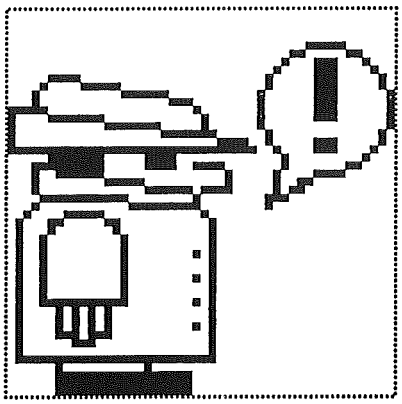
Social Presence on The Internet

Ian Bosley
Honours Project Report 1994



Contents

List of Figures

Chapter 1

Introduction

> The affirmation of the group should not be a sublimation of the individual but a framework for involvement. A good gathering requires participation– the effects of organisation, work and attendance–and in turn gives back sustenance for the body and soul, a sense of belonging, and the accomplishment of something that could not be done by the individuals alone.
>
> (Ming-Dao, 1992)

The importance of social relationships within collaborative partnerships is well known, and generally accepted. Substantial contributions towards group achievements are made through informal communications. Research into the patterns of collaboration in scientific research has shown that maintaining good personal relationships is fundamental to effective cooperation.

How do we maintain social relationships with our colleagues and workmates? Research suggests that they are maintained largely through casual interactions. These interactions are often random and depend upon physical proximity and high quality communication.

Modern advances in telecommunication and networks have made the mediation of distance between distant partners easier and more effective. Widespread use of large networks such as the Internet has made remote colleagues more accessible.

Currently the Internet connects millions of users. The traffic over this network has increased substantially in recent years and the rate of growth is continuing to rise. Goodman *et al* (1994) reported that the Internet numbered 2,217,000 hosts in January 1994, a 69% increase from the previous year.

Contacting distant neighbours through the Internet is not always a trivial task. Users need to know user login names and Internet addresses. Often they are required to use unhelpful command line utilities. Casual interaction is made difficult through the time and resources needed to make and maintain a connection more distant than across a local area network.

Technology for supporting social interaction and spontaneous communication for distant partners is an active field of research. Applications in this area

lean strongly in the direction of glamorous high technology solutions: video and multimedia communications.

Hollan and Stornetta (1992) argue that this approach is questionable in value. Systems have been developed as solutions for problems of collaboration between colleagues. Research into the patterns of use for these solutions demonstrate that most users use the system to maintain a simple social awareness of the activity of others.

The purpose of this project is research methods of providing a low level awareness of the presence and activity of colleagues. This functionality should be independent of geographic proximity, without requiring specialist hardware or large quantities of network bandwidth. This awareness is provided with the intent of promoting social interaction in collaborative partnerships.

SPI, a system applying these principles has been implemented using common Unix utilities as building blocks. The design of the SPI system, and the implications for further research will be discussed.

Chapter 2

Motivation and Related Work

This project draws together several fields of Computer Supported Cooperative Work (CSCW) and Groupware research. The primary motivation comes from research into social presence, which typically involves attempts to directly support communication between remote sites. The implementation techniques are drawn from information filtering research, and from active or "intelligent" mail systems.

In the following sections the related CSCW work is detailed.. Section 2.1 examines the research into patterns of interactions between collaborating colleagues, and the problems faced by partners who are separated by a physical distance. Section 2.2 discusses the systems that have been designed to address these problems using high technology. Section 2.3 reviews these systems and draws lessons for future social presence systems. The remaining sections describe low technology approaches to the provision of social presence.

2.1 Patterns of Collaboration

Much research has gone into assessing the types of interactions that lead to effective collaboration, especially in scientific research. A landmark paper in this field was given at the CSCW'88 Conference in Portland, Oregon by Kraut *et al*, (1988a). The conclusions of the paper were reiterated by the same authors in (Kraut *et al.*, 1988b). Briefly summarised, they argued the following:

1. Maintaining social relationships is fundamental to performing collaborative work.

2. Informal and casual interaction is fundamental to maintaining social relationships.

3. Physical proximity is fundamental to effective interaction, both spontaneous and planned.

They argue that the place of technology in this field is to mediate the effects of distance, so that collaboration partners that are physically separate can interact together as well as partners who are physically close, (Kraut *et al.*,

1993). For this kind of technology solution to be effective it is necessary to address these factors of proximity and informal communication.

1. **High frequency of communication.**

   Informal contact generally results from frequent opportunities for communication, and often leads to subsequent collaboration. People who see each often are more likely to have an effective personal relationship than those who rarely see each other.

2. **High quality of communication.**

   Rich visual and aural communication channels must be available. The communication media must be multi-way and multi-sensory.

3. **Low cost of communication**

   Cost of communication can be discussed in terms of financial expense, and effort expended. Long distance often involves significant personal cost. This is not a feature of proximal communication.

The geographic distance need not be great to have an effect on these factors. Even colleagues working in the same building can have trouble keeping track of each other, if they are on different floors.

Individually, casual interactions appear to acheive little purpose. In the aggregate, they are fundamental to the execution of work, the transmission of organisational culture, and the maintainance of member loyalty and morale.

Visual contact is an important part of social interaction. Mutual visual contact is often a social obligation to interact. The ability to identify both the potential partner and their availability is very important for casual interaction. People visually assess the locus of attention of potential partners and use this to determine how interaction may be initiated, if at all.. Visual contact is often a stimulus for topics of conversation.

Informal interaction important for both production and social work. Often the two have common goals, and the difference is only a matter of degree. Not only does work get done, but participants obviously enjoy each other's company.

2.2 Social Presence Systems

Much of the research in this area centres around the use of technology-oriented solutions The field is dominated by video and multimedia communication channels.

Three of the more notable systems are reviewed in the following sections: Video Wall, CRUISER and Portholes. Each has some valuable lessons for the SPI system.

The observations of Kraut *et al* and the availability of high bandwidth communication channels provided both the motivation and the technology for a new generation of groupware applications. Through the use of video and audio network links, these systems focus on mediating the effects of distance in collaborative effort.

2.2.1 The Video Wall, a.k.a. MEDIA SPACE

An early attempt at promoting "virtual proximity" was performed the Systems Concepts Laboratory at Xerox PARC in Portland and in Palo Alto, (Goodman & Abel, 1987). The researchers considered that interactions like conversations in a hallway are essential to group cohesiveness, and therefore to successful collaboration.

They set up a video link between the two establishments. The link could be switched to different locations at both sites, but the most common setting was a large "Commons" area. This area attempted to give the impression of looking through a large window into the other site. The video and audio communication link gave the impression of a shared environment, or a "media space."

(Bly *et al.*, 1993) traces the roots of this research more thoroughly. They also identify examples of use for the media space.

- Peripheral awareness of activity at the other site.
- Unintentional chance encounters, often leading to valuable conversations.
- Intentional attempts to locate colleagues at the other site.
- Video phone conversations between colleagues.
- Group discussions, both spontaneous and scheduled.
- Recording, retrieving and replaying video records of meetings in the media space, or "video memos"
- Experimentation and envisioning exercises for projects related to distributed group work.
- Video presentations that are potentially organisation-wide.
- Shared social activities, such as joint Christmas parties.

The Xerox PARC media space is reported to be a successful attempt at maintaining a group identity despite the geographic division of its members. Participants with private video connections would often open a connection to the common areas, and leave it open while they worked. This allowed them to maintain a social awareness regardless of physical remoteness, (Dourish & Bly, 1993).

2.2.2 Social Browsing with CRUISER

The CRUISER system at Bellcore and its antecedents have been the subject of a wealth of publications. The system began as an attempt to stimulate the informal interaction between co-located people using desktop video telephony, (Root, 1988; Fish *et al.*, 1993; Cool *et al.*, 1992).

The subjects of the initial experiment were housed in the same building and collaboration partners were often on the same floor, so the CRUISER system was simply a supplement to existing interaction protocols. CRUISER was controlled

through software and provided a 12-inch colour video monitor with speaker and microphone to each user.

The system provided three forms of call initiation: *Glances, Cruises* and *Auto-Cruises.* A Cruise was a three second connection to a specified colleague. The call could be extended if either party issued a 'visit' command. If a contact was not specified the system provided a series of random Cruise connections.

A Glance was similar to a Cruise, except that it only lasted one second before timing out. Either party could extend the link through the 'visit' command.

Auto-cruises were system-initiated random connections. These arguably invasive occurrences were an attempt to model random meetings in the hallways. This is a good example of taking a natural process out of context. In a hallway meeting there are several stages of initiation, such as eye contact, and facial expression. One rarely interrupts a colleague when they are already engaged in a conversation or some other activity. The auto-cruise excludes these protocols, and hence hinders its own acceptance.

The aim of the system was to prompt users to 'browse' a virtual environment looking for casual or spontaneousy interaction. They noted two behavioural innovations that were enabled by the system. Firstly, users would set up long duration connections to create a *virtual shared office.* Participants would work on their own, occasionally making a remark or initiating a conversation with the other person. Secondly, users would *waylay* other members of the team. For example, a user opens a video connection to a colleague. If the colleague was not there then the initiator would leave the connection open. They would peripherally monitor the other office while continuing their own work. Surprisingly, the initiator would usually physically go to the associates office once the colleague returned, rather than use the video link to interact.

Analysis of the patterns of use of the CRUISER system demonstrated that people generally used the system in a similar manner to the telephone, or simply to gain an awareness of the availability of colleagues. Users seemed to use the system to negotiate and set up physical meetings rather than in lieu of them, (Hollan & Stornetta, 1992).

The auto-cruise was an abject failure. From 236 system-initiated calls, only 3% were accepted, (Fish *et al.*, 1993). Besides that, 40% of the users mentioned the auto-cruise when asked what they disliked about CRUISER.

2.2.3 Portholes

The Portholes system at Xerox EuroPARC was motivated by an observation of the trials of the Palo Alto-Portland Xerox Media Space, (Goodman & Abel, 1987). It was observed that many users use media space technology simply to maintain a general social awareness, (Dourish & Bly, 1993). Users liked to observe public areas, and watch members of their group interact. They would engage long-term connections and leave them in "background" while they carried out their usual activity.

The aim was to provide the group awareness across distributed sites using less bandwidth than live video. They used video cameras to take still images of each participant's office. These images are distributed through the Internet

and displayed in a window on each user's terminal. The images are updated every few minutes. An enhanced version of the system allowed users to record audio "snippets" that could be played by observers[1].

Initial observations were generally positive; there did appear to be an increased awareness between the two sites as well as within them. However, specific problems were noted.

1. Questionable reliability of the images due to the latency in updating the display – can the images be "trusted?"

2. The window display took up so much screen real estate that it was commonly iconised. Reopening the display incurred too much overhead for the system to be generally useful.

3. Very little actually happened in the display. Images were only updated every so often.

Each of these limited the usefulness of the application, and hindered the motivation for users to accept the system. The positive reports of the system may have been in part an artifact of the novelty value.

The system did demonstrate that a social awareness across distance is a meaningful concept. Distant colleagues are still at a disadvantage to co-located ones, however, due to system limitations.

2.3 Criticisms and Lessons

Each of the above examples features a common factor. In each, the researchers have attempted to model the interactions of co-located colleagues by using high-bandwidth specialised video hardware.

A certain degree of success was noted in each, especially from the perspective of group identity and social presence. Participants in the experiments were able to use the system to remain aware of the activity of other members of their group.

However, this style of problem solving has limitations. None of the systems are easily extendible to a more global network. Network demands for video connections, and the availability and cost of the hardware make it difficult to generalise over arbitrary sites.

Hollan and Stornetta (1992) also criticise social presence research. They believe the focus on "virtual proximity," a sense of "being there" that accurately mimics face-to-face communication, is misplaced. Since technical imitations will never match the real world, they believe such systems will not achieve success until users will choose the system over face-to-face communication, even when the latter is possible. Their extreme[2] argument is that groupware applications

[1] Recording snatches of favourite pieces of music was a popular option. One participant even sang "Happy Birthday" to himself and recorded it as an audio snippet.

[2] In their paper, Hollan and Stornetta acknowledge that they are adopting an extreme stance in order to better champion their philosophy

should go beyond standard human communication, so that the systems will be used even when face-to-face commmunication is possible.

Behind all the approaches is a common aim: to promote and support social interaction between colleagues, both co-located and distributed. Communication channels exist that adequately support interaction between remote users. The problem lies in encouraging spontaneous and informal use of these channels.

People who are physically close are motivated to interact with their colleagues by chance encounters and a maintained visual and audial awareness of their environment. However, one of the interaction partners must consciously initiate a conversation.

