Proceedings of the Second European Conference on Computer-Supported Cooperative Work
Bannon, L., Robinson, M. & Schmidt, K. (Editors)
September 25-27, 1991, Amsterdam, The Netherlands

Being Selectively Aware with the Khronika System

Lennart Lövstrand
Rank Xerox EuroPARC, United Kingdom

Khronika is an event browsing and notification system that attacks the problems of information overload and information distribution for a wide range of information sources. It implements a shared network server that manages a database of general *events*, personal *event daemons*, and automatically generated *notifications* that are distributed over time. It supports both traditional search and retrieve operations as well as automatic notifications and combinations of both. Together, these two modes complement each other. The first provides a way for the user to actively find out about past, present, or future events; the second causes the system to automatically deliver notifications about pending events of interest. This means that a user can find information when she or he wants to know about it, and be automatically told when she or he needs to know about it.

Intro: A Morning in Leo Lagavulin's Life

It was a cold and dreary day in East Anglia, United Kingdom. "Just as usual," Leo Lagavulin sighed to himself as he was walking over the damp, grassy field towards his office at Rank Xerox EuroPARC. As he approached the back entrance, a subtle click could be heard as his presence was detected and the door unlocked. Leo entered, and inside the warm lobby a disembodied voice greeted him (with a slight Swedish accent): "Good morning, Leo. You have 25 new messages waiting for you. Don't forget your meeting with Margaret Macallan and Oliver Oban at 11:00 this morning. The coffee level is 5 percent." "Damn, someone has forgotten to refill the coffee machine again," Leo muttered as he climbed the stairs to his office on the 4th floor. As he entered his office, he ignored the voice as it came back and told him: "You have had visitors: Bob was here at 09:12." Leo sat down in front of his workstation and looked down at the two main windows — one with his 25 new messages (it would have been 125 in the old days), and one with the week's schedule neatly laid out on a time diagram with the events in which he had registered interest clearly marked. As he started reading his first message, his office started shaking with a thunderous sound from speakers hidden in the ceiling. Leo hastily reached out to turn down the volume — he had left it on an appropriate blues level from last night's late working session — and looked out of his window at the light rain that had just started falling. From his workstation, he could hear the distinct "thud" that indicated that even more mail was coming in as he was sitting there. Leo took a deep breath and proceeded with the chores of the day.

Overview

The main motivation behind the Khronika project is to increase peoples' awareness of what is going on around them over time by improving the effectiveness in which event information is dispersed in a work community. We see this problem as mainly characterised by *information overload* and *information distribution.* By "information overload" we mean the problem of a recipient receiving far too much information — and often irrelevant information at that — than he or she has time to process. The second, "information distribution" problem comes from the difficulty for a sender to correctly identify the appropriate recipients of a message. If the distribution is too narrow, there will be those who will never have a chance to receive it; if it is too wide, the effects of the previous problem will be increased instead. Finally, information about events is intrinsically time dependent: information that arrives too late or too early may be worse than no information at all.

Our solution is to introduce the notion of a networked *event notification service* that receives information about events from various clients, stores it in a database, and ultimately delivers notifications to those interested. Events range in size from "conferences" (days) and "meetings" (hours) to momentary events generated from automatic sensors (no duration). The connection between events and notifications is achieved by means of pattern-action based personal *event daemons* that monitor the event flow and generate notifications when triggered by a matching event. These notifications perform certain tangible actions, such as producing sound effects, synthesised speech, X11 popup windows, or automatically sending electronic mail messages according to the user's personal preferences.

A central notion behind Khronika is that of separating the senders' and recipients' responsibilities by making the system act as an intermediary *information channeller.* With Khronika, senders are only responsible for entering the information and keeping it up to date in case of changes. Instead of having the sender identify the recipients, the recipients themselves "teach" the system about what kind of events they are interested in and how they would like to be told about them. Khronika then automatically notifies them at appropriate times before the event is due to occur, thus obviating the need to manually remember the events at the right moments.

The Problem

A large amount of information is bombarding us every minute, yet something feels amiss. A lot of this "information" is unwanted and undesired, and in fact a serious hindrance to the normal function of our daily life in that by its sheer mass it hides the information we actually *are* interested in. Other pieces of information arrive too early or too late to be of any use or fail to reach us completely, either because we didn't know where to look for it or because nobody thought of sending it in our direction.

