Activity Server: You can run but you can't hide

Sanjay Manandhar

MIT Media Laboratory
Cambridge, MA 02139 USA
sanjay@media-lab.mit.edu

Abstract

The activity server is a software utility that provides high-level information on what a user or a number of users are doing by combining data inputs from various sources. These sources are: 1) the *finger server*, 2) the *phone server*, and 3) the *location server*. The finger server provides information on users' activities or lack thereof on hosts within a local network; the phone server abstracts information from call progress of users' digital phones whereas the location server abstracts information from active badge sightings. Responses from all these unobtrusive data gathering sources are put together to form a composite model of the activities of users. Information may be collected only from or on behalf of participating users; each user within the user community can determine how much of his own information the activity server may receive. The main attributes of the activity server are: 1) it maintains history of all the participating users, 2) it synchronizes among the many, possibly conflicting, pieces of information from three sources, 3) it distinguishes interdependence of activities among users, and 4) it provides a high-level abstraction about users' activities.

1 Introduction

People are creatures of habit. The variety of activities a person is involved in over the course of a work day is limited. Furthermore, these activities may recur at some frequency from day to day. Some earlier work goes as far as to say that everyday activity is wholly routine [1]. Although a random slice of human activity is not deterministic, recurring activities in an office environment can be modeled informally. A server that will help build a model of activities in an office environment is proposed. This server, the "activity server," and the various auxiliary clients will be described in later sections.

The activity server solicits information from three other servers, the finger server, the phone server and the location server. (For clarity the latter three servers are called Listeners and unless explicit, the "server" shall refer to the activity server whereas the "clients" shall refer to the clients of the activity server). The state maintained by the activity server helps build a dynamic model of office activities so that the cumulative results and histories of activities may provide useful information to the user himself and other members of his work place.

2 Related Work

Work in traditional artificial intelligence (AI) has concentrated on plan recognition where it is assumed that there is a precise plan that a user activity will follow [4,7]. Many researchers working in interdisciplinary fields have proposed models varying in scope, complexity, and computational formality. For instance, in a technical paper from Xerox PARC, many dynamic models at micro and macro levels were provided [3]. Some newer work argues against planned models since it is very difficult to allow for unpredictable and unanticipated circumstances [1,2,9]. Although not a user modeling effort, the activity server uses a dynamic model that can infer the state of users and adapt continuously. Most of the user modeling work in AI and cognitive science research, including dialog systems [6], expert systems and student modeling have focused on systems that monitor user activities, and react to them. The activity server only monitors and builds an activity model; however, clients of the activity server that benefit from the user model can execute (react) accordingly. In addition, unlike many user modeling efforts that attempts to enhance user-system interaction or understanding, the activity server focuses on augmenting user-user interaction with the help and coordination of many discrete systems.

In the following sections the motivation behind building a system like the activity server and its overall architecture will be discussed. Subsequently, the internals of the Listeners as well as that of the activity server will be presented, followed by a discussion section.

3 Motivation

The activity server is designed with a view to improve interaction of co-workers, given that richer forms of communications like meeting in person, talking over the telephone and electronic mail, etc., are not always possible. The activity server is intended to answer questions like,"Can I have a meeting now with colleagues A, B and C?" or "What is the breakdown of time spent on meetings today?" or "Where is colleague X?" or "Who came by my office while I was away?" Such everyday questions constantly arise in an office environment and require active participation from the solicitor of the information. It is highly desirable to have instant answers or at least intelligent inferences to such questions.

An Example

Let's take a simple example from above: Where is colleague X? To answer this question, the activity server queries its three Listeners. The following are possible answers that it receives:

> **Phone server:** X is not on the (default) phone.
> **Location server:** Badge X is in its (default) office; it has been there for the last 10 hours.
> **Finger Server:** Idle time on the machine in her office is zero minutes (connected to terminal d0).

The activity server draws up conclusions about the location of colleague X from each Listener response and assigns a confidence level to each. From the phone server alone, it is clear that X is not in her office. From the response of the location server, X is, in fact, in her office but has not moved for 10 hours. The latter piece of information undermines the plausible conclusion that she is in her office. These are conflicting conclusions. But the finger server responds that the idle time on her (default) machine is nil, i.e., she is active. However, she is not active on the console (located in her office), but rather on terminal d0. Since all dN (N is an integer value) terminals are open for dialin access, colleague X has dialed in. Hence, the activity server will return with the conclusion that colleague X is not around, she has dialed in remotely. From the example above, some of the key attributes that motivated the activity server can be illustrated:

1) *History*: The server maintains history on its data. This is important in order to be able to infer the user's activities from his past actions.

2) *Multiple sources*: The server will draw upon its Listeners to get multiple, possibly conflicting, views on real world activities.

3) *Multi-party*: The activity server monitors many users simultaneously. Every member of the group may affect the state of other members of the group.

4) *High-level abstraction*: The activity server can assemble many pieces of discrete information, which by themselves are of little value but cumulatively allow abstraction of intelligent, high-level inferences of the users' activities.

4 Overview of the system

As mentioned earlier, the activity server adopts a client-server model. However, the activity server itself is a client to three other servers, the finger server, the location server and the phone server, which provide asynchronous events that help build a model of user activities.