CRUISER attempted to increase the frequency of unplanned communication through the AutoCruise mechanism. This dropped both participants into an unplanned communication, which either or both might find intrusive.

The VideoWall common room increased casual interactions between the two sites, by allowing colleagues to 'bump' into each other as they passed by the video monitor.

For the system to be widely useful in a global sense, it must be easily extendible to include partners at arbitrary geographic locations. Obviously there are limitations imposed by network availability. However, it is possible to obtain a wide accessability by utilising lowest common denominator tools.

2.4 A Low Level Approach

At the simplest level, all that is needed is a awareness of the activity level and location of co-workers.. A peripheral awareness of this information could give the impression of a virtual proximity. A more active system could use this information to promote contact by offering advice about and access to appropriate communication mechanisms.

The simplest support for social browsing on a network just provides a list of users who are logged in to a specific machine, (Cockburn & Greenberg, 1993). Knowing that someone has logged in to a computer does not necessarily guarantee that they are available. However, the user's idle time can be used as a basis for heuristic inferences about their level of activity, from 'typing right now' through to 'hasn't logged in since Christmas.'

In Unix systems, this 'whereabouts' information is provided through commands such as `rwho`, `rusers`, and `finger`. Such commands, especially in the Unix environment, are typically difficult to use. They are characterised by arcane syntax, and knowledge of Internet addresses must be at the user's fingertips. In general they are tedious to use, and provide little guidance to the user. Interpreting the results of these commands can be equally difficult.

An example of Unix 'unfriendliness' is the idle time format for some implementations of `finger`. A sample of this given in figure 2.1. The idle is given here in the fourth column. An empty entry denotes a zero idle time – that is, the user is currently typing. A number followed by a colon is the number of hours the user has been idle. A number on its own or following a colon is the number of idle minutes. A number followed by a 'd' is the number of days the

```
Login       Name                TTY Idle    When    Where
geoffrey Geoff Thomas            cu       Mon 11:22  remote
dbain    David Bainbridge        p0  11: Mon 12:49  xcol6:0.0
vanmm22  van Mierlo Marcel       p1  19d Tue 18:09  cctrxe:0.0
ian      The Dark Paw            p4   57 Wed 11:36  xsun419:0.0
phil     Philip Saysell          pb 1:28 Mon 10:34  xsun408:0.0
```

Figure 2.1: Example `finger` information showing an esoteric idle time format

account has been idle.

The idle time format is not constant. In some cases, it will contain the number of seconds the account has laid idle as well. How is a user of `finger` to know whether the display includes seconds? The colon in the column display provides no cue. Users may interpret the colon to separate minutes and seconds, rather than hours from minutes.

Little help is provided to the users of these systems to aid them in either acquiring or interpreting the information.

2.4.1 The TELEFREEK System

The goal of groupware systems in the social presence domain must be reduce the guidance requirements placed on the users, and to minimise the cognitive work required to make use of the information supplied.

This aim is the basis of work done by Cockburn (1993) on a system called TELEFREEK, which supported a low-level of social browsing in a local network. The system provided awareness cues using common Unix tools, and users could initiate different communication links through different channels. TELEFREEK did not directly support communication but transparently provided access to them.

As further work, the paper suggests expanding the social presence facilities to the Internet using "active" electronic mail messages. (Cockburn, 1993; Cockburn & Greenberg, 1993). This addresses to global extensibility problems mentioned earlier.

A mechanism for an awareness monitor based on email as a transaction medium is described, and will be discussed in Chapter 3. In the next section the growing appreciation of electronic mail as an effective medium for supporting cooperative work is discussed.

2.5 E-mail as a Social Presence Solution

"The only successful CSCW application has been email"

Robert Kraut, (Grudin, 1994), p. 95

The use of electronic mail as a communication media is widespread. Goodman *et al* (1994) reports that around 150 countries have at least email connectivity to the Internet. Compared to video or voice communications, email requires minimal network bandwidth. Its simple text format lends itself to communication in a broad variety of forms, in manners that are readily machine-processable.

One of its main attractive features has been the informality permissable between all levels of an organisation when using email, (Licklider & Vezza, 1988). This has increased the personal acceptance of email, a factor that is vital to the success of any groupware system, (Grudin, 1994; Cockburn, 1993)

Email is known to be low on the spectrum of groupware technologies. It is asynchronous and provides little in the way of a shared environment, and yet many people find themselves using rapid exchanges of email to communicate almost synchronously (Ellis *et al.*, 1991). Email can now be used in lieu of `ftp` for file transfers by sending appropriate messages to automatic 'list servers.' The same servers are also useful for subscription to electronic mailing lists.

From a CSCW perspective the availability and flexibility of email greatly enhance the role of computers in communication. Across the Internet, access to email is essentially universal. The ability to add custom information to the message header, and the unformatted nature of the message body allow users to create their own structural rules.

The widespread adoption of email as a means of communication and collaboration has lead to an information deluge, (Malone *et al.*, 1989). Users are beginning to experience an increase in unsolicited, so-called "junk mail", and malicious attacks known as "spamming."[3]. There is increased motivation for some form of automated control of mail messages. There is increased motivation, therefore, for semi-autonomous mail filtering.

In turn, this increases motivation for Malone's semi-structured messages. (Baecker, 1993). Malone (1988) sets out a set of reasons why mail messages of an expected structure can be useful in computer-supported coordination.

1. They allow automatic processing of a wide range of information
2. It is possible to communicate non-routine data without a rigid structure
3. Semi-structured message processing is already used by people informally
4. Structure rules are useful to authors as well as to receivers of messages
5. They can simplify the design process of systems that are expected to be incrementally enhanced.

Semi-structured messages call for messages of identifiable types containing a known set of fields. The contents of the fields may be unstructured text or

[3]According to Internet folklore, this term comes from a comic vignette by British comedy group Monty Python's Flying Circus. Every time a character in the skit said the word "spam," a group of vikings would take the word up as a song, louder and louder, until the conversation was drowned. Email attacks similarly deluge a user's mailbox or a Usenet newsgroup with spurious messages.

other information. A set of rules is used by receivers to decide what to do with the message.

Malone (1989) proposes the Information Lens as an attempt to address these issues. The second-generation of this system is called Object Lens, and is discussed in (Lai *et al.*, 1988). A key figure of these systems is the high degree of user involvement in deciding on the rules for semi-autonomous agents.

Users may select and edit templates for mail messages, and specify actions for these messages. Incoming mail that matches any of the user templates is then treated accordingly. For example, certain authors may be deleted before reaching the user mailbox. Mail on a particular subject may be moved to a user folder automatically.

More justification for email as a CSCW platform is given by literature supporting the use of 'active' mail messages. Hogg (1985) proposes *imail* as active object mail messages, which are processed autonomously at the receiver and the results returned or forwarded for further processing. Hogg's implementation used a script language, in which user's would write queries.

Another attempt to use email for groupware applications is the Active Mail framework proposed in (Goldberg *et al.*, 1992). This system uses enhanced electronic mail to support communication and cooperation with other users.

2.6 Conclusions

The informal interaction and social activity has been shown to be valuable for collaboration. The two key issues in informal interaction have been identified as communication quality and spatial propinquity.

Casual interaction is characterised by high quality communication. In social presence system this is determined by the cost of interaction. Interaction cost is the effort required once communication has been established.

"Spatial propinquity," or physical nearness, is considered vital for social interaction as it presents opportunities for unplanned communication and reduces the costs. Equally so, visual contact is considered very important, especially for initiating a communication session. These concepts apply to social presence systems in terms of the effort required to establish a communication link.

The value of mediating distance is clear. Social presence systems aim to use telecommunication technology to reduce the problems faced in both establishing a connection, and the cost of interaction. If it is possible to present the same opportunities for social interaction to people physically distant as for those physically close, then a wider and potentially more effective range of research relationships are possible.

Several attempts have been made using video technology, and high bandwidth networks. These have reported limited success, but suffer from problems in availability and generalisation to a wide area network.

Because many communication channels are already available, perhaps our focus might shift to simply conveying a sense of presence in a global community. This should be possible using currently available network connections and common network utilities.

If a passive monitor of group activity is linked with an active interface to communication channels, then the cost of informal interactions is reduced. The utility which presents the information about a colleague becomes also the utility for initiating communication with them.

Such an application would be very similar to Dourish's Portholes system at EuroPARC, but would not require expensive hardware. Instead of video snapshots, the user is given an assessment of activity based on computer statistics. If Internet email is used to communicate information, then the system becomes more broadly available.

If such a system could gain user acceptance, perhaps it would become useful in promoting more frequent interaction between distant associates, leading to more effective and enjoyable collaborative relationships.

Chapter 3

Design of a Social Presence Monitor

In Chapter 2, the background to social presence systems was discussed. The SPI system is an example of such an application. The system provides users with a monitor of the presence and activity of their associates.

3.1 Functional Requirements

The target user population for this monitor system are people with access to Internet host computers. These computers are expected to form an integral part of their daily activities.

For this design, the availability of a Unix connection to the Internet is assumed. Once the general principles are proven however, it is possible to adapt SPI to other operating systems.

- **The system must provide information about the activity of the user's colleagues.**

  The system allows the user to specify a known set of colleagues. If those colleagues are willing to participate, the system will periodically check their activity on their computer system.

  This information is to used to promote social interaction. Implicitly required is maintenance of a database of contact information, similar to an address book.

- **Lowest common denominator tools are to be used for acquiring information and for communication.**

  The system uses common Unix and Internet utilities. From the user's perspective, it hides the complexity in both executing and interpreting these commands from the user.

  Awareness is extended to a global level by using email and email filtering, (discussed in Section 2.5). Extensibility is something not present in high-tech solutions. Adding a new member to the social group requires

specialised hardware and the availability of high bandwidth network connections. By using low level tools, SPI becomes widely applicable.

- **The system is required to ease the effort on the part of the user by performing some interpretation of the activity information.**

  The system uses the gathered data to make a judgement about the likely presence and availability of the colleague, decreasing the cognitive effort required of the user.

- **The information is to be displayed in a manner conducive to peripheral awareness.**

  This is presented to the user in a peripheral manner. The user can then maintain an awareness of their virtual community with little or no conscious effort.

The user of the SPI system can then use this information to select the appropriate communication medium, perhaps with assistance from the system itself. Building knowledge about available channels into the system will allow much of the effort of making a connection to be automated.

3.2 Assessing Activity Levels

The simplest support for social browsing on a network provides a list of users who are logged in, (section 2.4). This information is available through low level Unix utilities. Commonly, users can determine the whether a user is logged in at a particular machine. If they are logged in, it is possible to find out their idle time and the status of their mailbox.

The layout of this information varies from system to system. By convention, however, it is reasonably consistent. SPI can take advantage of the semi-structured nature of utility reports to extract what information is available using heuristic inferences.

3.3 Acquiring the Data Globally

The possibility of using e-mail as the transaction medium was suggested specifically in Cockburn (1993), page 162, as further work for the TELEFREEK system. The mechanism relies on an ability to run commands in someone.else's account at a remote site.

Once permission has been granted by the owner of the account, the user can then send email messages containing shell scripts to that account. These messages are identified by an additional header field. An email scanner detects these request messages and, once the sender and the contents have been authorised, executes their contents. The results of the execution are then mailed to the original sender, again identified by an additional header.

The issue of security is very important here, and will be discussed more thoroughly in later sections. For many local networks, electronic mail and

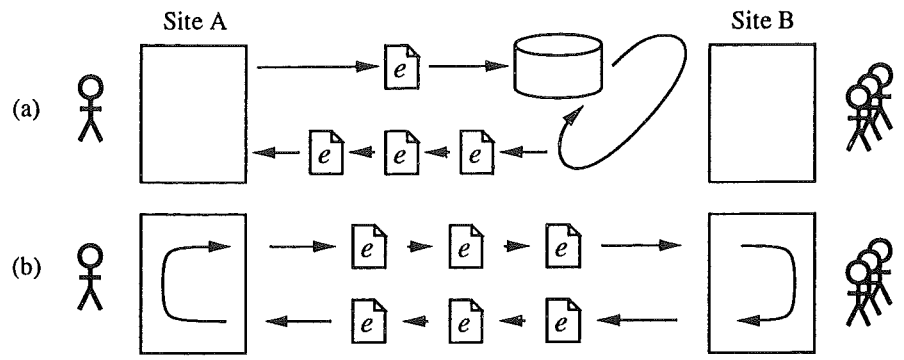


Figure 3.1: Mail communication mechanisms. (a) The client-server model. (b) The request-response method.

finger are part of a security 'perimeter.' The mechanism used by SPI provides a path through this perimeter, and consequently must be handled with great care.