If we look at the situation in terms of standard communications theory, we can describe it as one in which a *sender* is transmitting a *message* to a (set of) *recipient(s).* This gives us a vocabulary for describing the problems.

Information Overload

The first problem can be described as arising from the recipient receiving far too many or complex messages than he or she has capacity to process. This is known as the *junk mail phenomenon* [Denning-82].

The standard way around this problem is to apply various filtering methods, with the state-of-the-art solution today still being that of acquiring an information processing tool known as the *personal assistant.*[1] Unfortunately, these are hard to come by and often unaffordable to the ordinary individual.

Other solutions involve computerised filtering systems which usually operate on the recipient's electronic mail messages and perform actions such as sorting them into folders according to author or subject. A prime example is the *Information Lens* system [Malone-87], which implement rule-based *agents* that perform sorting, flagging, and deletion operations on a user's messages as they arrive in his or her mailbox, or even before that, by means of a redistribution mechanism known as the *Anyone* server. Users send messages to the Anyone server instead of directly to individual recipients or distribution lists; the server then runs all the potential recipients' rules and decides based on these who will receive the message. The Information Lens also promotes the use of additional header fields on messages for selection and processing, something that is carried forward in *Object Lens* [Lai-88], where the messages themselves have been turned into collections of general objects.

Spatial Information Distribution

A second problem concerns the difficulties senders have in correctly identifying the appropriate recipients for messages. With traditional communications systems, such as telephone, letters, or indeed electronic mail, it is up to the sender to decide in advance on exactly who will be receiving the message. If too few recipients are chosen there will be those who will never know of the message's existence despite potential interest. On the other hand, if it is sent to too many, the problem of information overload will be increased instead.

A primitive solution is the common *distribution list*, which allows the sender to free herself or himself from names of recipients and rather think in terms of topic categories. This has several problems, however, including (1) multiplexing a single channel for a multitude of different messages, (2) not supporting any notion of memory, which means new recipients cannot access old information, and finally (3) burdening the sender with being responsible for the physical transmission.

[1] Human.

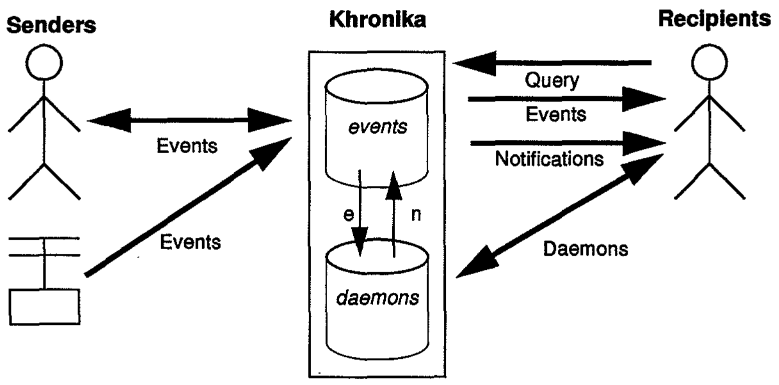

Figure 1. Schematic Overview

Electronic conferencing systems, such as KOM [Palme-84] or USENET [e.g. Spafford-87], remedy this by supporting multiple channels, one for each discussion topic, and by retaining a limited database of past messages so that newcomers don't enter a total void. However, while they often support a limited amount of filtering, they typically do not provide much help for navigating in their information spaces, nor do they automatically inform recipients about new information.

Temporal Information Distribution

A dimension often disregarded in electronic mail and conferencing systems is that of *time*, yet much information is intrinsically time dependent. A message about a seminar becomes fairly uninteresting after the seminar has happened. Similarly, although being told too soon is perhaps better than too late, one is likely to forget meetings if the only notification comes far in advance of the actual event. Clearly, the *act* of informing should be linked with the *time* most relevant to the contents of the message.

Zephyr [DellaFera-89] is a notification system that perform real-time redistribution of incoming messages to subscribing recipients. It uses a ⟨*class, instance, recipient*⟩ triple to dynamically match the classification of the message with possible interested recipients. In addition to providing an on-line interactive messaging facility, the system also handles such tasks as informing users about new mail, systems going down, locating users, etc. However, the system has no memory about the past, nor can it deal with future events. This means that if a recipient is unavailable when the message is sent, it will simply be lost.

