Expediting Rapid Recovery from Interruptions by Providing a Visualisation of Application Activity

Karen Renaud*
University of Glasgow
karen@dcs.gla.ac.uk

May 12, 2000

Abstract

Users will be interrupted by various things throughout their working day. It is desirable for a user to handle interruptions and to be able to easily resume their primary task thereafter. An approach to assisting the user in this task resumption, by means of a visualisation of application activity in a distributed, three-tier application, will be presented. The paper will argue that this visualisation will be invaluable for users in rebuilding context after interruptions. Why this visualisation is important, how the visualisation is achieved, and how it is presented to the user, is discussed. A generic framework which provides this visualisation by means of the observation and interpretation of application activity, is described. The viability of this approach is demonstrated by means of a prototype implementation.

1 Introduction

In executing a task, the user may take the direct route to proceed from beginning to end, as shown by Figure 1, moving directly from the initial state I to the final state F upon completion of the task. Using this direct path, with no detours on the way, is only *one* possible way of doing it. However, this is a simplistic and unrealistic view of the way humans interact with computer applications.

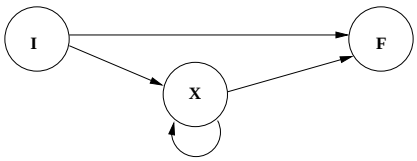

Figure 1: Initial and Final states in Task Execution

The execution of a task can be disrupted by an interruption, shown as node X in the Figure, either initiated by the computer system, by the user themselves, or by some external source. Many interruptions could occur, hence the recursive arrow. When a user is busy with some activity, they build up a *context* [7]. The context is a rich mental environment that stores all sorts of information that has been built up during the time using that particular system to execute some task. Cypher points out that even a momentary interruption will cause this mental context to collapse.

Czerwinski *et al* [8] have shown that advance warning of an interruption will enable the person to remember the context more effectively, and thus enable easier resumption of the interrupted task. People receiving unanticipated interruptions will tend to struggle more to re-establish their context upon resumption of their task. Simon [28] points out that humans are basically serial in their operation, that they can process only a few symbols at a time, and that these symbols must be held in a limited capacity area (working memory) while they are being processed. Seen in this light it is not surprising that interruptions can be so troublesome. Interruptions are therefore a fact of life, and we should try to find ways to assist users in coping with them. The following section will examine interruptions in detail. Section 3 will introduce a scheme for assisting users in coping with interruptions with the minimum amount of disruption to their primary tasks, and Section 4 will describe the prototype implementation of a framework which shows the viability of the proposal. Section 5 concludes.

2 Interruptions

Humans routinely handle up to five activities simultaneously, and with ease, by interleaving them. Cypher [7] maintains that they do this by *linearising* — organising the parallel activities into a single linear stream of

*On study leave from the University of South Africa (renaukv@alpha.unisa.ac.za)

actions. This interleaving of activities could be *voluntary* — such as when we decide that we don't want to wait for something to finish, and switch to another activity — or *involuntary* when, for example, the phone rings and has to be answered.

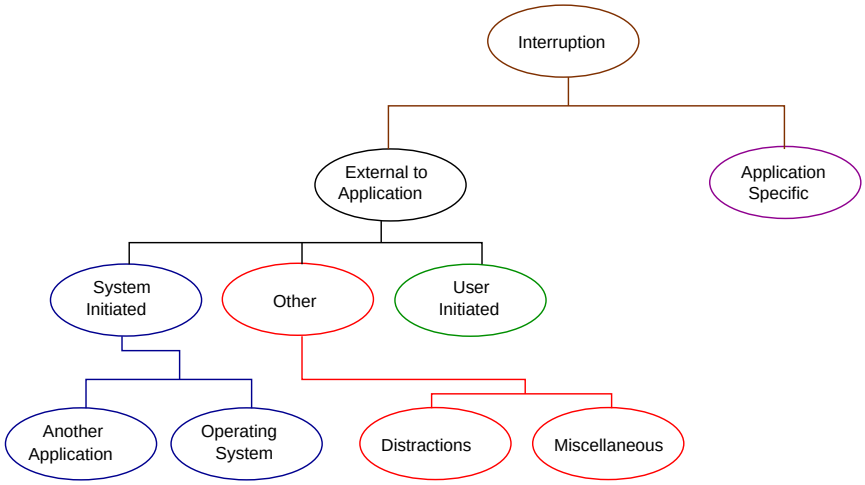

Figure 2: Classification of Interruptions

A classification of interruptions is given in Figure 2. Interruptions can originate either from:

