role of movement and position in the space are extremely important in the voting process. While Chat Circles and TeleAction used movement only for conversational filtering based on hearing ranges, movement in TeleAction is used to highlight positions in a constant video feed and demonstrate support for others' goals. This system therefore encourages a highly functional gesture language for collaboratively making decisions.

Chatscape takes elements from all of its predecessors, resulting in a highly unique graphical chat environment. A more flexible graphical representation is used to display several different aspects of identity and behavior. By default, the representation is a colored regular polygon, but the polygon can be distorted and adjusted in several distinct ways. Autonomous behaviors are available to each user in order to describe a detailed identity; with this identity, the combination of the user's actions and the actions performed by behaviors results in a complex, yet readable, online persona.

In this way, behaviors are used as another conversational medium, much like body language is used to augment real-life conversations with additional detail or contextual hints. Each user has a persistent profile in the Chatscape system, which allows behavior settings to be kept from session to session, allowing users to establish and use a personal and identifiable set of behaviors over a long period of time. The interface for adjusting behavior properties is made as simple as possible, geared towards being used to make small tweaks as quickly as possible.

The combination of behaviors with simple abstract graphical representations allowed me to create a completely new behavior language for Chatscape, one that would be free of pre-existing stereotypes and expectations. Nobody expects any particular mannerisms to be exhibited by colored shapes. However, there are a few visual metaphors that can be recognizable from colored shapes, such as roughness and smoothness representing chaos and tranquility, respectively. Even though certain properties of shapes can be generally recognized as different moods or meanings, the shapes do not come off as anthropomorphized. Body Chat, as mentioned earlier, features avatars that are rife with expectations; a humanoid avatar that did not exhibit basic human-like motions and expressions would be disconcerting and strange to users. Additionally, the lack of recognizable avatars allowed me to display interactions between users that would be unexpected or impossible in humanoid avatars, such as one user making another user move around against their will.

# 5. Client Description

The Chatscape user interface is designed to allow users to quickly grasp the concepts behind graphical chat, and manage the complexity that can be associated with controlling and configuring all of the behaviors. Figure 3 shows a typical chat session between 3 people.

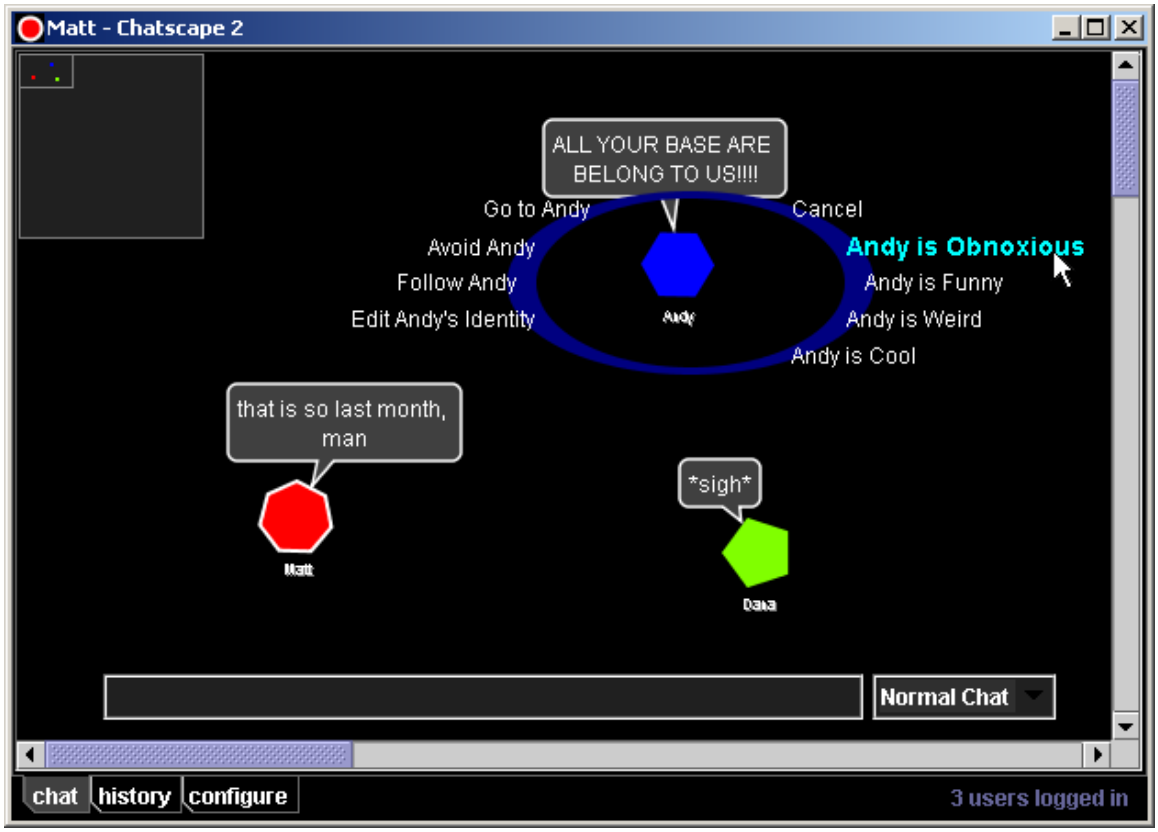


**Figure 3: The Chatscape graphical chat interface.**

The Chatscape window features several elements. The main area is a window into a large virtual chat environment; in this case, the chat environment is 2000 by 2000 pixels, and the window is only a few hundred pixels on a side. At the bottom of the window is the chat input area, where a message can be typed – pressing Enter after typing a message submits the message. It is accompanied by a “chat type” combo box, which can be used by behaviors to process chat messages in various ways before