The Khronika Solution

As illustrated in Figure 1, Khronika implements a shared *event notification service* that receives information about events from a number of different sources and provides a database of events with both manual browsing capabilities and automatic notifications to interested users. Another look at how event information may be handled in a work community will help in understanding how it operates and is used.

Placing the Recipient in Control

The model behind Khronika is that of clearly separated roles and responsibilities of the sending and receiving agents with the Khronika server itself in the middle as an *information channeller.* The sender of an event message might be the person hosting a seminar series, or the administrator that manages the community's calendar. It may also be a computer program that senses some internal or external sensors, and, as a result, posts an event whenever their state change.

With Khronika, it is the responsibility of the sender to enter the information and keep it up to date in case of changes, no more. Specifically, it is *not* the sender's job to find out who might be interested in the information that he or she is entering; that is up to the potential recipients to specify. This means that the problem of a sender having to identify the recipients of a message is avoided by transferring the task to the recipients themselves — with the aid of Khronika.

On the other side of the system, the recipients have two ways of receiving event information. One is by manual browsing, which directly corresponds to the traditional database search and retrieve operations found in other systems. This allows the interested user to find out about future, present, and past events that match the user's query. Information can presented either in list form or graphically as temporal fields in a weekly calendar.

The other way of receiving information is by means of automatic notifications, which are controlled by the user's personal event daemons. These make the system an active partner in keeping the user aware about what is going on. Users receive notifications by specifying a description of the events they are interested in and when and how they would like to be told about them. For example, a user interested in a particular seminar series might submit a daemon that looks out for events in this series and delivers synthesised speech notifications ten minutes before each seminar is due to begin.

The Triumphant Triumvirate: Events, Daemons, and Notifications

The three entities at the core of Khronika are the *events*, *event daemons*, and *notifications*. Events are at the foundation of the system and denote discrete real-world events of varying duration. For example, an event might be a one-hour seminar, a person visiting for two weeks, or even a proposed call for going to the pub in five minutes. Events are represented by sets of attribute/value pairs and describe objects positioned in a class hierarchy. They are usually presented as forms resembling structured header lines from email messages (see Figure 2). Certain attributes are well-known to the system, such as an event's time or class, while others carry untyped information specific to the class.

ID: 0x281a74ef; Owner: Chalmers; Status: Pending
Ctime: 2-Jun-91 13:39:51; Mtime: 2-Jun-91 13:39:51

Class:	Seminar
Time:	12 June, 12:30 for 1 hour
Speaker:	Graham Button & Wes Sharrock
Title:	Code as Artifact
Host:	Matthew Chalmers
Location:	EuroPARC Commons

ID: 0x2815e210; Owner: Loughnane; Status: Pending
Ctime: 26-Apr-91 16:19:50; Mtime: 12-Jun-91 09:19:22

Class:	Visitor
Time:	16 June to 3 July
Name:	Sara Bly (PARC)
Host:	Bob Anderson
Location:	Room 1.6

Figure 2. Sample Events

Event daemons map a user's personal interests, as expressed by a set of constraints, onto notification templates. Both constraints and templates are currently represented by inactive event objects, although work on a richer constraint specification language is underway. As with the events themselves, event daemons are presented as pairs of attribute/value forms to the user.

Whenever a new event is entered to the system, existing daemons are given a chance to trigger; likewise, when a new daemon is added, it is first compared with all existing events. Triggering occurs when an event matches the search pattern of a daemon and will cause a notification to be spawned as a new event scheduled at the time specified by the daemon's notification relative to the matched event. Thus, if user A enters a seminar event for 14:00 on Friday and user B has a daemon looking for seminars with a 15 minute warning, B's daemon will trigger and schedule a notification for 13:45 the same day.

Notifications are structurally identical to the events themselves, except that they cause some action to be performed by Khronika. For example, when a seminar event occurs, the only thing that happens within Khronika is that it is flagged internally as having started. On the other hand, when a speech notification is due to happen, Khronika will call the appropriate implementation routine performing synthesised speech, in this case a remote procedure call to a networked speech server.