Events from the Listeners are funneled through a common event handler which does some housekeeping such as timestamping and executing database lookups (see Figure 1). The most important of the housekeeping duties, however, is the updating of internal state. Further, a set of rules is applied to resolve conflicting Listener conclusions or emphasize conclusions in agreement. Hence, this module can be considered an arbiter and manager of the various Listeners. The high level inferences made by the rules are saved and are made avail

able to the the clients via the client library.

5 The Listeners

Listeners are the indispensable information-gathering servers that the activity server relies on. They are independent and service their own pool of clients; the activity server is a special client because it connects to all the Listeners simultaneously. Each Listener is self-sufficient and is oblivious of other Listeners; only the activity server, which collects data from all the Listeners, has the global view which allows it to make more complex inferences that span more than any one Listener's domain. What follows is a more complete description of the Listeners.

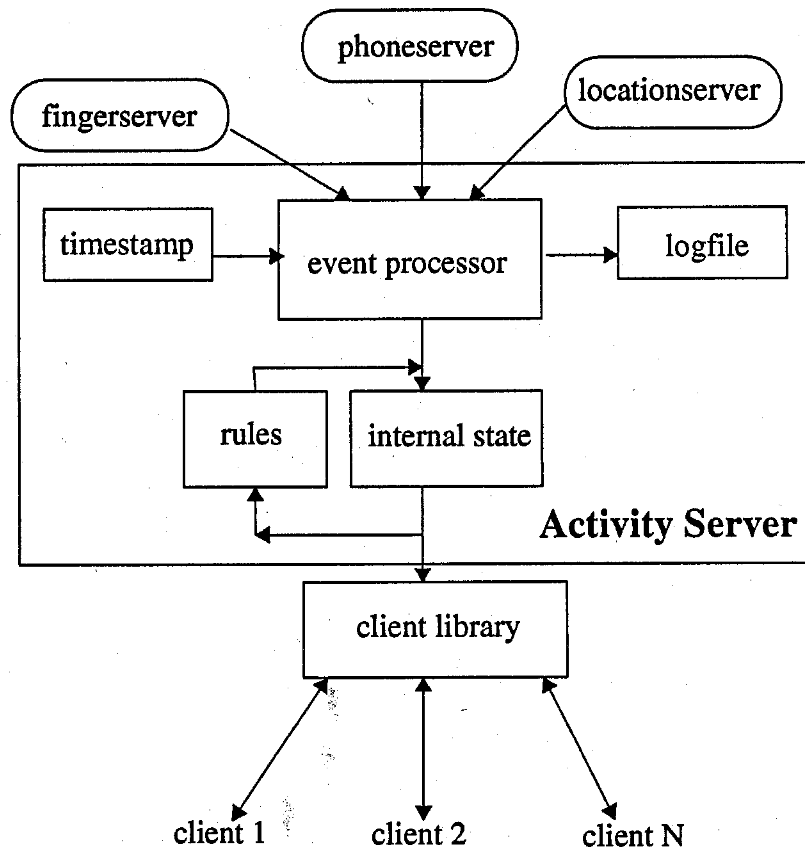

Figure 1. Overall system diagram of the activity server

5.1 The Finger Server

The finger server gathers idle times of users on a number of machines. This server depends on the finger daemon that runs on many machines. The finger daemon keeps state (idle times in minutes) of all users on that machine. The finger server queries finger daemons and collects and maintains information on many machines (Figure 2). This allows the finger server to compare information across the entire group of machines it is monitoring. For instance, if a user is logged in on more than one machine, it can tell where the user is most active, whether he is logged in remotely and from where he is logged in. In addition, the finger server can send asynchronous events (called alerts) to clients that express interest in them.

5.1.1 Architecture

The architecture of the finger server is based on the data objects it maintains. The finger server consists of doubly linked lists of three data objects (hosts, users and clients). The rationale behind the linked list is so that hosts, people, and clients are all dynamic data that may be added and deleted with ease. In addition, the basic linked list operations are the same for all the objects. Since the finger server is designed to keep state on a limited number of hosts, users and clients, data representation using linked lists is adequate; for a larger object domain, other optimal forms of data representation (e.g., hash tables) could be used.