There are two approaches to using this mechanism for awareness maintenance, shown in figure 3.1. The first approach is to have a SPI server at each participating site. Clients who wish to monitor a user or group of users at a site register themselves with the server. The server is then responsible for periodically performing the commands and sending the results to the client. The second approach is easier to implement. In it, the client is responsible for sending a request message whenever it requires an update. The remote site starts an instance of the processor whenever it receives a request, and each instance terminates immediately after returning the results.

The second method increases the mail traffic overhead. There are two mail messages for each update: a request and a response. In the first method, there is only one: a response, with a one-off registration request.

However, the first method suffers from robustness problems. If either the client or the server crashes at some point in the process, the other site will remain unaware. A solution to this is implement a global expiration period for registration. All clients and servers use the same period. Servers stop sending responses after this time, unless the client sends a renewal request.

The first method also increases the complexity of the system, and requires an additional server process to be running even when the site is not being polled. For these reasons, the current SPI implementation uses the second 'request-response' method.

3.4 User Interface Issues

The purpose of the SPI system is to provide a peripheral awareness of the activity of associates and colleagues. The user interface must be designed to provide this awareness, as well as more directed enquiries about a particular user or

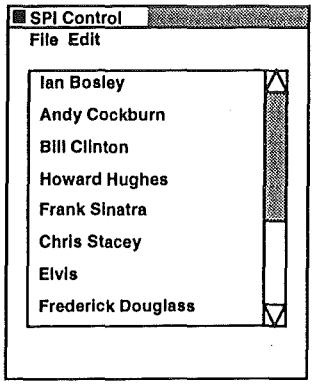


Figure 3.2: A minimal control interface, showing a selection list of colleagues

group of users.

In keeping with the abstraction of complicated low level tools, the system should provide as much guidance as possible to the user. At the same time it should minimise its dependency on guidance from the user.

The user interface can be divided into two logical sections: the user control interface, and the awareness monitor display. In terms of visual arrangement these entities should be separate entities: that is, the user must be able to display either, both or neither at will. When the user is working with one of the units, it does necessitate viewing the other.

3.4.1 The User Control Interface

The control interface is responsible for allowing the user to control the actions and configuration of the SPI system. At a minimum, it must allow the user to select contacts from a database of associates. A diagram of the control interface is shown in figure 3.2.

3.4.2 The Awareness Monitor Display

The monitor display is largely modelled on the Portholes system developed at EuroPARC. This system is discussed more throughly in Section 2.2.3. In summary, the display consisted of images of video stills that were taken at intervals and stored in a server database.

SPI design follows a similar layout for the display. User control features are separated into a separate interface entity. The awareness monitor allocates a pane of the display window to each contact to be polled. The pane displays the name of the contact and the activity information

One lesson to be learned from Portholes is the use of display 'real estate.' Some users of the Portholes system found that the display was too inert to keep open all the time. Very little happened in the display, and yet the video images took up considerable room on the screen. Consequently, they often 'minimised' the application. The overhead involved in re-opening the display

was greater than the perceived benefits. Once the display was closed, however, the peripheral awareness benefits were lost.

User acceptance depends on the display being visually interesting and useful, without grabbing disproportionate quantities of screen space. SPI is fortunate in that the size of each pane can be quite small. As for visual interest, that depends on providing an effective display techniques for the activity level information.

Display Techniques

Some consideration must be given to how the activity information is displayed. Information can be assessed by the system at several levels of granularity. At the lowest end of the scale is a simple numeric representation of the idle time. This allows the user to make their own assessment about the activity, but requires more conscious effort.

The aim of the system is to provide a *relative* indication of activity levels.

A slightly higher level might use dials or slider meters. This becomes easier for the user assess at a glance, and the scale of the meter encodes some information about reasonable idle times. The scale could be a user configurable parameter to the system. Thus if the user knows that his or her colleagues use their computers infrequently, then the scale should allow for large idle times.

A decision would have to be made as to what the dial displays: activity level or idle time. For activity level, a meter reading of zero means the account has been idle for a long time. For idle time, a meter reading of zero means the account is currently active. Whichever decision is made, a cue to the user must be displayed.

Another method emulates load monitors by using a bar chart of idle times. This is still at a low level of granularity. The idle time information is displayed 'raw,' but a historical record is kept. This can be useful to the user in assessing whether a colleague is working intermittently, or in sustained bursts.

These approaches are aimed more at the technologically inclined. Some users may find the numeric styles of display impersonal. SPI is attempting to model an awareness of colleagues to promote social relationships. This suggests a less mechanistic approach may be more suitable.

One such approach that is is still low on the above scale is use a picture or representation of a person, and shade it according to the level of presence. Users who are more idle than others appear to 'fade' out of view, whereas very active users appear solid. This method requires a scale for shading the image. Should it be linear? If so, what should the gradient of the line be? The suitability of these selections varies from one work situation to another. The settings must be configurable by the end-user. Examples of the above three methods are shown in figure 3.3.

SPI can also make assessments of it own based on idle times. The system may decide that users who are idle for up to five minutes are likely to be working on their computers. If they are idle for up to thirty minutes then they are probably working at their desks, but not directly on the computer. Longer idle times, up to a limit of several hours, may mean that the user is around today but not currently in their office. Idle times beyond several hours probably belong to

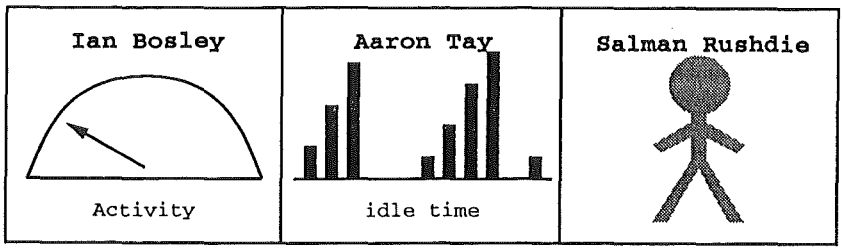


Figure 3.3: Examples of activity display methods. From left to right: A dial meter, a bar chart record, and a shaded figure representation.

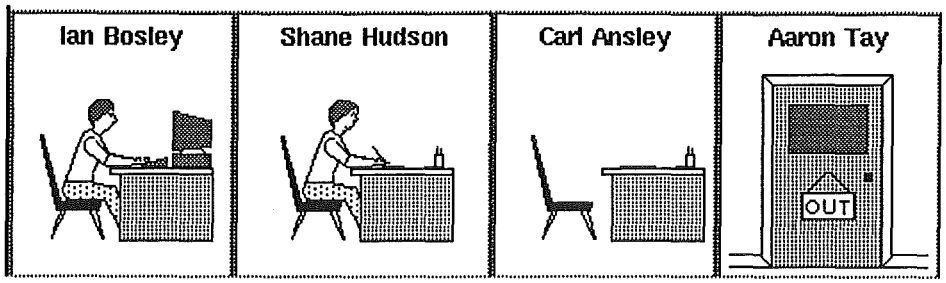


Figure 3.4: Examples of the iconic activity display method. Activity level decreases from left to right.

accounts that have not been logged off, even after the user has left the site.

The criteria used here are examples only. Longer or shorter times may be more suitable for different collaboration partnerships. This is an area strongly open to user specification and guidance. However, the principle is clear. If the system can use specified levels for assessing the activity and location of colleagues, then it can display this assessment instead of the raw data. Users can then maintain awareness simply by glancing at the monitor display.

Display methods that are appropriate for this information are needed. The above example has four levels of activity. The first level is for colleagues who are currently working. The representation of this might a picture of someone working on a computer. The next level might be someone writing with a pen. A picture of an empty desk might be used for users who do not appear to be in their office. The last level is the same for people with long inactive accounts or are not logged in. For this, a closed office door is a possible portrayal. Examples of this iconic style of display are shown in figure 3.4

Another interesting possibility is the use of facial expressions and emotion. (Maes, 1994) describes using facial caricatures to convey the state of semi-autonomous agents. These agents "look over the shoulder" of users while they perform daily routines. An example is reading electronic mail. The agent records information about each message and the actions the user performs on

it. Then, as the user works, the agent makes predictions. Depending on the confidence of the predictions, the agent displays a different facial expression: confused, pleased, surprised, and others.

Facial expressions can be used to represent different activity levels. A caricature of someone concentrating, perhaps with a pencil gripped in their teeth conveys the impression of active work. A face with closed eyes, an open mouth and a line of Z's emerging from it might represent an idle or logged out account.

Chapter 4

Implementation of the SPI System

The implementation of SPI was performed incrementally. First the local awareness principles were proven using low level Unix tools. Next, the email mechanism was implemented to provide global awareness.

SPI was implemented in John Ousterhout's tcl/tk language. This language is simple to use to create attractive user interfaces, yet sufficiently powerful data manipulation tools to perform the text processing. Because the language is interpreted, it is easy to develop applications in an exploratory manner.

4.1 Choosing the low level tools

SPI was intended to use as much of the available Unix utilities as possible. It was possible to use these tools to perform three of the basic functions.

4.1.1 `finger`

SPI uses `finger` to perform the activity checks. It is widely available, and can provide a wealth of information for very little effort. Also, the format of its output is highly predictable.

The system is easily adapted to use other presence tools instead of or in conjunction with `finger`. The heuristics used by the post-processor would have to be modified to accept the new input.

`finger` achieved notable attention as one of the utilities used by Morris' Internet Worm. This is discussed in more detail in Chapter 5, Security Concerns.

4.1.2 `sendmail`

`sendmail` is the basic interface to SMTP[1] Internet mail. To quote from the `man` pages:

[1]Simple Mail Transfer Protocol

> sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; sendmail is used only to deliver pre-formatted messages.

A loophole in the operation of `sendmail` was the other attack used by the Internet Worm. The hazards involved in the user of `sendmail` are discussed in more detail in Chapter 5, Security Concerns.

Access to the `sendmail` program itself is usually tightly controlled, because of the security risks it presents. The SPI implementation uses a higher level interface program, called `send`. This program is part of the `MH` mail handler suite of utilities. It pre-processes mail messages before handing them on to another `MH` utility, `post`. It is `post` which deals ultimately with the SMTP transport system.

4.1.3 `procmail`

This utility is a very useful mail filtering tool. It uses 'recipes' written by users to process incoming email. Recipes consist of a list of criteria and actions. If a message matches all the criteria, then the actions are performed upon it.

Criteria perform regular-expression pattern matching via an internal `egrep`. Actions can involve appending the message to a mail folder, piping it into a program's standard input or forwarding it to other users. `procmail` is typical of Unix commands in that the format of the recipe file is complex, difficult to learn and does not gracefully forgive mistakes.

SPI uses `procmail` for handling the email transactions that extend the monitor to a global level. Effectively, SPI forms a limited user interface to `procmail`. The user of SPI is not required to directly manipulate `procmail` or its recipe file. Indeed, the user may remain completely unaware of its operation, even while it works to their benefit.

`procmail` is used at both ends of the communication mechanism. The remote request processor uses `procmail` to identify incoming requests. The local monitor use `procmail` to intercept returning activity information messages.

4.2 Modular Design Approach

To aid incremental development and to promote a "plug-and-play" approach, SPI is implemented in a modular fashion. There are four components to the complete awareness monitor, as shown in Figure 4.1. At a local level, the mail interfaces and the remote processing modules are not used.

The advantages of this approach are obvious. It means that different styles of display can easily be tried out, or different heuristics for assessing the activity of a contact can be used without changing more than a module at a time. It also means that an individual module can be tested in isolation, by writing 'stub' procedures to fill in for the other modules.

The communication strategy for the modules is shown in Fig 4.2. Modules typically run as separate processes and communicate through intermediate files, or by sending tcl/tk inter-process messages.