submitting them. In the upper left corner, a mini-map reminds users what part of the chat space that the window is currently revealing; the current window is the small rectangle, and the boundaries of the chat environment is the large square. Users are represented in the mini-map with a colored dot, the same color as their representation. The scrollbars to the right and bottom of the window can be used to view other areas of the chat environment. Finally, at the bottom of the window are three display tabs, that take the user to the chat history and the user configuration screens, which will be discussed later.

In Figure 3, Matt is the local user, which is also reflected in the window's title bar. The three users, Matt, Andy, and Dana, are positioned at the upper left corner of the large graphical chat space. Each user is represented by a colored shape, and can move their own shape around by dragging it with their mouse. The local user's shape is outlined in white. The white outline graphical metaphor for the local user is a Sociable Media standard of sorts for graphical chat, dating back to the first version of Chat Circles in 1998. Shapes can be dragged to offscreen areas of the chatroom by dragging beyond the extents of the window; the scrollbars will scroll to the desired location, and the minimap will update accordingly. Chat messages are displayed in chat "balloons", similar to those of Comic Chat and other comic book illustrations. After a short period of time dependent on the length of the message, each chat balloon deflates and disappears. Each user's name is displayed directly below their shape, and is initially quite small. When the local user moves their mouse over a user's shape, their name expands to a more readable size. This was designed so that the chat space would not be cluttered by names.

Surrounding Andy in the chat space is a context menu, which was summoned when Matt clicked on Andy's shape in the chat space. The selection of items in this menu reflects the options that Matt's behaviors have provided for operation on Andy.

Andy has just spoken an oft-repeated nonsensical phrase, which by this point has become obnoxious to Matt, as well as Dana. Matt has moused over the “Andy is Obnoxious” command, and that command is highlighted since the mouse is placed over it. The command is executed with a single click, and the menu closes.

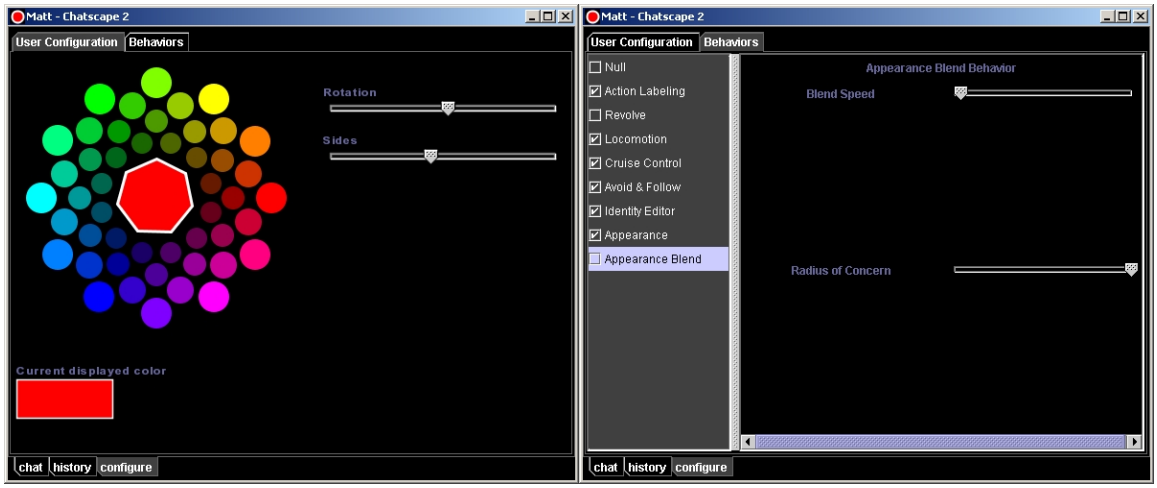


**Figure 4: The user and behavior configuration panels.**

Figure 4 shows the two configuration panels, accessible via the “configure” tab at the bottom of the window. The user configuration panel is where the user’s color and shape details can be changed. The shape has two directly configurable options; the side count can be changed from 3 to 16 sides, and the rotation rotates the shape about its center, allowing two users with the same number of sides in their shape to look different.