Khronika as a Semiformal System

Khronika can be seen as a semiformal system, as described in [Lai-88] who defines it as a computer system having the following three properties:

1. It represents and automatically processes certain information in formally specified ways.
2. It represents and makes it easy for humans to process the same or other information in ways that are not formally specified.
3. It allows the boundary between formal processing by computers and informal processing by people to be easily changed.

The design of Khronika as a semiformal system means that:

- Users can register informal information with little cognitive overhead; and
- This information can still be made available for computer processing, in our case filtering and automatic actions.

All entities in Khronika are described by the same frame-like structure of named attribute/value pairs. Some attributes apply to all objects, including unique ID, owner, access list, class, and time. Other attributes are dependent on the class itself, e.g. a seminar might have a speaker, a title, and a host, while a visitor event might be endowed with a name, a location, and special requirements. Only the globally applicable attributes are interpreted by Khronika, the others are meaningful only to the human reader and his/her agent, the event daemon.

In the current version, daemon based filtering can be accomplished using a combination of three basic operations: by time interval overlaps, by subclass inclusions, and by substring matches. The first two are specific to the time and class fields, while the last one applies to all other fields; in particular, to the class specific fields.

Although the class hierarchy itself is currently fixed on a server-wide basis, users can create what are effectively new classes simply by adding new fields to already existing ones. These are treated in exactly the same way as other class dependent fields, i.e. totally ignored by the system internally, but available for human inspection and daemon matching. For example, there exists a class called *personal*, which has no fields. A user wanting to record a dentist appointment could bring up a form based on the personal class and start adding fields called (say) "type" and "location," with values like "dentist-appointment" and "Mill Road Surgery," respectively. Daemons could then be specified to notify the user when encountering events with "type: dentist-appointment" fields.

Time

Time is a difficult notion to handle in this sort of system. Computer operations concerning time tend to be very formal and distinct while our everyday usage tends to be relaxed and relatively fuzzy. For example, what is "Thursday afternoon" supposed to mean to the machine? Our approach has been to implement a comprehensive date and time parser for common English expressions and to extend the standard UNIX™ notion of second based offsets from January 1, 1970 to tuples of the same denoting the beginning and duration (or end) of an interval. This allows us to handle constructs such as "tomorrow" or "this week" as well as a seminar that might be expected to be on at "3pm today for one hour". This may not solve all our problems, but at least gives us something that is more appropriate to deal with than "31-Jan-91 17:55:35".

Another problem is the difference in date denotations between Europe and the United States. For example, "1/2/91" may mean either February 1st or January 2nd depending on the user's cultural background. Also, such simple phrases as "this

UNIX is a trademark of AT&T Bell Laboratories.

week" take on a meanings depending on whether the user believes that weeks begins on Sundays or Mondays. By default, our date parser returns an error on ambiguous representations, asking the user to rephrase the expression. Alternatively, a flag can be set to force dates to be parsed with a European or US interpretation.

Manual Browsing vs. Automatic Notifications

Khronika supports two distinct modes of delivering information to recipients: query based listings and automatic notifications. The query facility gives a static view of the event database at the moment the user searched it. It is typically implemented by an interface which asks the user for an event pattern and then presents the result in a list browser.

As discussed before, the notification mechanism is implemented using event daemons and provide a way of automatically informing the user about interesting events. The daemons can be accessed in the same way as the events themselves, primarily by using an event browser-like application.

Together, the two modes complement each other in that one provides a way for the *user* to actively find out about past, present, or future events, while the other shifts the initiative over to the *system's* side, delivering notifications to the user without any further interaction necessary from the user's side. This means that users can find information when they *want* to know about it, and can also be automatically told when they *need* to know about it.

Recently, we have also been exploring interfaces that use an aggregate of the two modes to produce an *active view* of a selected subset of the database. This is achieved by making the interface dynamically create a daemon with a callback notifier that will inform the interface whenever any new, matching events are posted or any changes are made to already displayed events. Among other things, this makes an excellent *active calender* that always is up to date with the latest events.