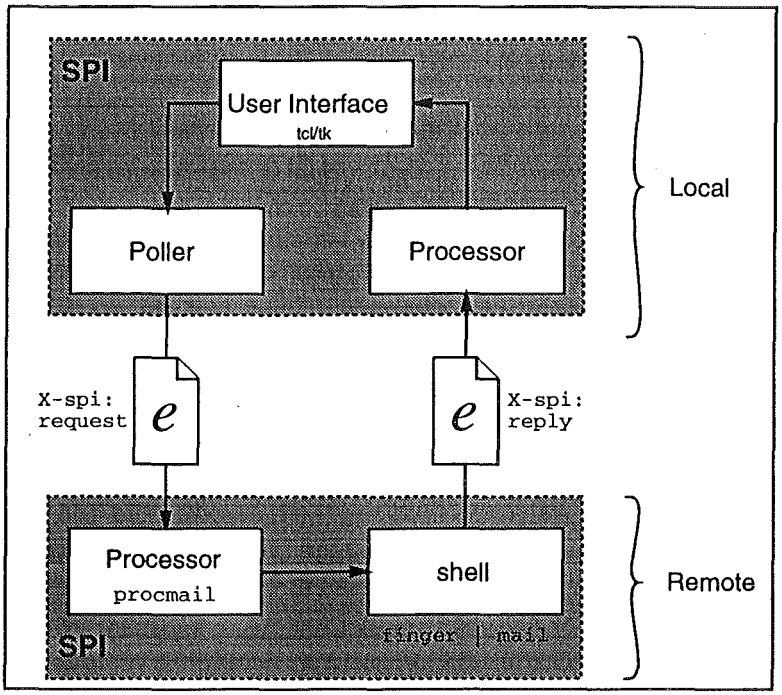


Figure 4.1: Modular components of the SPI system design

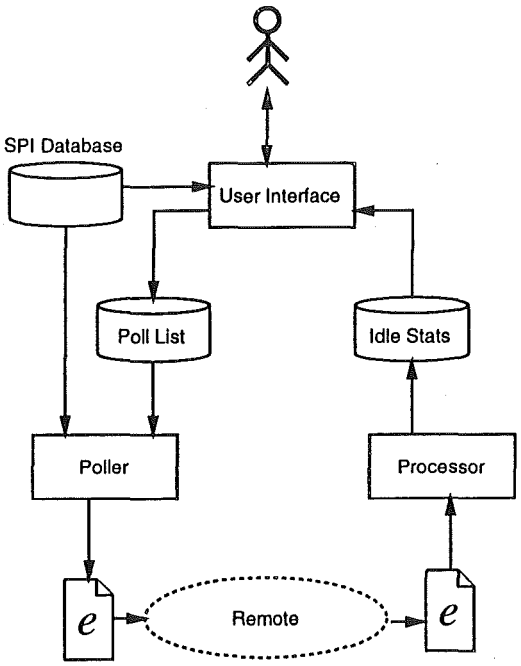


Figure 4.2: Communication between the active modules of the SPI system, using intermediate files.

4.3 The User Interface

The user interface module has the following primary roles:

- Responsible for displaying the activity status of each contact to the user.
- Allowing the user to configure the display.
- Maintains the user contact database, from which the user may select colleagues to poll.

Additional functions include automatic maintenance of the `procmail` recipe file, and allowing the user to edit the `.plan` file.

4.3.1 The Control Interface

The initial implementation of the control interface was simply a list of the database contacts surrounded by button widgets. Control of the activity monitor required selecting the contact, and then selecting an action. Actions included adding the name to the poll list, removing them from the poll list, editing and viewing the contact details. Other buttons were for editing the SPI configuration and terminating the SPI program.

It was noted that there was no indication of which names were in the poll list in the control panel display. For the user to identify whether or not a user was being monitored required mapping from the list to the Activity Display and back again. When the user made a selection, there was no immediate feedback to inform the user that action was being taken.

To address these issues the simple list box widget was replaced with a canvas. On the canvas was placed a list of the names, and beside each was placed a check-box. The check-box was set according to the poll status of the contact. This removed two of the global buttons in the original display, and provided feedback and status information to the user.

Another modification was to move the configuration and quit buttons into a File menu. These functions difficult operations to reverse, and consequently should be less simple to perform. They are also used infrequently compared to the selection buttons. A further change was to move the database editing buttons inside an Edit menu.

The final layout of the control panel is shown in Figure 4.3. The corresponding images in the Activity Display are shown in Figures 4.4 and 4.5.

4.3.2 The Activity Display

Separation of window into panes: one per polling contact. Dynamically adds and removes panes as users are added and removed from the poll list. Panes consists of a text label containing the user name, and a representation of their activity level.

If the poll list is empty it displays a single empty pane, containing a 'No Entry' sign bitmap.

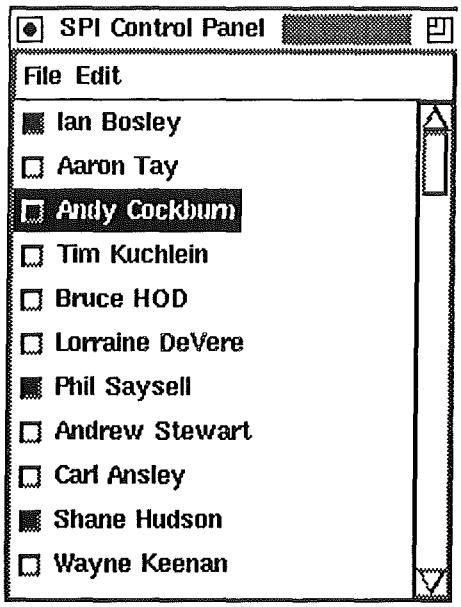


Figure 4.3: The final layout of the SPI User Interface Control Panel

Two of the different display methods discussed in Section 3.4.2 were implemented: the iconic representation model, (Figure 4.4), and the shaded figure model, (Figure 4.5).

4.4 The Poller

This module is responsible for creating the mail messages and sending them to the appropriate destinations. The content of the mail messages is defined by the system configuration and the database entry for each contact.

Database entries for colleagues list their full name, email address, and any

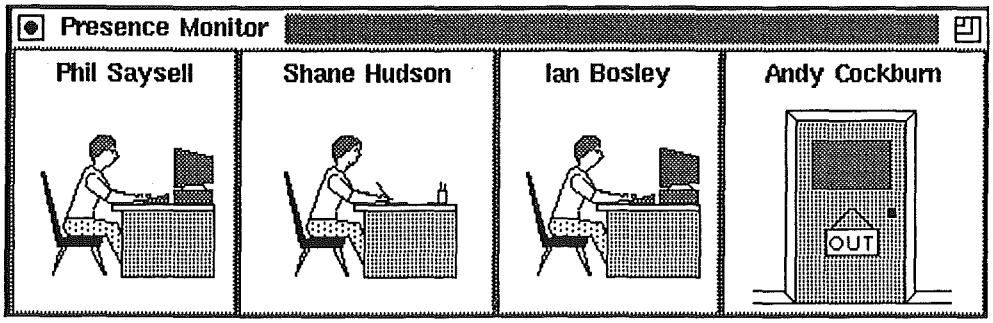


Figure 4.4: The final layout of the SPI Activity Display, showing the iconic representation

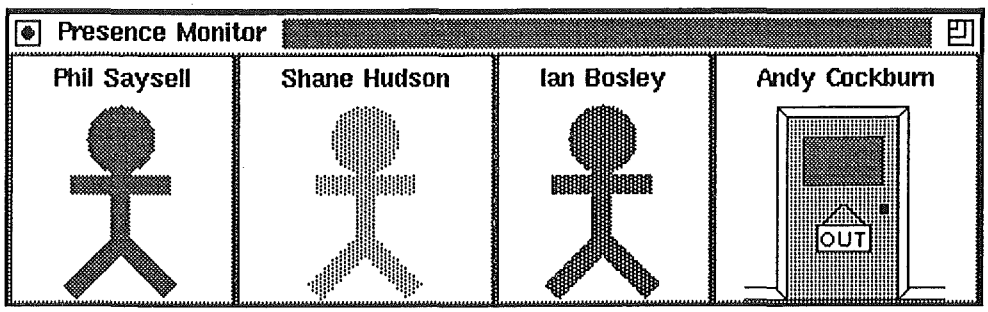


Figure 4.5: The final layout of the SPI Activity Display, showing the shaded figure representation

parameters that should be passed to the awareness command. This allows the monitor to be configured for local area networks, where the user may be one of several hosts.

This information is commonly available through the `finger` command. Although the exact information contained in a `finger` request is largely at the discretion of the system administrator, typically it contains an indication of whether or not the user is logged in, how long their account has been idle, and some information about the status of their mailbox.

An additional benefit to using `finger` is the ability to view a user-created 'plan' file. This file may contain the user's title, telephone number, contact address, work plans and any other information the user wishes to make available. There is a potential for later versions of SPI to make use of this information, and to use the plan file to communicate with other installations of SPIby embedding instructions in a structured format.

The Internet standard for the `finger` command is best described as flexible, (Zimmerman, 1991). The minimum information required to be returned by a `finger` RUIP (Remote User Information Program) is the login name and full name of a user. System administrators may include additional information at their discretion, (Zimmerman, 1991). For the purposes of this system, SPI assumes the presence of at least the login status of the user, and ideally the idle time as well. If `finger` is not able to provide this information, other sources may be available: the `rwho` utility, for example. These requires only minor modifications to the system.

The current implementation of SPI assumes that the poller is actually responsible for arranging the return mail. This requires that the request message contains a an explicit mail command. An example of a SPI request is shown in Figure 4.6.

The message is identified as a SPI message by the presence of the `X-spi` header. The return message is identified by a `Subject: SPI-Reply` header. The awareness command is `finger`, and the `-m -l` parameters force matching of only the user name and the use of the long report format.

The poll cycle consists of sending a SPI request message to each member of

```
To: lorraine@snrc.uow.edu.au
X-spi: request
Subject: SPI test

finger -m -l lorraine | mail -s "SPI-Reply" \
   ian@cosc.canterbury.ac.nz
```

Figure 4.6: An example SPI request message

the poll list. The module then sleeps for a given interval before sending another burst of requests. The interval is a system parameter that may be configured by the user for different environments.

It communicates with the user interface through an intermediate file. The user interface writes this file with the name if each colleague in the poll list. The poller reads it each poll cycle, and sends SPI requests to each of the listed contacts.

4.5 The Remote Mail Processor

This module is responsible for validating incoming mail messages, executing them, and returning the result to the sender. At the simplest level this is simply a `procmail` script that pipes the message into a file and calls a shell to source it.

For security and privacy considerations, this module should validate both the sender and the commands in the request. An easy way to enforce this is to have explicit recipes for each authorised contact, and to enforce a single command local to the remote site. This local command can thereby be configured by the remote contact to provide only the information they feel comfortable with.

4.6 The Returned Mail Processor

This module is responsible for receiving the returned mail messages and processing them to obtain an activity value for the contact. This information is passed to the user interface to display.

4.6.1 From Data to Information

SPI only uses the minimal level of information required: the contact name and the idle time. Additional information used by later versions may include any information from the `.plan` file, and status of mail reading.

Once `finger` has returned user status data, SPI needs to extract the activity information. Command line options for `finger` vary. Information can be in either long or short formats. The short format lists users and their information in a tabular fashion, as in figure 2.1.

```
Login name: ian                         In real life: Ian Bosley
Directory: /users/cosc/honours/ian      Shell: /bin/tcsh
On since Oct 19 11:36:31 on ttyp4 from xsun419:0.0
3 minutes Idle Time
Mail last read Mon Oct 24 15:03:16 1994
Plan:
Ian Bosley                  Currently working on Social Presence
1/12 Warwick Street         on the Internet, through low level
Christchurch 1              tools
Phone: +64 3 389 2343
```

Figure 4.7: Example `finger` information showing the long format

SPI directs its attention towards the long format. This format appears to be more common, and suits our purposes better: the `.plan` file is not shown in the short format.

An example of long format in given in figure 4.7. Other implementations of `finger` might include the office location and telephone extension.

The strongly structured nature of the format is clear. Fields immediately follow field names, which terminate in colons. The order of information is predictable. The report will begin with the login and full names of the user, and typically end with the contents of the `.plan` file.

Some differences appear with the login status and idle time data. The example shows an account which is currently logged in, and has been idle for three minutes.

Absence of an idle time field: If the account was not idle, then no idle information would be shown. The person who launched`finger` is asked to infer that the account is currently active from the knowledge that the user is 'On' and the absence of an idle time.

Indications of a logged out account: If the account is not logged in, then the line beginning 'On since' will begin 'Last Login,' and include the date and time of the last connection. Another alternative is the legend 'Never logged on,' which means that the account has never used the remote machine.

Placement of the Idle Time Statistics: There are also differences in the placement of idle time statistics, if they appear. If the 'On since' line is too long, then the idle time field is displayed at the beginning of the next line. This is the case in the sample format in figure 4.7. However, if the login time and location fields are sufficiently brief, the idle time may follow on the same line as the 'On since' information.