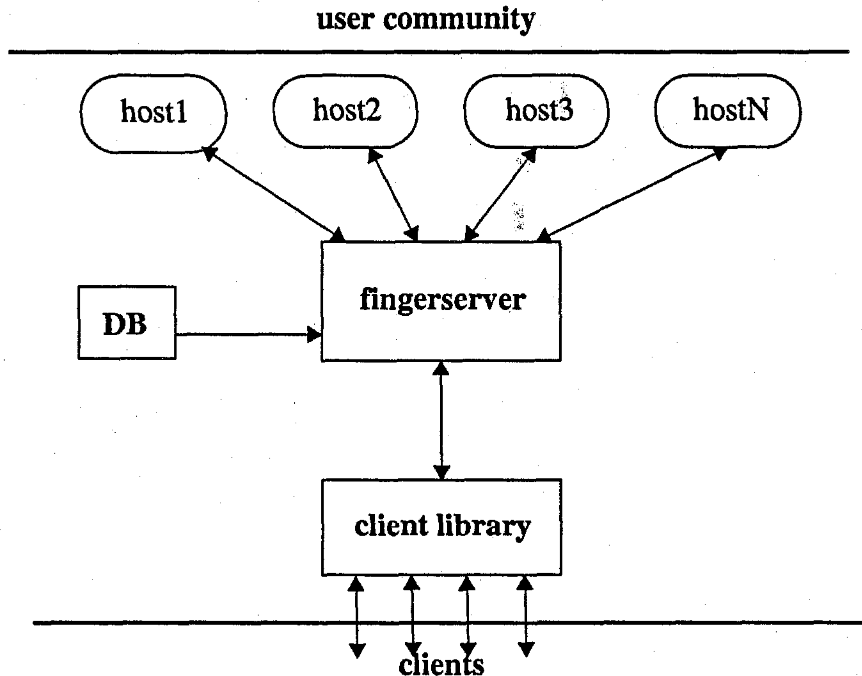

Figure 2. General architecture of the finger server

5.1.2 Mechanism

The finger server can be considered a client to the finger daemons while it is the server to its own clients. The finger server polls the list of machines that it keeps track of at certain intervals. It gets the list of machines and users to keep track of from a pre-defined text file which is checked periodically to see if has been updated. The results of the pollings are saved in the three linked lists mentioned above; queries from clients are serviced by accessing data from these linked lists.

5.1.3 Polling

A timer is registered so that the finger server can poll every time the the timer goes off. Typically the timer is set for 1.5 minutes. Note that the granularity of the finger daemon is in minutes so it is not useful to set the timer for less than a minute. Some machines may be polled less frequently if so desired. For instance, machines with many users or machines at a remote site may trade up-to-the-minute updating for better system performance.

It is imperative that the polling rely not only on the timer but also on the internal state of the polling session. It is unnecessary to query for finger information from a particular host a second time if the preceding poll has

not returned. If a socket to the finger daemon is opened at each polling interval, machines that take more than the poll cycle time to return from the poll may have, over time, multiple sockets open simultaneously.

5.1.4 Read Callbacks

At the time of polling, if the finger socket to a machine is currently not open, it is opened and a read callback is registered; the polling then moves on to the next machine - the server does not wait and block to read from a socket. Hence, the receipt of data is fully asynchronous. When data is available at a socket, its descriptor disambiguates the machine. The data is then read and the finger daemon closes the socket. Although it is a streams connection, the finger daemon keeps the socket open only as long as is necessary; that is, it is not possible to reserve a socket communication line for future polling.

5.1.5 Finger Information

Information returned by the finger daemon is machine-specific. So far, machines that use Unix, Ultrix or Genera (an OS for Symbolics Lisp machines) are supported. Typically the information received will have at least the username, hostname and the idle time; most Unix systems also report the terminal. All these pieces of data update the user and host data structures. After each host reports, its corresponding host data object and its internal linked list of users are updated and any changes are immediately reported to the clients that requested asynchronous notification. Reporting changes to the user data structure needs to wait until most machines have returned from the poll so that comparison of the most recent activity can be made and reported.

While idle times indicate the user's activity or lack thereof, the terminal name is useful in locating the user's physical location. If the user is connected to the console ("co") terminal of a machine and has a small or zero idle time, it is evident that no matter where else the user is logged on, she is physically located where the console of that machine is. On the other hand, if the user is on "dN" (N is an integer number) terminal, it is clear that the user has dialed in.

5.1.6 Protocols

There are two levels of finger server protocol. The lower level protocol is at the byte stream level; the second, a C interface, based on the byte stream level, is also available. Once a socket connection to the server is established, simple, case-insensitive, ASCII commands may be sent via the connection. The server uses the same connection to send its replies. This human-readable interface allows for understanding of commands and responses with a minimum experience with the system. A brief mode can be set so that non-human clients may get the important data easily. Figure 3 shows a sample session with the finger server using telnet. (User commands are shown in italics). The first command, help, gives a flavor for the kinds of functionalities that the finger server supports.

(toll: /u2/sanjay/due) telnet toll fingerserver
Connected to toll. Escape character is '^]'.
help
The text based commands and arguments are case insensitive.

?	Same as HELP
ASYNC	Commence asynchronous transmisions
BYE	Exit from the server
GET-HOST-INFO (host)	Get info on all users on HOST
GET-ALL-HOST-INFO	Get info on all users on all hosts
HELP	Print this help file
LIST-HOSTS	Get the names of all hosts the server knows about
LIST-PEOPLE	Get the names of all users the server knows about
LOCATE (person)	Find out where PERSON was logged on
LOCATE-ALL	Find out where everyone is logged on
QUIT	Same as BYE
RESET	Revert to default state (reset modes and requests)