The behavior configuration panel is where behaviors can be enabled or disabled, as well as configured. The list on the left shows the full list of behaviors that are available, along with a checkbox indicating whether each behavior is enabled. Double-clicking on a behavior enables or disabled a behavior, except for behaviors that are always enabled, such as the Appearance behavior. The always-on behaviors provide essential functionality to the system, and cannot be disabled. When a behavior in the list is selected using the mouse, configuration options for that behavior (if there are any) are

displayed in the right side of the panel.  In Figure 4, the two options for the Appearance Blend behavior are displayed, and changing these sliders affects the behavior in real time.

# 6. Behaviors

The goal of using behaviors in Chatscape is to allow participants to use behaviors as another channel for information exchange. This goal reflects observations from real-world conversations, where people often use body language to enhance, clarify, or contextualize their spoken words. Also, behaviors in a chat system can be used in ways that go beyond what is possible in real-life conversation; an example being behaviors that allow people to directly control another user's identity, without that user being able to intercede. Also, on a more functional level, behaviors in Chatscape are sometimes used to simplify complex or time-consuming tasks, like moving around in the space.

Chatscape differentiates itself from previous behavior-based graphical chat environments by using extremely simple graphical representations (colored shapes) as avatars. These shapes are endowed with many different visual attributes, including autonomous and user-driven behaviors. These attributes range from simple things like the color and shape of a user's representation, to complex system-wide behaviors that involve the actions of multiple users over a long period of time. These attributes can be combined together in various combinations by each user, resulting in a virtual identity profile that represents a user in the Chatscape environment.

Because Chatscape's user representations do not belong to a common theme in chat programs, I could define a brand new vocabulary for behaviors in a graphical chat system. Consequently, every user of Chatscape will learn to interpret this vocabulary without any prior expectations factoring into their interpretation.

Chatscape provides over a dozen behaviors and visual attributes that can be used to give each user a unique identity. Each attribute can be thought of as belonging to one of three categories. The first category includes attributes that a user directly

controls. The second category is the attributes that only other users have control over. Finally, the third category is attributes that only the chat system can control; these attributes usually affect users over a long period of time.

## 6.1. Personal identity traits

The first category is a very established type of attribute among all online communications tools. As such, these attributes are usually quite simple. In Chatscape, this category of attributes includes a user's name, a user's color, and the shape of a user's representation. These attributes are chosen for a new user immediately as they join the chatroom, making them the most immediately recognizable aspects of a user's identity. In addition to these simple attributes, there are two more attributes in this category that are more complex, yet still completely controlled by a single user.

The movement behavior allows a user to navigate about the chatroom in a more passive way than direct manipulation of the representation (dragging with the mouse). A user can simply click on a spot in the space, and the movement behavior will "walk" there. The movement behavior has several different styles of movement, each with their own individual character.

Another behavior, the avoid and follow behavior, was developed as an extension of the movement behavior. By clicking on other users, each user can maintain an internal list of users they would like to follow, and users they would like to avoid. When a user decides to follow two other people who are not near each other, the user will be drawn to a point between the two other people, as if they were in a tug-of-war match. Avoiding multiple users is more meaningful; the user is simply pushed away from all of the users they wish to avoid, as quickly as possible. Overall, the avoid and follow behavior allows a user to maintain a close distance to their friends, and stay away from

undesirable users, while freeing up their hands to chat as this movement is going on autonomously.

## 6.2. Identity traits set by others

Chatscape's second class of behaviors are those where other users can directly control aspects of your identity. There are very few examples from the real world that demonstrate this type of behavior, since most people are ultimately in control of their own identity at all times. One example of this behavior would be when a group of people gives a label to a person, and people react to that label in a manner separate from the reaction to the person. Military insignia works in this way as well; a small piece of decorative cloth granted by an organization of people brings with it expectations as to how people should act around a person wearing this insignia.

There are several online systems that use a variant of this concept extensively, in the form of "reputation" systems. Web sites like eBay, Epinions [Web 01], and other sites that negotiate transactions and information exchanges between their users require a way to associate an individual's value and responsibility with said transactions. The reputation that each user has in the system is directly influenced only by other users, and it is usually manifested in a numeric rating of some sort. Also, web site reputation systems tend to have long term and often irreversible effects; feedback collected from other users usually becomes part of your "permanent record" on a particular website. Chatscape behaviors that are controlled by other users are intended to operate in a similar fashion, but represent more subjective or intentionally vague topics than "how many people have said good things about me". The feedback mechanisms are much more casual, implemented through simple one-click interfaces that are visually connected with each user in the space. Also, Chatscape reputation feedback is

intended to be more short-term and ephemeral. No reputation-like behaviors have any permanent or even long-term effects; in most cases, any graphical indication of feedback from others is gone within minutes. The lack of permanence for reputation data contributes to a more friendly chat environment, where even the most annoying users are able to make amends with others over time.

The hearing range that is present in Chatscape results in users forming natural "conversation clusters" around the space. The appearance blend behavior can further differentiate these clusters by blending together the colors of users that group together for a period of time. For example, if a red shape and a blue shape were talking to one another for several minutes, they would both slowly transform to purple shapes. When the two shapes finally move apart, their colors blend back to their original hues. An anticipated side effect of this behavior is a sort of fashion feedback loop; since the effects of this behavior are delayed and work over time, certain hues could be transferred across a large group of users as they all move about.