A suggestion for processing this data was allowing user-specified 'spreadsheets.' The principle is to treat each character position in the report as a cell

in a spreadsheet. The system is configured to particular formats by specifying ranges of cell positions for each field.

This idea has merit, but is insufficiently flexible to handle the variable nature of the long `finger` format. The range of the idle time information is not constant in location or size, and may be absent altogether.

The best approach is to develop heuristics for scanning the data. If sufficiently general, it is then easy to adapt the processing to handle variations of `finger` implementations.

4.6.2 Process communication

This module communicates with the user interface through an intermediate file. It appends the latest record to the end of the file. The user interface periodically checks the file for recent additions and updates itself accordingly.

Coherency is of some concern here. As each returning message spawns a new instance of the processor, there may be occasions when one instance overwrites the changes of another. This could be addressed with a file locking operation. However, the data in an individual poll is not crucial. The user interface can afford to 'lose' a poll every so often. As long as clashes are not common occurrences SPI can afford to ignore the problem for the sake of simplicity.

4.7 External Integration

A important design consideration for groupware applications is the ability to augment existing systems with additional new or existing systems. This addresses two crucial issues relating to the success of groupware systems: how the system fits into the work environment in which other systems are used, and how they improve the fulfillment of collaboration needs.

The SPI system maintains a database of information about colleagues and their location. This information can be used to interface with other interaction tools. An example is the Unix utility `talk`. This simple interface for synchronous communication is another widely available low level tool. The SPI user can launch a `talk` session with a colleague simply by double clicking on the colleagues icon in the awareness display.

This demonstrates the ease with which external utilities and communication channels can be integrated with SPI. An further modification might allow the remote colleagues version of SPI to automatically accept the `talk` request if the user is present.

Chapter 5

Security Concerns

> "From an organisational perspective, Groupware is simultaneously a social and a technical intervention."
>
> (Marca & Bock, 1992)

In this chapter, social factors that are relevant to social presence research are discussed. The operation of SPI depends on a degree of openness and trust between participating sites. With openness comes an opportunity for abuse.

Privacy is a major consideration. The social presence function walks a fine line between promoting informal communication and facilitating surveillance.

Another consideration is that of trust and security. There is a potential for abuse in any act of information sharing. The dangers involved in the SPI are significant.

5.1 Privacy

The issues around privacy concern the ability of a user to control what information about them is made available, and to whom. Activity monitors such as SPI raise fears of 'Big Brother'-style surveillance, perhaps linked to productivity assessment.

A social presence system that has caused some concern is the Active Badge Locator System developed by Olivetti Research. This system is composed of small badges which users wear. Throughout their work environment are sensors which can detect and identify individual badges. A central system is continually updated with information about the location of each badge, and therefore it's wearer. Potential benefits include directed telephone calls to the correct extension, and system-supplied diaries of the user's meetings. (Want *et al.*, 1992).

The potential for abuse of such a system is clear, and personal acceptance of this form of social presence system is hindered by user apprehension. Baecker (1993) states:

> "I believe that the perceived dangers (not to mention the real dangers) from the potential misuse of this technology are so great that

no list of modest productivity gains will succeed in making this an acceptable technology."

SPI's intervention is not so great as the Active Badge system. SPI only makes use of information that is already available. The `finger` command only releases as much information as system administrators allow. The global awareness mechanism is essentially consensual. One cannot successfully send SPI messages to another's account without their consent.

The CRUISER system discussed in section 2.2.2 addressed privacy concerns in two ways. Firstly, users could remove themselves from the network by executing a *Private* command. Secondly, a *reciprocity rule* was enforced. This rule required that if someone could see and hear a user, then the user could see and hear that person. (Fish *et al.*, 1993).

The role of reciprocity in awareness facilitation transactions has to be considered. SPI provides some level of security by not allowing unauthorised polling. Reciprocity can be enforced by authorising any user listed in the colleague database.

Another way to address privacy concerns is to maintain a poll request history. Each time a SPI request is received it can be logged. The user can then use this information to remain aware of who has been monitoring their activity.

5.2 Malevolent Abuse

The basis of malicious attacks on computer systems is penetration: the ability to execute commands on the target computer. This is also the basis of SPI's global awareness mechanism.

The utilities used by SPI to acquire global awareness information are `finger` and `sendmail`. The Internet Worm of November 1988 took advantage of security holes in both these utilities to infect over six thousand Internet computers, (Spafford, 1989).

Security measures must be taken to ensure that system integrity is not compromised. There are three distinct issues: preventing unauthorised access, preventing the execution of unauthorised commands by legal users, and auditing the use of the system.

- **Access Authentication**

  SPI must only allow colleagues who have been explicitly declared 'trustworthy' to send request messages. This can be performed at the information filtering level by adding a user name criteria to the `procmail` recipe file. A recipe is written for each trusted colleague. Request messages will only be executed if the contents of the "From:" field match one of the recipes.

- **Message Authentication**

  The actions of the SPI program are limited to social presence. The set of commands that might be executed is therefore limited. SPI could check

that the message only contained legal commands before execution. An alternative is to use a 'restricted shell,' that only allows a safe set of commands to be executed.

Another technique that has benefits outside the security aspect is specifying a script file to execute. The contents of the script file are configured by the remote site, and will only contain commands that they are comfortable with. The additional benefit to this scheme is that configuring the command to the remote environment is done without effort on the part of the local SPI user.

- **Auditing Request Messages**

  This is a passive measure compared to the previous two. It simply requires logging the SPI request messages as they arrive. In the event of unexpected or malicious activity, the audit trail may be used to track any attack through the SPI system.

Computer security is an ever-present concern. The Internet is designed with the open exchange of information as a guiding principle. This inevitably brings security risks. The reduction of these risks must always be a concern for groupware designers.

Chapter 6

Analysis of the SPI System

Firstly, the shortcomings of the SPI system implementation are discussed, identifying areas where the system could be improved. Next, the social presence monitor design assessed followed by discussion of additions to the design that may be of value as continuing research

6.1 Assessment of the Implementation

Low level limitations: There are limitations to the applicability of low level tools in social presence systems. Tools like `finger` often require considerable effort to get accurate information. For example, often email is processed by a 'gateway' machine, which forms a security perimeter for a local network. Users at the site may habitually use a particular machine within the network, or may move around to balance the local network load. Configuring a `finger` poll to account for these idiosyncrasies requires some effort.

The best way around this was discussed in chapter 5. SPI messages simply contain a script command like `spi-awareness`. This command is configured at each remote site to account for the nature of the local network. Unfortunately, this requires effort on the part of participants who do not reap a tangible and commensurate benefit.

Security: The security of the system is not concrete. The reciprocity requirement is easily bypassed. The database keeps separately the email address of the recipient, the login name, and the parameters to pass to `finger`. If is possible, therefore, to send a SPI request to one email address, but to finger a different user.

This problem can be addressed by using the client-server model for the email transactions, (section 3.3). The register can enforce the polling of only the account of email address.

Activity Assessment: The interpretation of the activity data is necessarily arbitrary. This inevitably leads to inaccuracies as it fails to account for individual work habits. User configuration options allow some adjustment, but this increases the cognitive effort required from the user.

User Interface Additions: The user interface to SPI could be enhanced by adding functions to edit the database entries and the system configuration parameters. The database requires effective address book maintenance, including insertion, removal, editing and sorting. The user also requires the ability to easily modify the behaviour of SPI. The following parameters are a minimal list of additions:

- Interval between polls
- Adjustment of the thresholds for levels of activity
- Selection of the display method for activity information

Other display methods may also be added. Currently there are only two: iconic and shaded figure representation. Any and all of the other methods discussed in section 3.4.2 may be implemented, as well as an ability for users to add their own bitmap representations.

6.2 Evaluating the Social Presence Monitor

The design for the social presence monitor remains intuitively correct. There is a difficulty in adequately appraising social presence systems. Literature published about such systems relies heavily on anecdotal evidence and user impressions, rather than empirical observation.

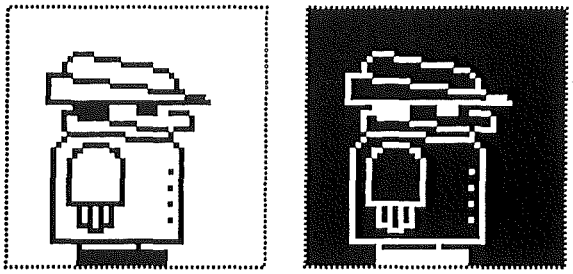
SPI attempts to engage the one of the two concerns facing informal interaction media: mediating physical proximity. Intuitively, a constant display of the activity of one's associates will enhance one's social awareness.

The system design may be improved by assuming more of the user effort in social awareness. A feature of social presence systems is the ability for users to utilise the information to 'ambush' a colleague: that is, to be alerted to a change in state of the colleagues activity, such as returning from a lunch break. While the current SPI can be used to perform this activity, it requires a conscious effort of the part of the user.

SPI can take a more active role by allowing the user to iconise the awareness display, and alerting the user when the state of user changes. A suggested method for this is by a change of icon for the SPI application, similar to many mail programs. Figure 6.1 shows the two states of the icon. The icon change may be accompanied by a beep, depending on user preferences. The advantage lies allowing the user to minimise the display, while retaining a degree of peripheral awareness, a lesson learned from Dourish's Portholes system, (section 2.2.3).

6.3 Further Work

The SPI presence monitor was developed in an incremental style, and still has potential as a 'work in progress.' The original functionality designed in chapter 3 has been successfully implemented, but during development and testing,



(a) (b)

Figure 6.1: The two states of the SPI icon. (a) The current normal icon. (b) The icon that might be shown to alert the user to a change of user activity level.

it became apparent that the system could be expanded and augmented. These additions are covered in the following sections.

This section begins by discussing how SPI can take a more active role in promoting communication. Next it explores uses that the `finger` plan file can be put to, and then the integration of social presence information from other gathering systems. Briefly mentioned are how the security features may be augmented to allow levels of trust for colleagues.

6.3.1 Active Communication Support

SPI can increase it's apparent value by taking a more active role in the initiation of communication links, and the choice of media. Users are likely to use the presence information to decide on an appropriate communication channel. SPI can offer suggestions as to which media to choose. For example, if the activity monitor indicates that the recipient is currently typing on their office computer, then SPI could suggest a telephone link. If the colleague is obviously present but might be out of their own office, email is a better solution. The user can be reasonably confident that it will be read soon, and isn't wasting time playing 'telephone tag' finding the correct extension.

SPI can reduce the effort required in initiating a conversation by using built-in knowledge about communication media and the details of the remote colleague to make a connection. An example that was implemented in the current version of SPI is the ability to launch a `talk` session from the awareness display, (section 4.7). SPI could use a modem to dial the phone number, or an start up an editor to create and send an email message.

The infra-structure is in place to integrate other groupware channels. If SPI can launch a simple `talk` session, why not a shared drawing tool or a group authoring application?

6.3.2 Richer Communication through `finger`

The basic social presence monitor only uses a minimum of the information returned by the `finger` command. A lot of information is discarded. Inferences may be made from the mail box status of the user, such whether they have recently read their mail. Some implementations of `finger` supply information about the physical location of the subject, such as office location and phone number, (Zimmerman, 1991). This information can be used to automatically update address databases.

A great potential exists for communication of information in the `finger` plan file. The plan file is already used creatively in several settings. It is possible to obtain inventories of soda vending machines, weather reports, earthquake information and data about solar flares through `finger` plan files, (Yanoff, 1994). Figure 6.2 shows the status of a vending machine at Carnegie Mellon University.

The plan file provides the opportunity for users to communicate information about their *expected* activity levels. A colleague who is taking three weeks holiday could ask his SPI implementation to store that information in his plan file. Users who poll his account will be warned of his absence by their activity monitors.

6.3.3 External Integration with other Data Sources

Other social presence tools than `finger` are available. The information processor module of SPI could be extended to handle more formats of data.

A more ambitious extension may include handling data from other higher level social presence systems. The mail transaction mechanism can be adapted to handle most forms of data. Even binary image data could be encoded into a format acceptable by SMTP mail.

The SPI system could be extended to use data high technology groupware solutions installed at remote sites. Video snapshots from Dourish's Portholes system could be encoded and distributed through email, effectively extending that system to the wider community of the Internet. Another potential integration partner is the Active Badge system.

6.3.4 Security Enhancements

The security measures is the original design are simplistic. There is one level of privilege: any trusted user is allowed as much access as any other. A more complete SPI system could include more levels of trust. Users could specify how much information each colleague should be given. Some colleagues may be permitted to view calendar data, for instance, while others are only permitted to see the minimal activity information.