SET-MODE (SHORT I LONG)	Set short or long output format
STATUS	Print out your personal requests and modes
SYNC	Stop asynchronous transmitions
USER-ON-HOST (user) (host)	Find out if USER is on HOST
TRACK (person)	Request asynchronous reporting on PERSON
TRACK-ALL	Request asynchronous reporting on all users and hosts
TRACK-ALL-HOSTS	Request asynchronous reporting on all hosts
TRACK-ALL-PEOPLE	Request asynchronous reporting on all users
TRACK-HOST (host)	Request asynchronous reporting on HOST
UNTRACK (person)	Cancel asynchronous reporting on PERSON
UNTRACK-ALL	Cancel asynchronous reporting on all users and hosts
UNTRACK-ALL-HOSTS	Cancel asynchronous reporting on all hosts
UNTRACK-ALL-PEOPLE	Cancel asynchronous reporting on all users
UNTRACK-HOST (host)	Cancel asynchronous reporting on HOST
VERSION	Current version of the finger server

locate barons
[PERSON barons] on [HOST leggett, TTY co] with [IDLE 3 min] at 03:21:02 PM

set-mode short
OKAY

get-host-info leggett
barons leggett co 6 03:23:21 PM

bye
Goodbye.
Connection closed by foreign host.

Figure 3. A sample session with the finger server

5.1.7 Alerts

Asynchronous reporting of changes in user or host status are of four kinds: login, logout, dormant and idle. All user data objects have their parent pointer pointing to the host object . After the poll to a host returns, the finger server updates all the user objects on that host and timestamps every object that is updated. In addition, all user objects maintain the idle and old idle times after the current and previous updates, respectively (reset value for both is -1). Given these premises, the algorithms that mark state transitions are relatively simple (see Figure 4). Note that IDLE_THRESHOLD is the ceiling of idle time up to which a user is considered active at a terminal. In the current implementation this user-defined value is 5 minutes.

State	Transition conditions	
LOGOUT:	**if**	user_timestamp > host_timestamp
LOGIN:	**if**	user_idle_old == -1
	and	user_idle > -1
DORMANT:	**if**	user_idle_old > IDLE_THRESHOLD
	and	user_idle > IDLE_THRESHOLD
IDLE:	**if**	user_idle_old > IDLE_THRESHOLD
	and	user_idle > IDLE_THRESHOLD

Figure 4. Simple algorithms for alerts

5.1.8 Shortcomings

There are some shortcomings which are inherent in the finger daemon while others were introduced by the finger server itself.

1. It is possible not to "see" quick logins and logouts on a machine. Since the polling cycle executes every few minutes, a login-logout pair on a machine by a particular user may not be reported by the finger daemon. Likewise, if a user exceeds the IDLE_THRESHOLD for a few seconds but then hits a key, this DORMANT state of up to 59 seconds is not reported. The granularity of the finger daemon is in minutes up to 59 minutes, then in hours and minutes up to 9 hours and 59 minutes, then in hours up to 23 hours and then in days. On the other hand more precise granularity may not be truly useful.

2. Activity in some programs are not noticeable to the finger daemon. For instance, activity solely in the gnuemacs editor program will increment the idle time as if the user was away from the terminal.

3. The finger daemon reports only the idle time, it does not report what program may be running. There are other Unix programs and daemons that monitor execution of programs at any particular time but their services were not used for a number of reasons.

- Not all machines run these daemons (rwho, w, rusers, etc.) but almost all machines keep the finger daemons running. (Some sites disable even the finger daemon for security reasons). Hence, finger-server can remain very general and modest in its requirements.
- Many of the other daemons run by mutual broadcasts and receives. This can put a severe burden on network and machine performance.
- Idle time history alone can be fairly useful.

4. One of the shortcomings of the finger server is that there is a latency in asynchronous reporting. Should there be a race condition between two hosts reporting back to the finger server, the updates of the second host on the queue will be delayed by the time it takes the first host to dump all its data and for the finger server to update all its data structures. Typically the latency is in the order of a few seconds. However, for machines with many users logged on, the reporting of everyone's information, and its subsequent parsing can take tens of seconds.

5.1.9 Optimizations

Querying the finger daemon to find a limited number of users' idle times is a very costly operation. It hurts network and machine performance. The finger daemon checks the /etc/hosts file and figures out usernames, real names, etc. Possible optimizations are: 1) provide options to the finger daemon that will return only the desired information. Unfortunately these options are not available on all machine types. If latency becomes critical, different requests can be sent to different machine types depending on what they support. 2) A modified finger daemon can be implemented that will cache much of the routine information. 3) A less frequent polling with interpolations between pollings (if necessary) can alleviate latencies.

5.2 The Phone Server

The phone server is another Listener that the activity server relies on to get phone activities. It monitors ISDN protocol that is exchanged between the 5ESS ISDN switch and the telephone sets to get the events of all telephone sets within its jurisdiction (see Figure 5). Much phone state information such as onhook, offhook, incoming call, dialing, held, etc. can be received from the phone server. For the purposes of the activity server, however, only onhook and offhook information are critical; to a lesser extent, the incoming call (caller number) may be useful if the calling and the called parties are both among the users the activity server is interested in.