While the appearance blend behavior is more of a participatory behavior, the group labeling behavior is a much more direct, person-specific way of allowing other users to affect one's identity. With this behavior, anyone can select a label from a list of labels and apply that selection to another user. The labels include "Obnoxious", "Funny", "Weird", and "Cool"; each label causes a different change in another user's appearance. The Obnoxious label is the most interesting label. It was conceived out of a desire for some sort of moderation system for obnoxious or disruptive users. When multiple people tag a user as "Obnoxious", their shape develops spikes, which symbolize hostility. The spikes are an example of how colored shapes can express a commonly acceptable notion like hostility while remaining nondescript and abstract. Over time, the spikes smooth out and disappear, provided the user has not been tagged as Obnoxious

anymore.  This allows someone to make amends to other users, or recover from unfair labeling.

# 7. The Chatscape Design Process

Chatscape began as an undergraduate research project in Spring 2000. It builds upon work done by others and myself before that time. While the first versions of Chatscape were very similar in appearance to the current version, the implementation has changed a great deal over the course of the project. During the various stages and prototypes of Chatscape's development process, several features and design elements have remained relatively constant, while others came and went as the project gained focus over time.

The behavior system is the largest aspect of the early Chatscape prototypes that was transferred to the final Chatscape without significant changes. This module was called the Behavior Manager. The prototype behaviors, much like Chatscape behaviors, were each individual objects, implementing a common interface. The Behavior Manager organized behaviors in an array, much like SMGServer organizes users in Environments. The entire volume of message traffic going into and out of the client was also sent to each behavior, essentially making each behavior a participant in the chat system just as much as the human user operating the client. This allowed behaviors to respond either passively to specific message types, or operate in a more active mode, periodically taking action based on an internal timer.

The other major feature that carried over relatively unchanged was the context menus. To activate behaviors on other users or locations in the 2D space, a context menu is used. The entries in these context menus are determined by the set of active behaviors and the selected user or location. Some behaviors rely on these context menus for input from the user; for example, the Avoid and Follow behavior uses context menu input to build the lists of users to be followed or avoided.

Instead of implementing a simple rectangular popup menu, similar to right-click actions in Windows applications, I chose a more interesting circular pop-up menu inspired by the context menus in the popular computer game “The Sims” [Maxis 00], which are based on research by John Carroll [Carroll 91]. In addition to being more visually attractive, the menu does not obscure the user or location bring selected, as the menu items surround the selection. The only drawback to these menus is scalability. As the number of items in the menu increases, all items in the menu get farther and farther from the selection.

While the context menu concept did not change in appearance during the project, the implementation did change a great deal. The initial implementation featured a reasonably complex command registration model. This model allowed behaviors to register a command that would appear on the menu, and if the behavior were to be disabled, it could unregister commands as well. This worked well, but it relied on the proper operation of behaviors to make sure that commands did not appear on the menu when they weren’t supposed to.

The second and final version of context menus featured a polling model, where each behavior is asked to provide commands for the menu each time the menu is opened, in the context of what the menu was opened upon. For example, if the user clicks on the user “Matt”, each behavior is asked what commands should be presented for the context menu for Matt. This model incurs more processing overhead than the previous model, but as a benefit, it allows commands to be customized for the user or location being clicked on. Instead of the avoid and follow behavior presenting the simple commands labeled “Avoid” and “Follow”, it can present “Avoid Matt” and “Follow Matt”. Additionally, if the user is following Matt and clicks on Matt again, the command “Stop Following Matt” can be presented by the behavior.

There were a few concepts and ideas that were dropped or not developed during the development of Chatscape. The first such concept was the notion of having users in the system that were completely controlled by behaviors, and therefore run as part of the chat server. These automated users were introduced for two reasons. First, they were seen as a way to make a more continuously interesting chat room, where things would be happening even if no human controlled users were in the space. Second, they were introduced as an experiment, to see if people could distinguish other behavior-enhanced human users from completely computer-driven users.

Two computer-driven personalities were developed, the first called “Village Idiot” and the second was “Eliza”, named after the famous virtual psychologist program. Village Idiot simply spouted out a random “insane” sentence when people chatted around Village Idiot. The insane sentences were harvested from the latest version of the text editor Emacs, where they are known as the “pinhead” automated response program. Eliza, as mentioned before, is a virtual psychologist. For the code, I re-wrote a simple JavaScript version of Eliza that I found on a web page.

Ultimately, the two personalities were just a gimmick. Their gross simplicity made them clearly distinguishable from a human, and they added no real value to Chatscape, other than some occasional entertainment. Also, the method of their operation was very similar to Internet Relay Chat (IRC) “bots”, which are simply scripted programs that respond to people using IRC [Oikarinen 93]. Based on these factors, I decided that the personalities were not interesting enough from a research standpoint to re-implement for the final version of Chatscape.