- *within the application* — either signaling an exception with respect to the application itself, or reporting a problem which originates from some other part of the distributed system.
- *some external entity* — the source could be:
  - another application on the system, or the operating system itself.
  - the user. Some examples of user-initiated interruptions can be found in [7, 9].
  - others, like a distraction in the form of a persistent noise, or a fire alarm. Distractions are a special type of interruption, since they do not require any handling, but do interfere with the execution of the task, since they take up processing power within our working memory to register them, and screen them out. When a distraction reaches the level that cannot be borne, the distraction will graduate to an interruption, and the user will take some action to deal with it.

In some cases, the user will resume their original task, but in 45% of cases, according to a study done by O'Conaill and Frohlich [22], the user will not resume the disrupted task, continuing with some other activity — as shown by the transition to node O in Figure 3. O'Conaill and Frohlich worked to quantify the effects of interruptions in a working day. They found that the interruption was often seen to benefit both the initiator and the recipient, so that very few of those who participated tried to dissuade the initiator from making the interruption.

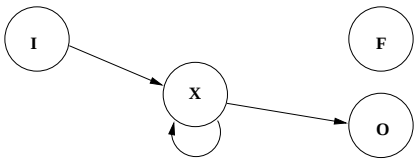

Figure 3: Failing to resume the Interrupted Task

Studies by van Solingen *et al* [31] into the effects of interrupts in software development found that the subjects of the study spent 1 to 1.5 hours per day on interrupts, and concluded that they spent up to 20% of their time servicing interrupts. The recovery time after an interrupt was gauged to be a minimum of 15 minutes.

2.1 Handling Interruptions

The user can either filter out the interruption, or choose to deal with it. In the first case the user carries on with their task and the interruption does not disrupt that process at all, except perhaps interfering with their concentration. In the latter case, the user acknowledges the purpose and content of the interruption, and chooses to either accept, or deny it. Of course some interruptions cannot be ignored, and in that case the user has no choice but to cease their activity and process the interruption. O'Conaill and Frohlich found that users often benefitted from interruptions and were therefore loath to screen them out altogether. It is thus realistic to expect our working day to be interspersed with such events.

If the interruption is accepted, the user needs to decide how the interruption will be dealt with, and change context in order to deal with it. Hitch [15] argues that the load on working memory increases directly in proportion

to the amount of material that must be remembered temporarily or "held in mind". Someone who is processing an interruption will necessarily use most, if not all, of their working memory for their "new" task, and if they are trying to remember what they were doing prior to the interruption *while* processing the interruption, they will not be doing justice to either activity. The speed and accuracy with which people can process information will depend on the working memory load, and if people are to be assisted in handling interruptions effectively, we should find a way to allow them to concentrate fully on the task in hand, rather than trying to retain information about suspended tasks while distractedly processing an interruption.

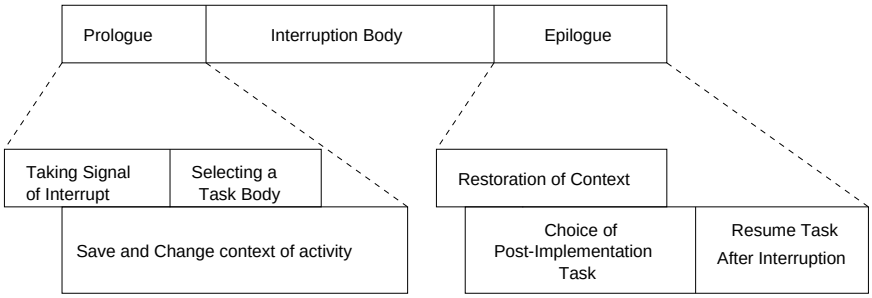

Figure 4: The Sequential Structure of an Interruption [16]

The sequential structure of interrupts is shown in Figure 4. There are three sequential stages, the prologue, the body of the interruption and the epilogue. The three together make up the task interruption. Jambon notes that the body of the interruption is generally independent of the interrupted task.