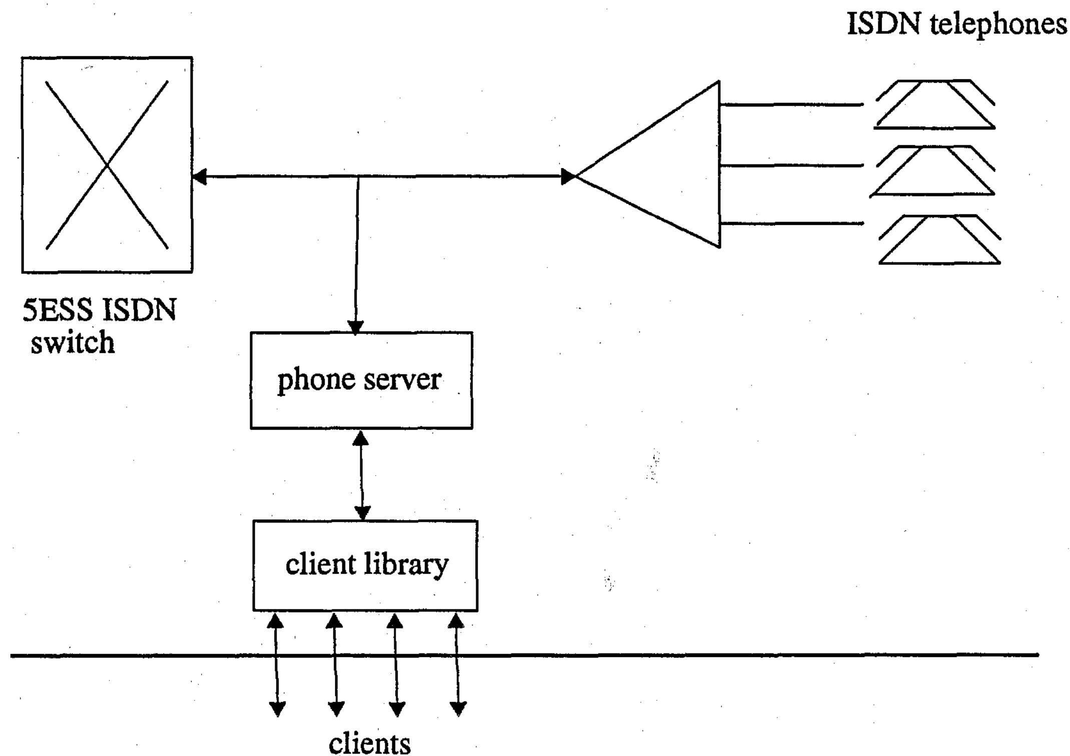

Figure 5. Architecture of the phone server

5.3 The Location Server

Another Listener, the location server, consists of a network of infra-red sensors that are placed around the building, in hallways, offices, lab areas and conference rooms. Participating members of the user community may choose to wear a badge, called the *active badge*, which will transmit and receive infra-red packets. Since each active badge has a unique code, this code and the owner of the badge can be mapped in a database.

When a user wearing his badge (designed and built by Olivetti Research Laboratory in Cambridge, England), moves around in the building, and the badge is sighted by sensors, the sensors register the unique ID of the badge and report it to the sensor server. The sensor server translates the identity of the reporting sensor to its location (e.g. East Hallway, Conference room, Lab area or Office 355, etc.) and translates the ID of the badge to its owner, thus ascertaining the location of the user (see Figure 6). The sensors are polled every 5 seconds to check badge sightings. By the time the events propagate to the activity server, there is a latency on the order of a few seconds.

6 The Activity Server

The activity server inspects the abstracted data from the Listeners and looks for overlapping activities. It filters events from the Listeners and updates its internal data structures. There is a rule-based inference engine [11] that operates on the incoming events to arrive at a high-level abstraction of the activities of the user community.

6.1 Representation

The activity server needs to keep state on a number of real world entities such as servers (Listeners), ma-