Shared Access vs. Privacy Control

While there are many benefits to reap from a shared server approach, we must also recognise the individual's requirement for integrity and privacy. Access control has been extensively explored in the area of file systems [e.g. ITS public access, TOPS-10 access patterns, TOPS-20 user groups, UNIX access bits, NORD-10 friend lists], but no single solution appears to be a superset of the others. With Khronika, we wanted to avoid the complexities of a fully fledged group protection system and settled instead on what we thought of as a relatively bare minimum: read access of events are controlled by an explicit access control list and write (change/delete) access is limited to the event's owner. The access control list is just another field of the event (or event daemon), which lists the users who may retrieve the event or otherwise be told about its existence. If left blank, it defaults to "everybody," meaning that every user of the system is allowed to see it. By this we hope to be able to promote automatic sharing of events like public seminar announcements, but still make it possible to keep private meetings and appointments accessible to only a limited number of people.

Khronika as an Environmental Interface

An area which we recently have started exploring is that of Khronika as an interface between the user and his physical environment. There are already several automatic event generators which either monitor external sensors, such as the active badge system [Lamming-91, Newman-91, O'Shea-91] or the weather server on the roof of the building, or internal processes, such as email deliveries or electronic A/V connections. In fact, one of the very first uses of Khronika was that of delivering audible notifications whenever a video connection was opened to a user's camera using *iiif*, the EuroPARC audio/video switching service [Buxton-90]. To achieve this, we modified the switch server to post an iiif feedback event whenever a connection was set up or disconnected. To this was added a set of connection/disconnection daemons for each user that generated appropriate sound notifications on connection events. The effect was that each time someone glanced at another person, that person would hear the sound of a squeaky door opening as the connection was set up. This allowed them to be immediately aware that someone else could see them. When the connection eventually was disconnected, the virtual door would slam shut, indicating that the watching person had left. Modifications to this included versions where the name of the connecting person would be spoken aloud as well as various popup panels and other display devices.

We now have sensory agents and corresponding daemons in use for a number of different types of automatic events. For example, the author, just like the mythical *Leo Lagavulin*, receives new mail notifications by means of a discreet thumping sound and is told about rainy weather by a mighty thunder clap.

The Sights and Sounds of Khronika

It is fair to say that of the available notification actions, *sound* remains as one of the most popular ones. Sound effects of various kinds are currently used to indicate both sensory events as described in the previous section, as well as impending meetings and seminars, etc. For this to work in an office environment, great care has to be devoted both to the technical facilities that make these ubiquitously available as well as to the design of the sounds themselves. Khronika currently supports digitised sound cues delivered over the EuroPARC A/V network or produced directly on the user's workstation. This means that sounds can be played virtually everywhere in the building, including workstation-less offices and meeting rooms. The design of the sounds themselves are described in [Gaver-91], where he also gives examples of other systems using sound for collaboration.

On the visual side, notifications can be made to generate X11 message windows (or indeed any other X11 window or UNIX action) that appear on the user's screen.

These can be made to stay until dismissed by the user, or to go away automatically after a predefined time. For non-workstation notifications, limited support exists for sending video pictures over the A/V network, but it would be a SMOP* to set up a server that would render a text message as a still video frame and transmit it a selected monitor.

Bridging the Gap

In summary, our intention with Khronika as an environmental interface is to try and *blur* the boundaries between computational and real-world events. This is achieved partly by providing a general mechanism that unifies both worlds into one event space, and partly by providing ways of making the resulting events tangible in the human-perceptible world.

Usage Experience

At the time of writing, Khronika has been in continuous use at EuroPARC for over a year. Of the lab's staff of 20-25 people, the majority use Khronika to receive notifications of common events, but only half of these are using all its facilities including the sending part.

Although there have yet not been any formal usage studies, our informal experience is very positive. There have been several cases in which users reported that they received unexpected, but welcome, reminders about events that they were unaware of or about which they had forgotten. In another interesting case, the system became unavailable for a short while and people started expressing concern because they didn't feel like they knew what was going on around them anymore.

Khronika as an environmental interface was an unexpected success. It is now used by a number of people to provide new mail indications and A/V feedback for a variety of connection types as well as the more frivolous rain sound. Khronika is also used directly as a sound generating server by other EuroPARC projects, such as the Portholes and the Activated Active Badge systems, because of its simplicity of use and ubiquitous reach.

Of course, there have been problems too. One of the most difficult ones has been to try and find a balance between the informality of free text and the formality of structured records in describing and representing the events. If they are too structured, it becomes too difficult for people to translate real world events into Khronika events; if they are too "loose," it becomes impossible to do anything meaningful with them. The current solution is more of a working compromise than anything else, and one future direction would be to investigate alternative representations further.