The prologue and epilogue are often dependent on the interrupted task. The user has to take some action in anticipation of handling the interruption. For example, the user may choose to save the document they are working on before answering the door. The epilogue will require the user to change context once again. They have to try to remember what they were doing, and perhaps retrieve a document from the disk once again before resuming work. The epilogue could also lead to the user deciding to work on another task altogether — and not resuming the interrupted task. Waern [32] notes that working memory is only able to retain information for a couple of seconds at a time, and that unexpected interruptions can thus be fatal to an entire problem solving process.

Sometimes the handling of an interruption is interrupted by yet another interruption. Either the first interruption is suspended so that the most recent one can be dealt with, or the recent one is queued and forced to wait until the handling of the first one has been completed [33]. This mode of handling interruptions is characterised by their sequential nature. However, the user may choose to interleave the handling of the interruptions, as is often done when a person suspends one phone call in order to answer another incoming call, and then attempts to handle both in an interleaved fashion.

After the interruption has been dealt with, the user then needs to change context again, and decide which task to proceed with. Simon [28] (p40) cites two examples from literature about effects of interruption which show that while humans generally can retain up to seven items of information if there is no interruption between the encoding of the information, and the recounting — they only retain two "chunks" of information after an interruption. In order for a computer system to support the user in their linearising of multiple activities, it is essential that the user be provided with some sort of memory aid. This should keep the activity visible, and provide a way for the user to "pick up the threads" as quickly as possible upon resuming an activity. There is, however, a continuous trade-off between providing the user with external memory aids, and the limitations of working space [19]. The following section will examine mechanisms for assisting the user in handling interruptions efficiently.

3 Assisting the User in Handling Interruptions

Upon examination of the effects of interruptions, it becomes clear that what is actually required in alleviating the negative effects of interruptions, is assistance in re-establishing context. Users need to be reminded of what they did, and of what the application did as a result. A mere list of their inputs to the system, such as provided by the `Unix` history, or the `Emacs` selective paste, is of limited use, because they cannot be expected to remember the state of the application at the time of their action. So, together with a reminder of their actions, they must also be able to access an explanation of what the system did in response.

Phillips [24] argues that visual imagery is superior to verbal representation in aiding memory and thinking. Gardiner [13] agrees, saying that recall is better for dynamically interacting items than for items stored in isolation. She avers that recall is further improved if items are presented pictorally, rather than textually. Users have particular strengths which can be utilised by non-textual feedback mechanisms, like processing visual information rapidly, coordinating multiple sources of information, and making inferences about concepts or rules from past experiences [23].

Since the user's interaction with modern computer systems is essentially based on recognition, rather than recall [10], and is intensely visual, it would be less than optimal to try to describe a set of user or system actions in a textual format.

Therefore, the best way to assist in the re-establishment of context after an interruption is by providing a *visualisation of the application's activity* spanning the user's current use of the application. The visualisation should provide a reconstruction of the user's interaction with the application, linked to an explanation of what the system did as a result thereof.

In providing the application activity visualisation, a model of the user's interaction with the application, and the application's reaction to these actions is built, and converted to a helpful and meaningful visualisation. The traditional way of providing any type of feedback has been for the application programmer to provide it *while* developing the prime system functionality.

An alternative approach, described in the rest of this paper, provides a visualisation *independently* of the application. Such a generic facility can be used by many different applications to provide the required visualisation. The framework *observes* application interaction with the environment and provides the visualisation based on this information.

Section 3.1 positions the framework in terms of the type of application it supports, and explains how built-in features of these applications support the framework. Section 3.2 addresses the structure of the visualisation of this activity.