One concept that never left the planning stages was that of interactive backgrounds, where the background of the chatroom would have some sort of geographical diversity that would interact in some way with behaviors. The most interesting scenario that I developed was that the Chatscape space could have areas

with different visual properties, such as areas in the light and areas in shadow, and that behaviors would be able to distinguish between the two and move around accordingly. Also, similar to the audio spots in Talking in Circles, there would be “landmarks” in the chat space that would be interesting or intriguing, causing behaviors to react differently in their vicinity. While this concept was not developed at all in Chatscape, a variant of this was put into Chat Circles in November 2000, in the form of pictures that could be placed around the chat space. The pictures were meant to serve as conversational pieces, much like the paintings in an art gallery.

# 8. The SMGServer Design Process

SMGServer was conceived as a server in a client/server architecture that would serve the needs of several Sociable Media Group online chat projects, including the new version of Chatscape described in this thesis. At the time, Sociable Media's first major online chat project, Chat Circles, was a permanent web installation, and was attracting a small yet steady flow of regular visitors. However, the Chat Circles server was not capable of running for long periods of time, especially with a constant load of users. At the beginning of fall term 2000, I decided to build a new server for Sociable Media, one that could be initially used for a new version of Chat Circles, and then used for my thesis project a few months later. Additionally, the TeleAction project that was just getting started in Sociable Media was in need of a chat server. As the server was going to be used for a variety of Sociable Media Group projects, I named the project "SMGServer".

The most important factor in the design of SMGServer was the assumptions that were made about what features projects would demand from SMGServer in the future. There were several projects still in planning phases, and some projects that were already well established, like Chat Circles. These projects collectively required a large number of features from a chat server, and I had to decide which features would be the most generally useful to most or all projects.

The first project I took into consideration was Chat Circles 2. This project was to be a complete overhaul of an established project, with the goal of providing a more usable and reliable graphical chat platform for future projects to build upon, as well as a permanent fixture on the Sociable Media web site. Chat Circles did not demand much from its server; the feature list included simple message passing between users, a small amount of properties to be kept with each user, and a simple file logging interface for writing chat log files to disk. The login process was very open; no persistent user

accounts or passwords were needed to login to Chat Circles. Additionally, Chat Circles only required one “room”, while most other chat systems, like IRC, support hundreds or thousands of simultaneous rooms.

The next project that I considered was the TeleAction project. This project, from the point of view of a developer, was very similar to Chat Circles. Only one room was needed, the login system was the same, and very little information was needed for each user. However, TeleAction introduced a live video stream into the graphical chat interface, which needed to be controlled by an “administrator” that would somehow be logged in to this chat system. Briefly, I considered allowing SMGServer to be the mechanism that would deliver the streaming video to clients; however, after realizing the performance requirements for this task, and the fact that only TeleAction would be using video, I decided to leave the actual task of delivering the streaming video to commercial software designed for that task. Therefore, SMGServer would only have to distribute control messages of some sort that would instruct clients about how to access a separate streaming video server.

Finally, I considered Chatscape as the third project to use SMGServer. Chatscape was somewhat similar to Chat Circles as well, but a few important new features were required above what Chat Circles and TeleAction needed. First, the data associated with a user was going to need to be very detailed and very flexible; each behavior would have a multitude of properties, and the various aspects of each user’s identity would need their own data elements as well. This set of properties would need to be kept with each user on the server, and distributed out to each user as they connected to the system. Also, to support long-term development of identities, Chatscape needed two things – persistent storage of user properties and a password-protected login to prevent identity theft.

After getting a list of features from the projects that SMGServer would be used in, I briefly studied the existing Chat Circles server code for additional insight into the design task. I noticed that while the project-specific code in Chat Circles was fairly robust, the "plumbing" code that took care of more mundane tasks like message passing, network I/O, and user logins was not very reliable, especially over long periods of time. This made sense, since from the point of view of an academic researcher, the unique aspects of a specific project are worthy of more attention than the underlying generic code that is required in order to make the system work. I concluded that if SMGServer could provide a generic, robust, and reliable level of "plumbing" to project servers, it would save a lot of development time and allow researchers to focus on project-specific server code and not the intricacies of network programming.

The original design document for SMGServer was never formally completed, but the high-level design and the feature list were discussed extensively with Dana Spiegel (a master's student who would be using the server), Prof. Donath, and others. The server is implemented in Java, which is the most popular development language in the Sociable Media Group, and therefore SMGServer would be more accessible to other developers in the future. The high-level design of the server is a message-passing architecture, where messages from users are received from each user and sent out to all other users of the system. This is a very standard architecture, and is used by many commercial multi-user servers for gaming, chat, and other applications. Message passing is only one of the set of services that SMGServer provides, along with user logins, user data management, and user group management.

The process of accepting a login from a user involves several steps. First, an incoming TCP/IP connection from the client must be opened and kept track of. Next, login credentials from the user must be validated, resulting in the decision to accept or

refuse login. Finally, a successfully logged in user must be added to the current set of users that are already logged in, so the new user can communicate with other users.