6.4 Enhancing the SPI System

When discussing maintenance to the design and implementation of a system, it is always appropriate to consider the ease with which modifications may be

```
$ finger  coke@cs.cmu.edu

[ Forwarding coke as "coke@l.gp.cs.cmu.edu" ]

[L.GP.CS.CMU.EDU]
Login: coke                          Name: Drink Coke
Directory: /usr/coke                 Shell: /usr/local/bin/tcsh
Last login Wed Oct 12 14:27 (EDT) on ttyp1
from PTERO.SOAR.CS.CMU.EDU
Mail came on Wed Oct 26 17:35, last read on Wed Oct 26 17:35
Plan:
Thu Sep 29 17:33:39 1994
M&M validity: 0         Coke validity: 0
(e.g. da interface is down, sorry!)
Exact change required for coke machine.
    M & M                        Buttons
   /-----\           C: CCCCCCCCCCCC............
   |     |         C: CCCCCC......   D: CCCCCC......
   |**   |         C: CCCCCC......   D: CCCCCC......
   |*****|         C: CCCCCC......   D: CCCCCC......
   |*****|                           C: CCCCCC......
   \-----/                           S: CCCCCC......
      |         Key:
      |           0 = warm;  9 = 90% cold;  C = cold;  . = empty
      |           Leftmost soda/pop will be dispensed next
   ---^---
```

Figure 6.2: A `finger` poll of a vending machine at Carnegie Mellon University, showing the current inventory.

made. The SPI system was implemented in `tcl/tk`, a powerful and easy to use script language.

Implementing systems with graphical user interfaces is made easier by such an implementation base. The `tk` extension library was invaluable in the work on SPI. Additional menus and buttons could be added easily.

Another benefit of the `tcl/tk` basis is the inter-process communication this make possible. The `tk` extensions include a `send` command, which can send messsages to other `tcl/tk` interpreters running on the same display. This makes integration of external systems much easier.

Consider a design to add mailbox feedthrough to the awareness display. When email from a work associate arrives in the user's mailbox, they wish some indication of it in the awareness display. If the mail reader used by the user has also been written in `tcl/tk`, then this communication is made much easier. The mail reader application can send an alert message to the SPI process. If the mail message is from a colleague in the poll display, then some indicator can be shown.

The modular design of the SPI system will make additions and modifications easier. The development of the system was a series of augmentations, so further work is simply a continuation of this process.

Chapter 7

Conclusion

> "Networks aren't made of printed circuits, but of people. Right now, as I type, through my keyboard I can touch countless others: friends, strangers, enemies. I can talk to a physicist in Japan, an astronomer in England, a spy in Washington. I might gossip with a buddy in Silicon Valley, or some professor at Berkeley.
>
> My terminal is a door to countless, intricate pathways, leading to untold numbers of neighbours. Thousands of people trust each other enough to tie their systems together."
>
> (Stoll, 1990)

The success of scientific collaboration has been shown to dependent on the establishment and maintenance of a personal relationship between research partners, (Kraut *et al.*, 1988b). Making contact is a prerequisite for *any* collaboration to occur. While physical proximity is certainly the most effective of contact means, in some cases it can be very difficult to achieve. Aside from a simple geographic distance, even partners working in the same building can have trouble keeping track of each other.

Root (1988) identifies the importance of social interaction in organisations, and asks this question:

> *Whether we can identify some function which mediates proximity effects and devise a way to provide that function using advanced communications technology*

His and other papers describe several principles for mediating proximity effects. The two primary issues are cost of interaction, (quality of communication), and cost of establishing contact, (Physical nearness). This project researched ways to provide the necessary functions using *existing* communications technology.

The SPI system is a modelled in part on the CRUISER system, which allowed users to 'browse' a virtual environment looking for casual or spontaneous interaction. The CRUISER system used high bandwidth video technology. In

a network this requires complex hardware, and high bandwidth network connections between partners. The SPI system is aimed at raising social awareness in a similar manner, but using tools common to Internet sites, while creating a minimum of network traffic. Users may find these techniques less intrusive than an unsolicited video connection. The motivation for this low level approach is to maximise the flexibility and availability of the system.

There were decisions to be made in the development of the spi system. The implementation made use of the following tools and mechanisms:

- `tcl/tk` as an implementation platform.
- `finger` for the awareness gathering command.
- Cockburn's (1993) mechanism for using email as a transaction medium, where 'active' messages are executed in remote user accounts.
- A request-response method for making awareness polls, as opposed to a client-server method.
- Iconic and shaded figure representations as choices for the awareness display.

As all social presence system, the awareness monitor raises questions about privacy and security. These issues can be addressed by design and implementation measures. The design can be altered to enforce reciprocity. The implementation can include active security measures and auditing records.

The value of a social presence monitor such as the one designed and implemented is intuitively justified. The monitor was used extensively during the project in it's various incarnations. It is a tool for raising social awareness to promote informal interaction and the development of social relationships. The successful introduction of presence monitors may prove to be widely beneficial to collaborative work, especially that facing difficulties caused by physical distance between group members.

References

Baecker, RM. 1993. *Groupware and computer-supported cooperative work.* Morgan Kaufmann.

Bly, SA, Harrison, SR, & Irwin, S. 1993. Media spaces: video, audio, and computing. *Communications of the ACM*, **36**(1), 28–47.

Cockburn, AJG. 1993. *Groupware design: principles, prototypes, and systems.* Ph.D. thesis, Department of Computing Science and Mathematics, University of Stirling, Scotland. FK9 4LA. Available from British Libraries, or as Technical Report 107 from the University of Stirling.

Cockburn, AJG, & Greenberg, S. 1993. Making contact: getting the group communicating with groupware. *In: COOCS '93. the conference on organisational computing systems. November 1st to 4th. Milpitas, California.*

Cool, C, Fish, RS, Kraut, RE, & Lowery, CM. 1992. Interactive design of video communication systems. *Pages 25–32 of: Proceedings of the 1992 acm conference on computer supported cooperative work* october 31 to november 4 1992. toronto, canada.

Dourish, P, & Bly, S. 1993. Portholes: Supporting awareness in a distributed work group. *In:* (Baecker, 1993).

Ellis, CA, Gibbs, SJ, & Rein, GL. 1991. Groupware: Some issues and experiences. *In:* (Baecker, 1993).

Fish, RS, Kraut, RE, Root, RW, & Rice, RE. 1993. Video as a technology for informal communication. *Communications of the ACM*, **36**(1), 48–61.

Goldberg, Y, Safran, M, Silverman, W, & Shapiro, E. 1992. Active Mail: A framework for intergrated groupware applications. *In:* (Baecker, 1993).

Goodman, GO, & Abel, MJ. 1987. Communication and collaboration: Facilitating cooperative work through communication. *Office: Technology and people*, **3**(2), 129–146.

Goodman, S E, Press, L I, Ruth, S R, & Rutkowski, A M. 1994. The global diffusion of the Internet: Patterns and problems. *Communications of the ACM*, **37**(8), 27–31.

Greif, I (ed). 1988. *Computer supported cooperative work: A book of readings.* Morgan Kaufmann Publishers, San Mateo, California.

Grudin, J. 1994. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, **37**(1), 93–105.

Hogg, J. 1985. Intelligent message systems. *Pages 113–133 of:* Tsichritzis, D (ed), *Office automation.* Springer Verlag.

Hollan, J, & Stornetta, S. 1992. Beyond being there. *Pages 119–125 of: Proceeedings of the CHI'92 conference on human factors in computing systems* monterey, may 3–7, 1992. Addison-Wesley.

Kraut, R, Egido, C, & Galegher, J. 1988a. Patterns of contact and communication in scientific research and collaboration. *Pages 1–12 of: Proceedings of the second conference on computer supported cooperative work* september 26–28 1988. portland, oregon.

Kraut, R, Galegher, J, & Egido, C. 1988b. Relationships and tasks in scientific research collaborations. *In:* (Greif, 1988).

Kraut, RE, Fish, RS, Root, RW, & Chalfonte, BL. 1993. Informal communication in organisations: Form, function and technology. *In:* (Baecker, 1993).

Lai, KY, Malone, TW, & Yu, KC. 1988. Object lens: An "spreadsheet" for cooperative work. *In:* (Baecker, 1993).

Licklider, JCR, & Vezza, A. 1988. Application of information networks. *In:* (Greif, 1988).

Maes, Pattie. 1994. Agents that reduce work and information overload. *Communications of the ACM*, **37**(7), 31–40.

Malone, TW, Grant, KR, Lai, KY, Rao, R, & Rosenblitt, D. 1988. Semistructured messages are surprisingly useful for computer-supported coordination. *In:* (Greif, 1988).

Malone, TW, Grant, KR, Lai, KY, Rao, R, & Rosenblitt, DA. 1989. The information lens: An intelligent system for information sharing and coordination. *In:* (Baecker, 1993).

Marca, D, & Bock, G. 1992. *Groupware: Software for computer supported cooperative work.* IEEE Computer Society Press.

Ming-Dao, Deng. 1992. *365 tao.* Harper San Francisco.

Root, RW. 1988. Design of a multi-media vehicle for social browsing. *Pages 25–38 of: Proceedings of the second conference on computer supported cooperative work* september 26–28 1988. portland, oregon.

Spafford, Eugene. 1989. The Internet Worm: Crisis and aftermath. *Communications of the ACM*, **32**(6), 678–687.

Stoll, C. 1990. *The Cuckoo's Egg.* The Bodley Head Ltd, Great Britain.

Want, R, Hopper, A, Falcao, V, & Gibbons, J. 1992. The active badge location system. *Acm transactions on information systems*, **10**(1), 91–102.

Yanoff, Scott. 1994 (July). `fingerinfo` *version 3.4 source code.* Posted in USENET alt.sources newsgroup.

Zimmerman, D. 1991 (December). *The finger user information protocol.* ARPANET Working Group Requests for Comments: RFC-1288.

Appendix A

SPI Main Module Source Code

The SPI system is launched by executing the **spi** command. SPI takes one optional command line argument. This argument determines the display method to use. Two values are legal: **icon** and **shade**, for the iconic and shaded figure representations respectively.

The main module forks a separate process to manage the awareness display and to perform the polling. The source for this management module is contained in `spipm.tcl`.

A.1 Main executable module – `spi.tcl`

```
#!/usr/local/tcl/bin/wish -f
##################################################
# SPI Executable Module                          #
# Social Presence on the Internet Project        #
# Honours project 1994, Ian H Bosley             #
##################################################

###################################
# Configuration Commands
global env
global argc argv

###################################
# Load appropriate libraries

source db.tcl
source fproc.tcl
source ui4.tcl
source mail.tcl

###################################
```

```
# Global Variables
set spidb ""
set polllist ""
set USER $env(USER)
set DELAY 30000
set level(1) 300
set level(2) 900
set level(3) 14400

if {$argc > 0} {
   set DISPLAY_METHOD [lindex $argv 0]
}
set pm_pid [exec spipm.tcl $DISPLAY_METHOD &]

###################################
# Global Key Bindings
bind all <Control-q> {quit_spi}
bind all <Control-c> {exit}

###################################
# Initialisation Functions

proc init. {} {
  global spidb

  set spidb [openDB "spi.db"]
}

###################################
# Exit Procedure
proc quit_spi {} {
# Tidy up and exit SPI
  send spipm.tcl die
  puts "SPI:  Exiting normally"
  exit
}

###################################
# User Button Procedures
proc start_checking_sel {name} {
# If the current selection is not already being polled
# add them to the poll list, and start a new display for them
  global polllist spidb

        if {![findrecord $polllist 2 $name]} {
           set crecord [getrecord $spidb 2 $name]
           # Name isn't in poll list, so add it to the end.
```

```
            lappend polllist $crecord
            # Create a new display panel for the
            # name, using the full name
            # from field 0 in the database
            send spipm.tcl add_name $name
        }
}

proc stop_checking_sel {name} {
# If the current selection is being polled, remove them from the
# poll list, and withdraw their display.
  global polllist spidb

        if {[findrecord $polllist 2 $name]} {
            set crecord [getrecord $spidb 2 $name]
            # Name is in poll list, so remove it
            set polllist [delrecord $polllist 2 $name]
            # Withdraw the display panel
            send spipm.tcl del_name $name
        }
}

proc do_select_contact {} {
# User has pressed "Select Contacts" Button
  # Open a new window with the Database listed.
  # Buttons to add or delete from poll list.
  open_contact_select
}

proc do_config_spi {} {
# User has pressed "Configure SPI" Button
  # Open a new window with SPI configuration options dialog
  # Poll interval, etc
}


###################################
# Begin (Main Routine)
init.
init_display.
```