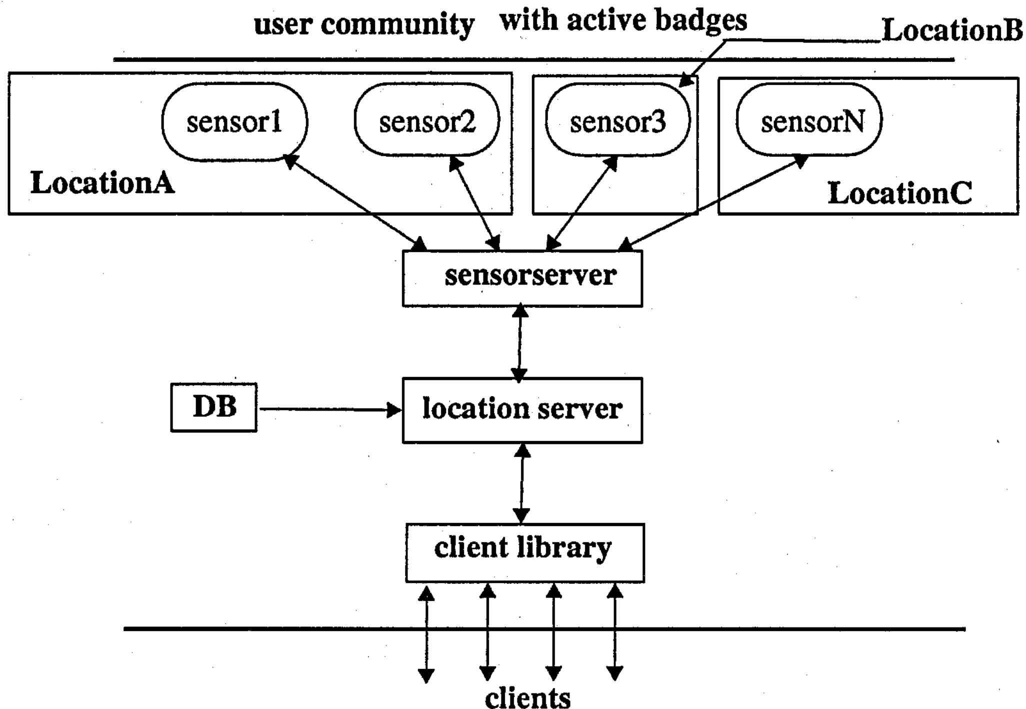

Figure 6. Architecture of the location server

chines, places and people. These entities are modeled as data objects which are described below.

6.1.1 Person object

The person object consists of a person's name which is used as an addressing token, their office location, their office phone and a timestamp. The assertions of the inference engine pertaining to this user are saved as current state and the degree of fit (given by confidence levels between 0 and 100) into the model of busy, alone or not around is saved. The head and tail pointers of the list of all significant events pertaining to the person are also saved.

6.1.2 Place object

The place object consists of the name, for addressing and informational purposes, and the update timestamp. The head and tail pointers of a list consisting of all occupants who entered (and may have left) will be maintained. Head and tail pointers of the list of all occupants currently in the room will also be maintained. By definition, users who do not have an exit timestamp are current occupants. Therefore, the list of current occupants is really a list of pointers to a subset of all occupants.

6.1.3 Server object

The server object maintains the name, the server state (up or down) and the last down time.

6.2 Mechanism

6.2.1 Startup

When the activity server starts up there are a number of initializations. It must open a socket in the advertised port to add the service; clients may then connect to the server via this port. It saves the old log file that saves system activity and creates a new one. Routine information, such the help file that will be sent to clients upon

request, and phone number, office number, usernames associated with the user community is cached. An interrupt handler is registered so that when an interrupt signal is received, the server can close sockets, update the log file and make a graceful exit.

Next, connection to all the Listeners are attempted. If the connection is successful, the initial state information is solicited. The finger server will return the least idle time and hosts of all users; the location server will return the location of all users and the phone server will return the current phone state of office phones of all users. Once the initial state information is received, each of the Listeners are given commands to report asynchronous events. Should the attempt to connect to the Listeners fail, a timer is set so that a reconnection can be attempted when the timer expires.

6.2.2 Event handling

All events, synchronous and asynchronous, pass through the event handler. The event handler is able to distinguish which Listener is reporting by inspecting the socket the information is received on; it unbundles the data accordingly and updates the data structures pertaining to that particular Listener. The event is discarded if the unbundled data reveals that the activity server does not care about that data.

The event handler timestamps all events, even though all the Listeners return their data with a timestamp. This is necessary because system times vary; a single point of timestamping on a single machine will provide a standard reference point to compare temporal information of events.

6.3 Rules

Some rules are necessary to draw higher level conclusions and to resolve conflicting data from the various Listeners or accentuate data that is in agreement across more than one Listener. Each event triggers some number of rules. Each rule has one or more conditions and one or more assertions each of which will have a confidence level associated with it. It is also possible to assign weights to some of the conditions to affect the confidence level. For instance, for someone who uses the phone rarely, the rare use of the phone may be very significant. Hence, the weights on the phone events can be increased; this will affect the decision making in building the activity model.

6.4 History Mechanism

As mentioned earlier, the person object saves all the events that affect it and the place object saves enter and exit times of all occupants. There must be a mechanism of pruning this list, however, otherwise the lists could extend beyond control. Hence, a window of interest that extends from the present time to a duration in the past is introduced. This window of interest can be set to be different for different Listeners. Events occurring beyond the window of interest are removed from the lists that form the internal state.

In addition, some rules in the inference engine can help compact the amount of state that is maintained. For instance, if a user is seen in location A and is "seen" at UNKNOWN location and is again seen at location A, the inference engine will check other Listeners and consider the second event an anomaly, i.e., the active badge was not visible to any sensor, and will discard the event and the changes it brought about.

7 Clients

There are a number of clients that rely on the activity server, and their use will test the information embedded in the activity server. Some of the clients are described below.

The *Directory Client* gives information about a particular person or a group as a whole. For example, the server can return the users' activity on the phone, the workstation or location around the building; it will also return one of the three activity classifications with its estimate of confidence in the assertion: busy, free or not around. For the busy category, the reason (on the phone, another person visiting the office, meeting in the conference room, or visiting another office) is also provided. This client generates simple text in English which can, in turn, be piped to other hardware such as a Dectalk text-to-speech synthesizer so that synthesized information can be received over analog phone lines from a remote site.

The *History Client* gives a summary of the activity within a reasonable duration of time (e.g., last hour, today, etc.) in textual format. When the user queries, this client provides breakdown of time spent on various activities.