User data management refers to the maintenance of a set of data that is associated with each user. For a typical chat project, this usually includes data like a user's name and the IP address they are connecting from, for logging purposes. More complex chat systems may need to store more detailed information about a user, such as color, shape, and location in a virtual space. To provide the most flexible and extensible user data storage structure, SMGServer maintains a simple key-value data structure for each user, which allows any arbitrary object to be stored and referred to by a string name. A hashtable is used for this data structure. While a hashtable may be considered overkill for storing one or two data items, it maintains a high level of efficiency when tens or hundreds of properties are involved, and the interface remains the same regardless of the amount of contents.

SMGServer arranges users into "environments", which are simply arrays of users that represent a group of users that can communicate with one another. This concept has different names in other chat systems; for example, Internet Relay Chat calls these groupings "channels". SMGServer includes this functionality for two main reasons. First, the message passing architecture is centered mostly around the environments, as this is where messages from one user are broadcasted out to other users. Second, this functionality is necessary for all of the projects SMGServer was being designed for.

Although all of the projects that I considered during the design phase only required one room, I decided in the design phase to support multiple simultaneous environment objects in SMGServer. The added complexity of supporting multiple environments versus just one environment was not a burden, and it seemed like the right thing to do at the time, even though there was no perceived need for this feature. This design decision was vindicated later, when Dana Spiegel started the ChessMates

project. ChessMates is an offshoot of the TeleAction project featuring team-versus-team collaborative chess gaming. Initially, Dana tried to put both teams in one environment, but the complexity of separating team conversations into two groups of users in one data structure proved to be too great. A second environment was added, so that each team would use its own environment. The dual-environment approach proved to be the right choice, as the complexity of maintaining two separate teams of users was somewhat offloaded onto the SMGServer architecture.

To customize SMGServer for a specific project, a developer must build only two modules that contain all of the functionality that makes that specific project unique. Through the usage of Java dynamic class loading, these modules are discovered and loaded at runtime by SMGServer. The first module, called the Server Core, includes several hooks into SMGServer that are activated for various events. These hooks allow the Server Core to perform custom startup tasks, modify users as they log in, and receive a copy of every message that is sent through the system. Since the Server Core is kept abreast of all of these different events, it can perform custom tasks in response to just about anything that happens within the server. The second module is called the Login Validator: it allows a developer to validate user logins based on the set of credentials that the client presents at the beginning of a session. This module was made customizable so that a developer could make complex decisions based on login credentials. For example, a developer could look up the credentials in a database, and only allow access to users in the database.

A developer using SMGServer for a project must implement the two aforementioned modules, but is not expected to have to modify SMGServer itself. In fact, a design goal of SMGServer is that it is distributed as a library, not a code base. Developers should not need to modify SMGServer source code in order to achieve their

desired functionality. By strictly enforcing the modularity of SMGServer in this way, the long-term usefulness of SMGServer as a generic server platform will be better ensured.

SMGServer was also designed to make client development easier for a new project. A great server is useless if communicating with it is difficult. Therefore, a significant portion of the network communications code on the server side is also used on the client side, and a few extra client-side modules were designed to make client development relatively hassle-free.

# 9. Implementation

This section will cover the technologies used to implement Chatscape and SMGServer, and highlight several important implementation details that affected the operation and performance of both projects significantly.

## 9.1. Technologies Used in Chatscape and SMGServer

Both Chatscape and SMGServer are implemented in Java, more specifically the Java2 platform. Early Chatscape development work was done in Java2 version 1.2, while SMGServer and the final version of Chatscape were done with Java2 version 1.3. Java was chosen so that the Chatscape client could run on a variety of Java2 platforms, including Windows, Linux, and most popular Unix operating systems. SMGServer was written in Java so that it could better integrate with Java clients, and also because of the easy to use and feature packed Java class libraries.

Development was done primarily on Pentium 3 workstations running Windows 2000, using the Symantec/WebGain Visual Café 4.0 development environment. Visual Café was chosen because of its familiarity and feature set, and its successful use in past projects. Some SMGServer development work was done by Dana Spiegel on similar workstations, using the Metrowerks CodeWarrior development environment. One of the benefits of Java was that the same code could be developed by people using different development environments; other languages such as C++ tend to be more tied to a specific development environment in order to use advanced features.

SMGServer was deployed on Linux first, followed by Windows 2000 Advanced Server. Initially, the Sociable Media “server machine” was a Pentium 2 300 MHz computer running Linux, and the first Chat Circles 2 server powered by SMGServer was deployed on that machine. However, after months of active use, it became apparent

that the Java runtime environment on Linux was not adept at handling applications with dozens of threads, like SMGServer. The Chat Circles 2 server was then moved to a dual processor Pentium 2 300 MHz machine running Windows 2000 Advanced Server, and the server has performed much better ever since.

The Chatscape client, like the Chat Circles 2 client, is deployed as a Java applet on a Web page, running on the Java Plugin. The Java Plugin is a full implementation of the Java2 platform that operates as a browser plugin for popular browsers on many platforms. Although a Java runtime engine is a common built-in feature in Web browsers, the engine provided with almost all popular browsers, including Microsoft Internet Explorer and Netscape Navigator, is outdated and not very reliable. The Java Plugin allows any plugin-compatible browser to run a full-featured, modern Java2 environment, with a seamless appearance to the user.