A.2 The Poll Manager Module – spipm.tcl

```
#!/usr/local/tcl/bin/wish -f
################################################
# SPI Presence Monitor Module                  #
```

```
# Social Presence on the Internet Project       #
# Honours project 1994, Ian H Bosley            #
################################################

###################################
# Configuration Commands
global env
global argc argv

###################################
# Load appropriate libraries

source db.tcl
source fproc.tcl
source ui4a.tcl
source mail.tcl

###################################
# Global Variables
set spidb ""
set polllist ""
set USER $env(USER)
set DELAY 30000
set level(1) 300
set level(2) 900
set level(3) 14400

if {$argc > 0} {
   set DISPLAY_METHOD [lindex $argv 0]
}


###################################
# Global Key Bindings
bind all <Control-q> {exit}
bind all <Control-c> {exit}

###################################
# Initialisation Functions

proc init. {} {
  global spidb

  set spidb [openDB "spi.db"]
}

###################################
```

```
# Exit Procedure
proc die {} {
# Tidy up and exit SPI
  puts "Presence Monitor:  Exiting normally"
  after 1 exit
}

proc add_name {name} {
# If the current selection is not already being polled
# add them to the poll list, and start a new display for them
  global polllist spidb

        if {![findrecord $polllist 2 $name]} {
           set crecord [getrecord $spidb 2 $name]
           # Name isn't in poll list, so add it to the end.
           lappend polllist $crecord
           # Create a new display panel for the
           # name, using the full name
           # from field 0 in the database
           add_display $name [lindex $crecord 0] \
                       [lindex $crecord 1]
           do_poll $crecord
        }
}

proc del_name {name} {
# If the current selection is being polled, remove them from the
# poll list, and withdraw their display.
  global polllist spidb

        if {[findrecord $polllist 2 $name]} {
           set crecord [getrecord $spidb 2 $name]
           # Name is in poll list, so remove it
           set polllist [delrecord $polllist 2 $name]
           # Withdraw the display panel
           del_display $name
        }
}


###################################
# Polling procedures
proc do_poll record {
  # Perform a finger check
  global USER polllist

  exec finger -m -l [lindex $record 3] > /tmp/spi.$USER
```

```
  set pfile [open /tmp/spi.$USER r]
  set newpoll [getlowest [getact $pfile]]
  close $pfile
  set name [lindex $newpoll 0]
  set idle [lindex $newpoll 1]
  if {[findrecord $polllist 2 $name]} {
    showidle $name $idle
  }
}

proc autopoll {} {
     global polllist DELAY
     foreach record $polllist {
             do_poll $record
     }
     after $DELAY autopoll
}

###################################
# Begin (Main Routine)
init.
create_pm_window
autopoll
```

Appendix B

Database Schema

B.1 Requirements

Fundamental to the SPI concept is a user maintained list of colleagues with whom the user may collaborate or communicate with. If the user feels that they wish to be aware of the presence of a particular colleague, then the SPI system needs to know several things about each one.

The database is required to contain sufficient information to allow the colleagues to be selected by the user from a list of potential contacts and to establish an awareness link to their home accounts. The format must be simple, consistent, and easily edited.

As an interface to the SPI programmer, the database must provide basic database manipulation function: load, search, update, save, and so on.

B.2 Database Schema

B.2.1 Field description

The database for the SPI awareness application has the following fields for each contact.

- Display Name

  The name that should be shown to the user. eg. "Ian Bosley"

- Contact e-mail Address

  The e-mail address to send the awareness polls to.

  eg. iancosc.canterbury.ac.nz

- Status address

  The internet address to give to `finger` for status polling. This allows for the polling to be customised to allow for colleagues who habitually use certain machines in the local network. eg. Bosley@huia

B.2.2 Structure description

The data structure for disc storage is to as follows:

- One line per record, fields listed in schema order.
- Whitespace separates each field.
- If a the content of a field consists of more than one word, it should be delimited with quotation marks.

This is then read in directly into tcl/tk as a list of lists. For example,

```
{{ "Ian Bosley"    ian@cosc.canterbury.ac.nz  ian   ian@huia  }
 { "Chris Stacey"  chris@snrc.uow.edu.au      chris chris@pooh}
 { "Andy Cockburn" andy@cosc.canterbury.ac.nz andy  andy@tete }}
```

B.3 Database API

The following functions are supplied to the SPI programmer for database access and manipulation.

- `openDB` *filename*

  Opens the specified file name and returns a tcl list comprising the contents of the file as a database. It simply reads each line and appends it to the database as a list. Consequently, it performs no checking of the structure of the database and requires no information about the field schema.

- `saveDB` *database filename*

  Opens (destructively) the specified file for writing, and writes the given database structure into it. The format for writing is as given for the database schema

- `dbrecord` *database record-number*

  Returns the record list from position *record-number* in the given database structure. If the index number is out of bounds for the database then the procedure returns a value of zero.

- `dbfield` *database record-number field-number*

  Returns the contents of field *field-number* in record *record-number* from the given database structure. If either of the index numbers are out of bounds for the database then the procedure returns a value of zero.

- `delrecord` *database field-number search-key*

  Searches the database for records that contains the search key in the field number specified. Then removes matching records from the database. Returns a database without the matching records. NOTE: It does not make any changes to the database passed a parameter.

- `findrecord` *database search-key*

  Searches the database for the record that contains the search key in the field number specified. If a match is found, then it returns a true (non-zero) value. If not found, returns a false (zero) value.

- `getrecord` *database field-number search-key*

  Searches the database for the record that contains the search key in the field number specified. Returns the entire record as a list. If not found, returns a negative value.

- `getfieldlist` *database field-number*

  Returns a list built from the contents of the field specified in each record.

B.4 Database Module Source Code – `db.tcl`

```
#################################################
# Database Procedures Library                   #
# Social Presence on the Internet Project       #
# Honours project 1994, Ian H Bosley            #
#################################################

###################################
# File access procedures

proc openDB {dbfile} {
# Open the specified filename, and read it's
# contents into a database list (Returned)
        set dbfile [open $dbfile r]

        while {[gets $dbfile line] >= 0} {
                if {[llength $line] > 0} {
                        lappend dblist "$line"
                }
        }
        close $dbfile
        return $dblist
}

proc saveDB {dbfile dblist} {
# Save the database list into the specified file.
        set dbfile [open $dbfile w]

        set len [llength $dblist]
        for {set i 0} {$i < $len} {incr i 1} {
                puts $dbfile "[lindex $dblist $i]"
        }
```

```
        flush $dbfile
}

###################################
# Database access procedures

proc dbrecord {db recno} {
# return the record at position <recno> in the database db
        # Bounds checking
        if {$recno < [llength $db]} {
                return [lindex $db $recno]
        }
        return 0
}

proc dbfield {db recno fldno} {
# return the field numbered <fldno> in the record at position
# <recno> in the database db
        # Bounds checking
        set rec [dbrecord $db $recno]
        if {$rec == 0} {
           return 0
        }
        if {$fldno < [llength $rec]} {
           return [lindex $rec $fldno]
        }
}
###################################
# Update Procedures

proc delrecord {db fnum key} {
# Find the record matching the key and
# remove it from the database list

 set newdb ""

 for {set j 0} {$j < [llength $db]} {incr j 1} {
   if {[string compare $key [dbfield $db $j $fnum]] != 0} {
      lappend newdb [dbrecord $db $j]
   }
 }
 return $newdb
}

###################################
# Searching procedures
```

```
proc findrecord {db fnum key} {
# Search field <fnum> in database <db>
# for the first match to <key>
# Return true if found, and false if it is not found.
 for {set j 0} {$j < [llength $db]} {incr j 1} {
   if {[string compare $key [dbfield $db $j $fnum]] == 0} {
      return 1
   }
 }
 # Search must have failed
 return 0
}

proc getrecord {db fnum key} {
# Search field <fnum> in database <db>
# for the first match to <key>
# Return -1 is not found.
 for {set j 0} {$j < [llength $db]} {incr j 1} {
   if {[string compare $key [dbfield $db $j $fnum]] == 0} {
     return [dbrecord $db $j]
   }
 }
 # Search must have failed
 return -1
}

proc getfieldlist {db fnum} {
# return a list containing the contents
# of field <fnum> in each of
# the records in database <db>
  set flist ""
  for {set j 0} {$j < [llength $db]} {incr j 1} {
      lappend flist [dbfield $db $j $fnum]
  }
  return $flist
}

#########################################################

# Test stubs
#set clist [openDB spi.db]

#saveDB new.db $clist
#set nlist [openDB new.db]
#puts $nlist

#set name "Carl Ansley"
```

```
#puts [findrecord $clist $name]

#####################################################
# Notify version number and successful installation
# puts "SPI Database Module v. 0.10:  Installed"
```

B.5 Example Database File – spi.db

```
"Ian Bosley"     ian@cosc.canterbury.ac.nz    ian     ian
"Aaron Tay"      aaron@cosc.canterbury.ac.nz  aaron   aaron
"Andy Cockburn"  andy@cosc.canterbury.ac.nz   andy    andy
"Tim Kuchlein"   kuchlein@cosc.canterbury.ac.nz  kuchlein        \
                 "kuchlein kuchlein@cosc"
"Bruce HOD"      bruce@cosc.canterbury.ac.nz  bruce   bruce
"Lorraine DeVere" lorraine@cosc.canterbury.ac.nz  lorraine\
                        lorraine
"Phil Saysell"   phil@cosc.canterbury.ac.nz  phil  phil@kiwi
"Andrew Stewart" stewart@cosc.canterbury.ac.nz stewart stewart
"Carl Ansley" carl@cosc.canterbury.ac.nz carl "carl carl@cosc"
"Shane Hudson" shane@cosc.canterbury.ac.nz shane shane@cosc
"Wayne Keenan" wayne@cosc  wayne "wayne wayne@cosc"
"Peter" peter@cosc peter  "peter peter@cosc"
"Jamie Anstice" anstice@cosc anstice "anstice anstice@cosc"
```

Appendix C

User Interface Library Source Code

The user interface library is split into two sections. One section deals with interface issues for the User Control Interface. The other deals with the functions required by the Awareness Display.

C.1 User Control Interface Module – ui4.tcl

```
#####################################################
# User Interface Library                            #
# Social Presence on the Internet Project           #
# Honours project 1994, Ian H Bosley                #
#####################################################

###################################
# Globals internal to the UI
set UI_disp_count 0

set DISPLAY_METHOD icon
set polling(null) 0

###################################
#  Initialisation Functions
proc init_display. {} {
  global polllist
  global spidb
  global DISPLAY_METHOD

  wm positionfrom . program
  wm sizefrom . program
  wm geometry . +70+156
  wm iconname . {SPI: Control Panel}
  wm iconbitmap . @bitmaps/spidialog.bit
```

```
wm maxsize . 1152 900
wm minsize . 300 500
wm title . {SPI Control Panel}

frame .rim \
  -borderwidth {2} \
  -geometry {300x500} \
  -relief {raised} -bg grey

canvas .listC \
  -width 200 \
  -scrollregion {0c 0c 10c 50c} \
  -yscrollcommand ".scrollC set"

scrollbar .scrollC \
  -command ".listC yview" \
  -relief {sunken}

frame .menu -relief raised -borderwidth 1
menubutton .menu.file -text "File" -menu .menu.file.m
menu .menu.file.m
.menu.file.m add command \
             -label "Configure SPI" \
             -command "do_config_spi"
.menu.file.m add command \
             -label "Quit" \
             -command "quit_spi"

menubutton .menu.edit -text "Edit" -menu .menu.edit.m
menu .menu.edit.m
.menu.edit.m add command \
              -label "Edit contact details" \
              -command "edit_sel"
.menu.edit.m add command \
              -label "Add a new contact" \
              -command "add_new_record"

button .remove   -text "Stop checking name" \
                 -command "stop_checking_sel"
button .add      -text "Start checking name" \
                 -command "start_checking_sel"
button .edit   -text "Edit contact details" \
                 -command "edit_sel"
button .new      -text "Add a new contact" \
                 -command "add_new_record"

pack .rim -side left
```

```
  pack .menu -in .rim -side top -fill x
  pack .menu.file .menu.edit -side left

  pack .listC .scrollC \
       -in .rim -side left -fill y

  set UI_y 0
  foreach record $spidb {
    insert_contact .listC $record $UI_y
    incr UI_y
  }
    # Initialise the poll list displays
  foreach record $polllist {
          add_display [lindex $record 2] [lindex $record 0]
  }

}

proc insert_contact {c record y} {
     global polling

     set fullname [lindex $record 0]
     set name [lindex $record 2]
     set polling($name) 0
     frame $c.$name -width 100 -relief flat
     $c create window 0 [expr $y * 22] -window $c.$name \
        -anchor nw
     checkbutton $c.$name.cb -command "swap_state $name" \
                 -text $fullname \
                 -variable polling($name) \
                 -relief flat
     pack $c.$name.cb -in $c.$name -side left -fill x
}

##################################
# Button Procedures
proc swap_state {name} {
     global polling

     if {! $polling($name)} {
        stop_checking_sel $name
     } else {
        start_checking_sel $name
     }
}
```

```
proc edit_sel {} {
# Edit the database entry for the current selection
  spi_error "Unfortunately, this function\
 has not been implemented yet"
}

proc add_new_record {} {
# Add a new database entry for the current secletion
# and save the database changes
  spi_error "Unfortunately, this function\
 has not been implemented yet"
}

###################################
# Error Dialog
proc spi_error {message} {
     global tk_library
     tk_dialog .spierror "SPI: Information" $message \
               warning 0 "OK"
}

###################################
# Version Notice
# puts "SPI User Interface Library v. 4.10:  Installed"
```