The *Watcher Client* is a graphical interface to a client that updates the activities of participating members of the group; this client is an enhancement of a previous effort [10]. Watcher is an X Window System application that displays bitmapped images identifying other users. The display is updated whenever any user changes state on the phone, in physical location or on the network of workstations. Many tracking and messaging facilities are also available. Figure 7 shows how a lineup of users can be displayed (the users get to choose their own cartoon characters or a bitmap of their own pictures); the lower window shows how the user information may be displayed.

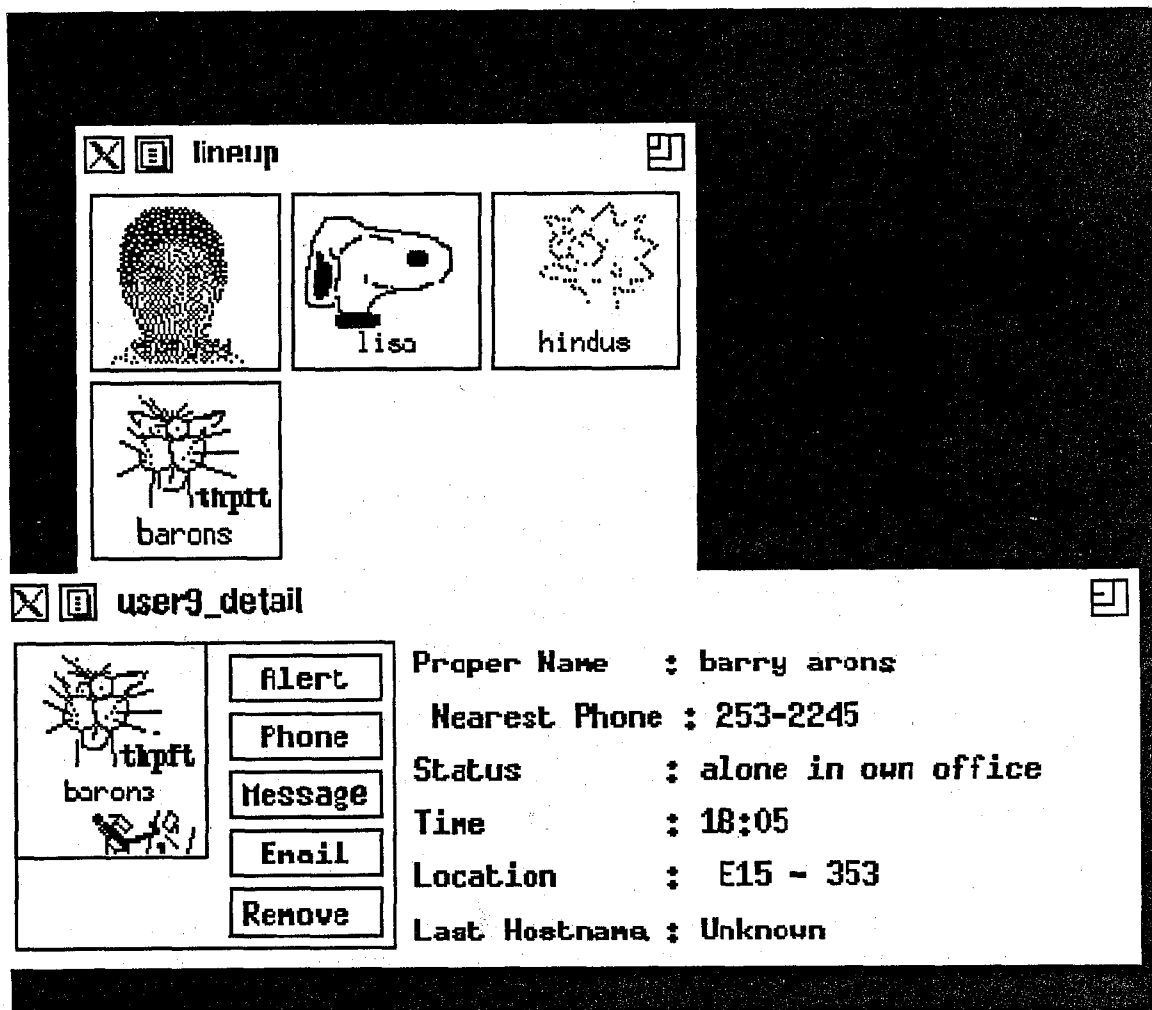

Figure 7. An example of Watcher

8 Discussion

A system like the activity server might uncover privacy issues. As emphasized earlier, a tool such as the activity server can be used very effectively in small groups of trusting individuals (e.g., project teams). However, the amount of information that the server receives can be controlled by the user himself using a customization directive alterable only by that individual. Therefore, if a user does not want others to get telephone information, for instance, the activity server will shift the emphasis to other Listeners. Having the user decide the inputs into his own model is not new; some user modeling systems have allowed the user to "edit" the model to enhance the model [5]. This "editing" concept can be used to enhance the model and also preserve user pri-

vacy. In addition, in future implementations, each of the Listeners and/or the activity server may require a password so that the information is provided only to trusted clients.