## 9.2. SMGServer Implementation Details

This section highlights the interesting aspects of SMGServer's development, including changes that were made to the design of the server long after the project was initially developed.

### 9.2.1. Threading

Fundamentally, a chat server is responsible for passing messages from one user to all other users, and perform this task for all users at once. However, a computer can only do one operation at a time, so a strategy must be developed for performing many simultaneous actions all at once while doing only one action at a time. The simplest solution is to just address each user individually in a round-robin fashion. However, this leads to a problem – if one user takes a large amount of time to send or receive

messages, the whole system must wait before servicing the next user. This problem, known as “blocking”, can be avoided by using multiple “threads” of execution, where many individual tasks are done at once, and the operating system is responsible for timeslicing all of the threads, so that only one thread is running at a given instant.

The Quicktime Streaming Server [Apple 00], also known as the Darwin Streaming Server, is a freely available multimedia streaming server for the popular Quicktime digital video format. While this product bears little functional similarity to SMGServer, part of its design can be compared to how SMGServer handles many user connections at the same time. The Darwin Streaming Server FAQ briefly discusses the system architecture, which describes how the system uses only a handful of threads, and does not spawn one or more threads for each user.

SMGServer uses threads extensively in its design; in fact, 2 threads are spawned for each user that is connected to the server. That way, if one user is slow, only their own threads are blocked, and the rest of the system operates normally while the slow user waits. Therefore, a consistent level of service is given to users who are operating normally, regardless of the quality of operation of other users in the system. However, this is not the only benefit; since each thread is a separate running entity, an error that occurs in one thread does not necessarily affect other threads. In this way, threads can provide a level of fault isolation, so that errors in one part of the system do not bring down the entire system. Additional threads are employed to “clean up” after threads suddenly die due to errors; this ensures the system remains operational unless a truly devastating error occurs.

The large number of threads used by SMGServer is somewhat of an anomaly compared to other multi-user servers, like the Darwin Streaming Server. Java lacks a feature known as asynchronous input/output (I/O), which is a way of accessing network and other communications resources in an asynchronous fashion, rather than a

synchronous fashion. Asynchronous I/O ensures that sending and receiving data never blocks a thread of execution; instead, the underlying operating system sends notifications to a program when communications operations start and finish, in an asynchronous fashion. In a way, the large thread count in SMGServer is a workaround to this limitation in Java. The added complexity that is required for the threads in SMGServer are handled by well-debugged libraries in other languages. Despite this seemingly major drawback, the rest of Java's features make up for this glaring omission – but because of popular demand, asynchronous I/O will be a part of the next major release of Java.

### 9.2.2. Sample Modules for Developers

As mentioned earlier, developers must implement two modules in order to make a custom SMGServer-based chat server for their project. While the requirements for these modules are not very demanding, I decided to give developers even more to work with as they start a new project. Sample modules are included with SMGServer that provide a very basic level of functionality, and developers can choose to either use these sample modules as-is, or extend upon them using Java object inheritance rules. The sample modules include NullServerCore, a simple chat server core that implements one room; FreeLoginValidator, a login validator module that accepts all credentials and logs users in; and FileLogModule, a sample logging module that developers can use within their Server Core module to log output to a file.

### 9.2.3. Tighter Integration with Windows NT/2000 Services

The process of running a program in the background on a Unix machine such as Linux is relatively straightforward; the command-line utility "nice" can run a program in

the background, and keep it running after you log off. However, Windows NT/2000 operating systems do not provide any such utilities. Background processes, or "services", as they are called in Windows, must be specially designed native-code applications. While this may seem like an additional hassle for developers, the special structure of a Windows service executable allows Windows to send special messages to each service for startup and shutdown procedures, allowing services to start and stop more cleanly.

The JavaServ service runner [Giel 00] combines a Windows service executable with a Java runtime environment, which allows Java programs like SMGServer to be run as a service. I downloaded JavaServ, made a few small modifications, and now distribute JavaServ with SMGServer. Additionally, JavaServ has been used successfully for the Chat Circles server for several months now.

### 9.2.4. Developer Feedback

Feedback from developers was very important during the development of SMGServer. Dana Spiegel, during the development of TeleAction and ChessMates, contributed many suggestions and some development work towards the betterment of SMGServer. As designed, the server was used only in binary form in Dana's work, and he commented how stable the server was, as well as how quickly he was able to implement projects using SMGServer. Dana's development contributions to SMGServer fell mostly in the areas of error handling and a better structure for the Environment object's connection back to Server Core objects. This better structure made the development of the ChessMates server much more straightforward and error-free.