The current method of using a predefined static set of event classes also clearly needs to be changed. While too many classes can cause chaos and confusion, too few

* "Small Matter Of Programming"

is like force-fitting the world into your personal shoebox [sic]. We need some way of organising the event space in a form which can evolve, yet maintain its meaning and have a simple structure.

Finally, Khronika is currently not very good at supporting one-shot reminders, i.e. reminders for individual events. While daemons may be made to trigger on only one specific event, posting a daemon just to create a simple reminder is a bit awkward and unintuitive. Also, while events themselves are automatically removed after a given time, there is no garbage collection for daemons, so they require manual deletion when they become obsolete.

Conclusion

In this report, we have described how Khronika implements a shared event notification service that receives information about events from a number of different sources and automatically delivers notifications at the appropriate times to users interested in the information. By acting as a repository for past, present, and future scheduled events, it supports search and retrieval operations that allow users to interactively inspect the database using different kinds of browsers. Its event daemons let users specify patterns of events about which they would like to be notified, and determine how and when notifications are to occur. This architecture promotes clearly separated roles and responsibilities for the sending and receiving agents, with Khronika itself in the middle as an information channeller.

While calendrical events still make up the core of the system, we have recently started exploring other event types and sources, such as those automatically sensed by electronic agents, leading to blurred boundaries between computational and real-world events. With automatic event sensors and people posting events, we are able to transform information about the environment to a form which can be processed and acted upon by the computer. Likewise, we are carrying events internal to the computer out to the real world by making them, or more correctly, traces of them, tangible by means of human-perceptible sights and sounds.

Acknowledgments

The author would particularly like to thank Thomas P. Moran, formerly director of EuroPARC, now at Xerox PARC, for setting him on this fruitful track and for being a rich source of good ideas and suggestions. Allan MacLean at EuroPARC and Austin Henderson and Sara Bly at PARC have also given a lot of useful input as well as having been an invaluable source of enthusiastic support. Both Bill Gaver and Paul Luff helped considerably with many useful suggestions and corrections to this manuscript; any grammatical errors still lingering are entirely at my own fault. Finally, this report would not be complete without a warm thank-you to Nadia, who has spent many lonely evenings at home while I obsessively worked long and late hours.

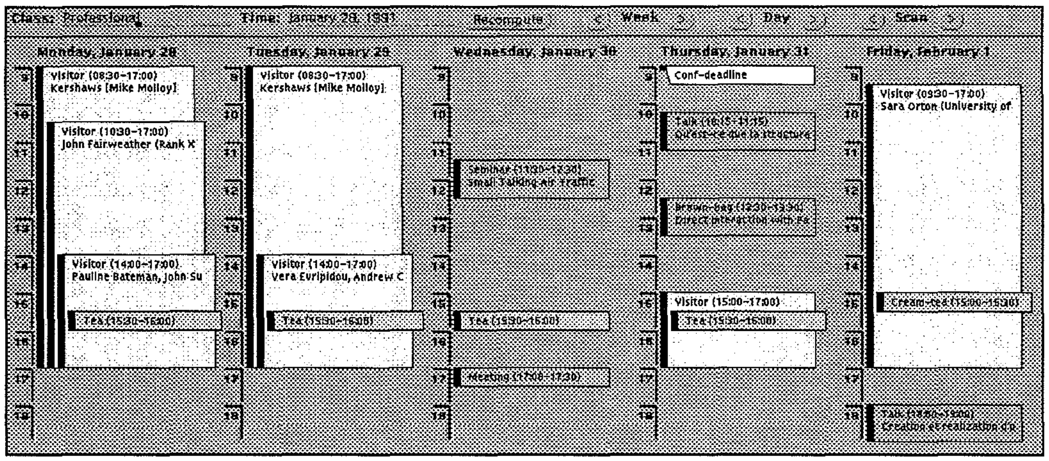
Class: Professional
Time: January 28, 1991
Recompute
< Week >
< Day >
< Scan >
Monday, January 28
Visitor (08:30–17:00) Kershaws [Mike Molloy]
Visitor (10:30–17:00) John Fairweather (Rank X
Visitor (14:00–17:00) Pauline Bateman, John Su
Tea (15:30–16:00)
Tuesday, January 29
Visitor (08:30–17:00) Kershaws [Mike Molloy]
Visitor (14:00–17:00) Vera Evripidou, Andrew C
Tea (15:30–16:00)
Wednesday, January 30
Seminar (11:30–12:30) Small Talking air traffic
Tea (15:30–16:00)
Meeting (17:00–17:30)
Thursday, January 31
Conf-deadline
Talk (10:15–11:15) Qu'est-ce que la structure
Brown-bag (12:30–13:30) Direct interaction with Fe
Visitor (15:00–17:00)
Tea (15:30–16:00)
Friday, February 1
Visitor (09:30–17:00) Sara Orton (University of
Cream-tea (15:00–15:30)
Talk (18:00–19:00) Creation et realization d'u