The reliability of the activity server may be undermined by many factors. The foremost factor is the varying latency in receipt of events from the Listeners. For instance, the phone server reports with minimum latency while the location server has some latency. The finger server may have variable latency depending on net traffic and the kinds of machines polled. Hence, for very fine-grained, up-to-the-second activity reports the activity server would not be suitable. Secondly, the reliability depends, to a large extent, on external factors such as whether or not the users are wearing their active badges. For instance, if a user is at a meeting in the conference room and is not wearing her active badge, the finger server is the only Listener that is useful. If the mouse on the machine that the user is logged on is moved slightly for any reason, the finger server will report an active terminal. The inference, the wrong one in actuality, would be that the user is active at that machine. Barring these constraints, the activity server can provide enough information to make reasonable activity classifications.

The principal contribution of a system like the activity server is the mechanism it provides a user community to coordinate each other's office activities. This can ease scheduling problems, reduce phone tags, and, in general, induce more productive office work. To this end, the activity server consults a number of unobtrusive Listeners, constructs an informal model and draws conclusions. Although much user modeling work relies upon a single source of information, such as user-input, eye tracker information, the activity server benefits from three different sources of information. One of the sources, the finger server, was implemented expressly with the activity server in mind, but it can have many uses independent of the activity server, as well.

There are several areas in which the work started by the activity server can progress. The first area is an on-line calendar, which although more static over the course of a day, is certainly very dynamic over the course of many days or weeks. It would be possible to modify the Unix calendar so that its granularity would be an hour or half hour slots rather than a day. The only drawback would be that, unlike other Listeners, the calendar would require direct involvement from the user (i.e., update the calendar regularly). Unless the user is conscientious about updating, the events that the calendar Listener can provide will be too sparse for them to be very useful in modeling the activities. Nevertheless, for projections on user activities in the future, the activity server might benefit from an added Listener like the calendar server. Secondly, the actual utility of the activity server can be gauged by a usability study. It would be interesting to find out whether or not people use the activity server actively. Investigating informed users' apprehensions or lack thereof towards a system that monitors unobtrusively might shed light on the actual utility of systems like the activity server. Thirdly, many potendial clients can benefit from the activity server. Some earlier work has concentrated on messaging services after locating the user on a local-area network [8]; the Watcher Client provides some messaging service as well but this capability can be greatly augmented. Finally, the information provided by the Listeners can be building blocks to a more rigorous user modeling.

9 Conclusion

The activity server is yet another tool that can be used in an office environment. It is unique, however, because it uses multiple information gathering techniques which provide redundant and possibly conflicting information to construct an informal model of user activities. It is hoped that the knowledge embedded in the activity server can benefit small trusting groups of users in the office environment.

10 Acknowledgements

Chi Wong wrote the majority of the phone server code and Derek Atkins along with Jim Davis wrote most the location server. Steve Tufty wrote the Watcher client. The author would like to thank Barry Arons, Debby Hindus and Mike Hawley for making constructive criticisms to earlier drafts of this paper. And finally, Chris Schmandt deserves many thanks for posing the problem and giving many helpful suggestions.

11 References

[1] Philip E. Agre. The Dynamic Structure of Everyday Life. MIT Artificial Intelligence Laboratory, Technical Report 1085, 1988.

[2] Philip E. Agre and David Chapman. What are plans for? MIT Artificial Intelligence Laboratory, AI Memo 1050, 1988.

[3] Clarence A. Ellis. Formal and informal models of office activity. Technical document. Xerox PARC, 1984.

[4] Henry Kautz. A circumscriptive theory of plan recognition. Intentions in Communications (Eds. Phillip R. Cohen, Jerry Morgan and Martha E. Pollack). MIT Press, Cambridge, Mass. 1990.

[5] Judy Kay. *um*: A toolkit for user modelling. *Second International Workshop on User Modeling.* March 30-April 1, 1990.

[6] A. Kobsa & W. Wahlster (Eds.). User Models in Dialog systems. New York. Springer Verlag. 1989.

[7] Martha E. Pollack. Plans as complex mental attitudes. Intentions in Communications (Eds. Phillip R. Cohen, Jerry Morgan and Martha E. Pollack). MIT Press, Cambridge, Mass. 1990.

[8] L.F.G. Soares, S. L. Martins, T.L.P. Bastos, N.R. Ribeiro and R.C.S. Cordeiro. LAN Based Real Time Audio-Data Systems. *Proceedings of ACM SIGOIS '90 Conference on Office Information Systems.* 1990. pp. 152-157.

[9] L. Suchman. Plans and situated actions: The problem of human-machine communication. Cambridge University Press, New York, 1987.

[10] Steven Tufty. Watcher. MIT Bachelor's Thesis. 1990.

[11] Patrick H. Winston. Artificial Intelligence. Addison-Wesley Publishing Company, Reading, MA, 1984.

Sanjay Manandhar recently received his MS from Massachusetts Institute of Technology. As a grauduate student and a research assistant with the Speech Research Group at the MIT Media Lab, he worked on incorporating a speech recognition system to the X Window System and on audio drivers and desktop audio tools. His current interests include desktop audio, storage and transport of mixed media objects and broadband networks.

Manandhar has a BS in electrical engineering from Massachusetts Institute of Technology.