## 9.3. Chatscape Implementation Details

The final version of Chatscape was built directly on top of the Chat Circles 2 code base, which was developed in October 2000. Several factors were involved in the decision to use Chat Circles 2 as a starting point for Chatscape. First, Chat Circles 2, on a high level, is very similar to Chatscape, in terms of a single graphical chatroom and simple geometric representations. Most of Chatscape's development consisted of adding modules to this relatively small codebase, instead of taking things away. At the point where Chatscape development started, Chat Circles 2 was relatively mature and very stable. Chat Circles 2 underwent several months of live testing by real users on the Web, and it had also been prepared for a public exhibition, in which it would have had to run virtually without interruption for at least 3 months. As a result, Chatscape was stable and usable from day one, making it easy to test new features and modules.

Chatscape employs a database on the server side to hold user account information and user properties. So far, Chatscape is the only project that uses a database; all other SMGServer-driven projects do not use persistent user accounts. The database is a SQL database, using an industry standard ODBC database connection. This type of database connection should allow any standard SQL database with ODBC drivers to be connected to Chatscape. However, so far only a Microsoft Access database has been used. I had the option to use Sociable Media's Oracle server, which is used for several other unrelated projects. Despite this availability, I decided to use Access for its ease of configuration and its portability; since the database consists only of one file, it can be copied anywhere and used anywhere. This allowed me to work at home or at the Media Lab. Oracle requires a network connection on a local area network, which would have prevented me from working at home. The only drawback to

using Access is the 256-character maximum size of data fields, which can be very limiting for some properties.

# 10. Future Directions

There are many ways that Chatscape and SMGServer can be extended in the future. Due to Chatscape's flexibility, Chatscape enhancements could follow along the same lines of research as described in this document, or investigate completely different topics of research.

New identity variables and aspects of the visual representation could be introduced. Right now, users are represented by simple shapes that can be distorted and warped in several different ways, but more can be done without leaving the realm of abstract graphics. I would like to see extensions and attachments to the visual representation that represent different aspects of identity, much like clothing or jewelry, but in a more abstract sense. However, care must be taken to ensure that the visual representations do not become overly complex.

I would also like to see more interactive behaviors, especially behaviors that act upon patterns of past events stored in the history. A great deal of social interaction involves references to history, or is placed in a context of knowledge of prior events. Online interactions should be able to reflect this as well.

Finally, more can be done in the area of long-term system effects, especially in terms of "status" and "ranking" of different users. This is related to the reputation systems described earlier, except in this case, the system itself would be able to apply some judgment upon users. Active members of an online community, or particularly interesting members of the community should have those facts automatically reflected in their visual appearance.

# 11. Conclusion

Chatscape has shown that behaviors can make a chat room more interesting, yet more complex at the same time. It has definitely become a good platform for future work involving behaviors and their application in chat. Out of the set of behaviors that was implemented for this project, the reputation-based behaviors are the most interesting. A casual yet comprehensive reputation system in place in a chat environment could make a chatroom not only more informative for its users, but more enjoyable as well.

Through the work done on Chatscape, Chat Circles, and TeleAction, SMGServer has become a very important part of Sociable Media's software toolkit. I believe that it has achieved all of the goals it set out to achieve, including scalability, reliability, and performance. The feedback gained from Dana Spiegel, my own experiences, and feedback from users of the system serve to bolster this position. A forthcoming document will fully explain the technical procedure of developing chat servers around SMGServer.

# 12. References

**[Viegas 99]** Viegas, F. and J. Donath. 1999. “Chat Circles”. In *Proceedings of CHI 99*

**[Vilhjálmsson 98]** Vilhjálmsson, H.; Cassell, J. 1998. "BodyChat: Autonomous Communicative Behaviors in Avatars." *Proceedings of ACM Autonomous Agents '98*, Minneapolis, May 9-13, p.269-276.

**[Kurlander 96]** Kurlander, D., T. Skelly and D. Salesin. 1996. “Comic Chat”. In *Proceedings of ACM SIGGRAPH*.

**[Rodenstein 00]** Rodenstein, R. 2000. “Talking in Circles:
Representing Place and Situation In an Online Social Environment”. M.S. Thesis, MIT Media Laboratory

**[TeleActor 01]** Donath, J.;Spiegel, D.;Lee, M.;Dobson, K.;Goldberg, K. 2001. “Tele-Direction: A New Framework for Collaborative Telepresence”. In *Proceedings of CHI 2001*.

**[Web 01]** eBay, http://www.ebay.com; Epinions, http://www.epinions.com

**[Maxis 00]** 2000. “The Sims”. http://www.thesims.com

**[Carroll 91]** Carroll, John M. 1991. “Designing Interaction”. Cambridge, MA. MIT Press.

**[Oikarinen 93]** Oikarinen, J.; Reed, D. 1993. “RFC 1459: Internet Relay Chat Protocol”. Located at http://www.ietf.org/rfc/rfc1459.txt

**[Apple 00]** Apple Computer, Inc. 2000. “Darwin Streaming Server – FAQ”. Located at http://www.publicsource.apple.com/projects/streaming/faq.html

**[Giel 00]** Giel, B. 2000. “JavaServ”. Located at http://www.kcmultimedia.com/javaserv/