C.2 The Awareness Display Module – ui4a.tcl

```
#####################################################
# User Interface Library                            #
# Social Presence on the Internet Project           #
# Honours project 1994, Ian H Bosley                #
#####################################################

###################################
# Globals internal to the UI
set UI_disp_count 0

set DISPLAY_METHOD icon
set polling(null) 0

###################################
#  Initialisation Functions

proc create_pm_window {} {
# Create the window to use for the presence monitor
  wm positionfrom . program
```

```
  wm sizefrom . program
  wm geometry . +70+0
  wm iconname . {SPI Monitor}
  wm maxsize . 1152 900
  wm minsize . 100 100
  wm iconbitmap . @bitmaps/spi.bit
  wm title . {Presence Monitor}

  blank_pm
}

proc blank_pm {} {

  frame .blank -relief raised -borderwidth 1
  label .blank.bit -bitmap error
  pack .blank
  pack .blank.bit -in .blank -padx 1c -pady 1c

}
###################################
# Display Procedures

proc add_display {name fullname fulladdress} {
  global UI_disp_count
  global DISPLAY_METHOD


  if {$UI_disp_count == 0} {
     destroy .blank
  }

  frame .$name \
        -bg grey \
        -borderwidth {2} \
        -geometry 150x100 \
        -relief {ridge}
  message .$name.name \
          -text "$fullname" \
          -padx 5m -pady 2 \
          -justify center \
          -width 150

  incr UI_disp_count

  switch -regexp -- $DISPLAY_METHOD {
    icon  {
            label .$name.idle \
```

```
                -text "???" \
                -relief {flat}
        }
    shade {
            canvas .$name.idle \
                -width 100 -height 100
        }
  }
  pack .$name -in . -side left -fill y
  pack .$name.name -in .$name -side top -fill x
  pack .$name.idle -in .$name -side bottom

  bind .$name.idle <Double-1> " \
       exec xterm -e talk $fulladdress  "
}

proc del_display {name} {
# Remove a named display panel
  global UI_disp_count

  pack forget .$name
  destroy .$name
  incr UI_disp_count -1
  if {$UI_disp_count == 0} {
    blank_pm
  }
}

proc shade_update {name bitmap} {

    .$name.idle delete idle
    .$name.idle create oval 33 2 67 36 \
         -tags idle -outline "" -fill black
    .$name.idle create line 50 36 50 70 -tags idle -width 10
    .$name.idle create line 25 95 50 70 75 95 \
                           -tags idle -width 10
    .$name.idle create line 25 42 75 42 -tags idle -width 10
    .$name.idle itemconfigure idle -stipple $bitmap
}

proc show_shade_idle {name idle} {
    global level pmd

    if {$idle < 0} {
       .$name.idle delete idle
       .$name.idle create bitmap 50 50 \
                      -bitmap @bitmaps/closed2.bit
```

```
    } elseif {$idle == 0} {
       shade_update $name @bitmaps/grey100.bit
    } elseif {$idle < $level(1)} {
       shade_update $name @bitmaps/grey75.bit
    } elseif {$idle < $level(2)} {
       shade_update $name @bitmaps/grey50.bit
    } elseif {$idle < $level(3)} {
       shade_update $name @bitmaps/grey25.bit
    } else {
       shade_update $name @bitmaps/grey12.bit
    }
}

proc icon_update {name bitmap} {
  .$name.idle configure -bitmap $bitmap
}

proc show_icon_idle {name idle} {
     global level

    if {$idle < 0} {
       icon_update $name @bitmaps/closed2.bit
    } elseif {$idle < $level(1)} {
       icon_update $name @bitmaps/oncomputer.bit
    } elseif {$idle < $level(2)} {
       icon_update $name @bitmaps/atdesk.bit
    } elseif {$idle < $level(3)} {
       icon_update $name @bitmaps/emptydesk.bit
    } else {
       icon_update $name @bitmaps/closed2.bit
    }
}

proc showidle {name idle} {
  global pmd DISPLAY_METHOD
  switch -regexp -- $DISPLAY_METHOD {
    icon  { show_icon_idle $name $idle }
    shade { show_shade_idle $name $idle }
  }
}


##################################
# Version Notice
# puts "SPI User Interface Library v. 4.10.a:  Installed"
```

Appendix D

Heuristic Processing Module

The following functions are provided by the heuristic data processing module.

- **getact** *file-identifier*

  This function reads data from the given file, scanning for activity information. It assumes data is in the long `finger` format. It returns a list of name-idle time pairs.

- getlowest *activity-data-list*

  This function expects data in the format given as output from **getact**. It searches teh list looking for the name-idle time pair with the lowest idle time. Because a negative idle time denotes a logged out account, it has to return the lowest non-negative value unless there are none.

The function **flget** *parameters* will call `finger` in the local environment with the given parameters and return the result in long format. This is for debugging and testing purposes.

```
##################################################
# Activity Data Processing Library               #
# Social Presence on the Internet Project        #
# Honours project 1994, Ian H Bosley             #
##################################################

###################################
#  Utility Functions

proc isnumber {str} {
# Returns true if the str is comprised solely of digits
  return [regexp {^[0-9]+$} $str]
}

proc isalpha {str} {
# Returns true if the str is comprised
# solely of alphabetic characters
```

```
  return [regexp {^[a-zA-Z]$} $str]
}

proc getlowest {data} {
# Assumes data is a list of name / idle-time pairs,
# such as given by activ().
# Return the pair with the lowest positive idle time
# Negative idle times mean "not logged in".
# Only return a negative
# idle time if there are no positive idle times.

  set cname [lindex $data 0]
  set clow  [lindex $data 1]

  for {set idx 2} {$idx < [llength $data]} {incr idx 2} {
      if {( $clow > [lindex $data [expr $idx + 1]] ) ||
          ( $clow < 0 ) } {
         if {[lindex $data [expr $idx + 1]] >= 0} {
           set cname [lindex $data $idx]
           set clow  [lindex $data [expr $idx + 1]]
         }
      }
  }
  return [list $cname $clow]
}

proc gethighest {data} {
# Assumes data is a list of name / idle-time pairs,
# such as given by activ().
# Return the pair with the highest idle time

  set cname [lindex $data 0]
  set clow  [lindex $data 1]

  for {set idx 2} {$idx < [llength $data]} {incr idx 2} {
      if {$clow < [lindex $data [expr $idx + 1]]} {
         set cname [lindex $data $idx]
         set clow  [lindex $data [expr $idx + 1]]
      }
  }
  return [list $cname $clow]
}

###################################
# For finger -i | awk

proc getacti {data} {
```

```
# Data is a list of tokens.  \
# Tokens can either be login names, numbers
# or one of day(s), hour(s), minute(s), or second(s).
  set idx 0
  set rlist ""

  # First line should be a name.
  while {$idx < [llength $data]} {
     set cname [lindex $data $idx]
     incr idx
     set cact 0
     while {[string compare "END" [lindex $data $idx]] != 0} {
         set cfig [lindex $data $idx]
         switch -regexp [lindex $data [expr $idx + 1]] {
           day(s?)    { incr cact [expr $cfig * 86400] }
           hour(s?)   { incr cact [expr $cfig * 3600] }
           minute(s?) { incr cact [expr $cfig * 60] }
           second(s?) { incr cact $cfig }
         }
         incr idx 2
     }
     lappend rlist $cname $cact
     incr idx
  }
  return $rlist
}

###################################
# For long finger format

proc getact {fileid} {
# Process the file stream looking for keywords and
# recognisable patterns
# fileid is a stream of finger output
  set idx 0
  set cname ""
  set rlist ""

  while {[gets $fileid line] >= 0} {
    if {[llength $line] > 0} {
      switch -regexp -- [lindex $line 0] {
        Login   {
                  set cname [lindex $line 2]
                }
        Last    { lappend rlist $cname -1 }
        Never   { lappend rlist $cname -1 }
        On  { # Grab Idle Time
```

```
            # If this line ends in 'Idle Time',
            # then process it
          if {[string compare "Idle Time" \
                    [lrange $line [expr [llength $line] - 2] \
                    end]] == 0} \
          {
            set cidle [find_idle_time $line]
          } else {
            gets $fileid line
            # Check that the line is a valid idle time line
            if {[isnumber [lindex $line 0]]} {
              set cidle [find_idle_time $line]
            } else {
              set cidle 0
            }
          }
          lappend rlist $cname $cidle
           }
      }
    }
  }
  return $rlist
}

proc find_idle_time {data} {
# Assume the data is a list of tokens, starting with
# a full english format idle time
  set idx 0
  set cact 0
  if {[llength $data] > 0} {
    while {[string compare "Idle" [lindex $data $idx]] != 0} {
      set cfig [lindex $data $idx]
      switch -regexp [lindex $data [expr $idx + 1]] {
        day(s?)    { incr cact [expr $cfig * 86400] }
        hour(s?)   { incr cact [expr $cfig * 3600] }
        minute(s?) { incr cact [expr $cfig * 60] }
        second(s?) { incr cact $cfig }
        default    { incr idx -1 }
      }
      incr idx 2
    }
  }
  return $cact
}

###################################
# Returns sample finger data
```

```
# requires presence of fi.awk and
# fsw.awk

proc figet {contact} {
     return [exec finger -f -i -m $contact | awk -f fi.awk]
}

proc fswget {contact} {
     return [exec finger -f -s -w -m $contact | awk -f fsw.awk]
}

proc flget {contact} {
     return [exec finger -l -m $contact ]
}

# puts "SPI Information Processing Module v. 0.10:  Installed"
```

Appendix E

Mail Interface Module

E.1 Mail Interface Source Code – `mail.tcl`

The mail interface consists of a single function call, `sendpoll`, which takes two arguments, *database* and *full-name*. It searches the database for the details of the contact, and constructs a request message.

For debugging purposes, there are options to send the message to a specific local account rather than to the contacts email address. Comment out the inappropriate line.

```
##################################################
# Send Mail Interface Library                    #
# Social Presence on the Internet Project        #
# Honours project 1994, Ian H Bosley             #
##################################################

# Requires SPI Database Library to be loaded

###################################
# Sending routines

proc sendpoll {clist i} {
# For the contact with full name i, send a status poll message
# to their e-mail address using database clist entries
   global USER

   set eaddress [lindex [getrecord $clist 0 $i] 1]
   set fingername [lindex [getrecord $clist 0 $i] 3]
   set tmpfile [open .spitmp w]

# For simplification of testing, just send it to my account
#  puts $tmpfile "To: $eaddress"
   puts $tmpfile "To: ian"
# Swap the commenting on previous line and the line before
# that for real operation.  For debugging change
```

```
# $tmpfile to stdout.
   puts $tmpfile "X-spi: request"
   puts $tmpfile "Subject: SPI test\n"
   puts $tmpfile \
     "finger -m -l $fingername | mail -s \"SPI-Reply\" $USER"
   close $tmpfile
   exec send ./.spitmp
}

# puts "SPI Send Mail Interface Library v. 0.10:  Installed"
```

E.2 procmail Recipe File Example

```
:2 cb
^To:.*ian*
^X-spi: request
| csh; cat > spi-command; source spi-command
:A
.spi-audit-file
:2 b
^To:.*ian*
^Subject:.*SPI-Reply*
spi/spi-responses
```