Figure 3. The XKhBrowser Interface

Implementation Description

The current version of the Khronika server is written in about 10,000 lines of C code and runs on a Sun 4/60 under SunOS 4.1.1. Clients, i.e. user interfaces and automatic event generators, communicate with the server using the SunRPC remote procedure protocol over TCP/IP. Several different suites of client interfaces exist; two using graphical user interfaces (Interlisp-D and X11) and one teletype based one:

Interlisp-D	A browser for events and event daemons; An editor for events and event daemons; Buttons for entering specific events and event daemons; Modifications to the Lafite mail system to post newmail events
X11	*xkhron*, a simple XView/Scheme based search and update tool; *xkhweek*, an experimental active update weekly browser; *xkhbrowser*, a graphical weekly browser with colour coding and proportional temporal layout (see Figure 3)
UNIX	*khputevent, khgetevent, khlistevents*: programs for storing, retrieving, and listing events; *khputdaemon, khgetdaemon, khlistdaemons*: ditto for event daemons; *khgetclass, khlistclasses*: ditto for event classes; *khmkevent, kheditevent, khmkdaemon, kheditdaemon*: shell scripts for interactively creating and editing events and event daemons.

There are currently four different automatic event generators:

khabc	Transfers events from the EuroPARC Active Badge system.
khbiff	Periodically checks if users have received new mail.
khiiif	Listens for connection information from the A/V switch server.
khweather	Periodically polls the EuroPARC weather server for its status.

References

Buxton, W. and Moran, T. (1990): "EuroPARC's Integrated Interactive Intermedia Facility (iiif): Early Experiences," *Proceedings of the IFIP WG8.4 Conference on Multi-User Interfaces and Applications*, Heraklion, Crete, September 1990.

DellaFera, C. A., Eichin, M. W., French, R. S., Jedlinsky, D. C., Kohl, J. T., Sommerfeld, W. E. (1989): *The Zephyr Notification System*, Project Athena, 1989.

Denning, P. J. (1982): "Electronic Junk," *Communications of the ACM*, vol. 25, no 3, pp 163-165, March 1982.

Gaver, W. W. (1991): "Sound Support for Collaboration," *Proceedings of the European Conference on Computer Supported Collaborative Work*, 1991.

Lai, K-Y., Malone, T. W., Yu, K-C. (1988): "Object Lens, 'A Spreadsheet' for Cooperative Work", *ACM Transactions on Office Information Systems*, vol. 6, no 4, October 1988.

Lamming, M. and Wellner, P. (1991): *The EuroPARC Active Badge Service*, forthcoming report from Rank Xerox EuroPARC.

Malone, T. W., Grant, R. G., Lai, K-Y., Rao, R., Rosenblitt, D. A. (1989): "The Information Lens: An intelligent system for information sharing and coordination," in M. Olson (ed). *Technological support for work group collaboration*, 1989.

Newman, W., Eldridge, M., Lamming, M. (1991): "PEPYS: Generating Autobiographies by Automatic Tracking," *Proceedings of the European Conference on Computer Supported Collaborative Work*, 1991.

O'Shea, T., Lamming, M., Chalmers, M., Graube, N., Wellner, P., Wiginton, G. (1991): *Perceptions and Expectations of Ubiquitous Computing: Experiments with BirdDog, a Prototype Person Locator*, submitted to ITAP-91.

Palme, J. (1984): "You have 134 unread mail! Do you Want to read them now?," Computer Based Message Services, *IFIP Proceedings*, 1984.

Spafford G. (1987): "USENET Software: History and Sources", recurrent article in *news.admin* and *news.announce.newusers*, USENET, 1987.