3.1 Supporting Application Features

This generic visualisation providing facility will be developed in the context of thin-client systems. Such systems are generally structured in three, often distributed, tiers with the user interface software on the client machine, the logic at the middle tier, and the data at the third tier. In these systems the sole function of the client tier is to provide the user interface. Inputs are obtained from the user, and sent to the middle tier for processing. These systems will therefore exhibit great amounts of interaction with their environment — both the user, and the rest of the three tier system. A generic facility relies on, and exploits, the following features of these systems:

1. Their tiered structure, with most of the processing is done outside the client machine. The client application makes extensive use of "external" entities to carry out processing on its behalf.

2. The object oriented nature of inter-tier communication. The client program issues requests to the middle tier, and receives replies indicating the success or failure of the processing carried out as a consequence.

3. The fact that the business logic provided by the middle tier of three-tier systems is often supplied by server components housed within an application server. This means that the middle tier server components, being independently developed, are accessed via defined interfaces must be self-describing, and are accompanied by at least some form of documentation which can be harnessed by the framework.

4. The event-based nature of the graphical user interfaces. It is therefore relatively simple to detect meaningful activity, from the application's point of view, at the user interface.

These four features are essential in supporting an independent facility since it must observe application behaviour and supply a explanations and a visualisation of activity based on its interpretation of this observation. The first feature ensures that much of the application activity will be observable. The second ensures that the communication with the middle tier is easily understood, since it is structured in a predictable format. The third ensures that the essence of the communication thus observed can be interpreted correctly, and the fourth feature supplies the framework with an understanding of the importance of events at the user interface. The following section will address the structure of the application activity information to be visualised.

3.2 Application Activity Visualisation

Visualisation provides an interface between the human mind and the computer. In providing any visualisation, there are two issues to be resolved [6], the structure of the information, and the visualisation thereof. The challenge is to find designs that condense detail and hide complexity, rather than presenting the user with a confusing profusion of clutter.

With respect to the application activity structure, Suchman's insight can be used — *"the user's actions can be grouped in a series of displays such that the last action prescribed by each display produces an effect that is detectable by the system, thereby initiating the process that produces the next display"* [29].

The application activity data can therefore be modeled as *event-anchored serial episodic data* [3]. This type of data has episodes, each composed of user activity followed by application activity, with different durations. Each episode is triggered by user actions, signaled by events. Episodes follow each other in serial form, mirroring the serial nature of human processing capabilities.

Carlis and Konstan [3] present a scheme for visualising what they call *serial periodic data* which displays data along a spiral so that both serial and periodic qualities of the data become visible. They have also incorporated

some dynamic querying facilities into their visualisation, feeling that it was not obvious how the focus and context technique would be applied. User experience with their visualisation continues, and no conclusive results have yet been published. Their scheme uses a three dimensional visualisation for the data. However, researchers have recently started expressing some reservations with respect to the use of three dimensional techniques for portraying what is essentially two dimensional data [5].

To model this application activity behaviour in only *two* dimensions, each application thread will be considered in turn, with the sequence of user actions being called a *UI-sequence* (User Interface sequence). A thin-client will relay the user input to the middle tier, as a request for service. The series of requests for service, and replies from the middle tier thus precipitated, is called an *RR-sequence* (Request-Reply Sequence). When a UI-sequence is matched to an RR-sequence, this mapping will be called an *Episode*, as illustrated in Figure 5.

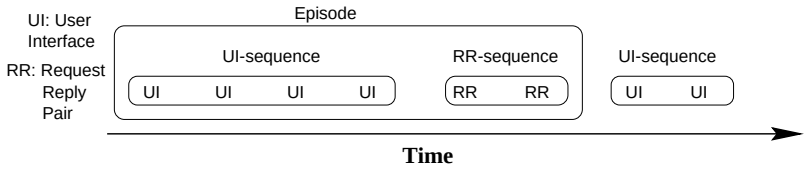

Figure 5: UI-sequences, RR-sequences and an Episode

This visualisation is not merely a matter of displaying the dialogue content, but needs to be linked to the request-reply pairs which were precipitated as a result of the dialogue. If the UI-sequences are considered to be the application's interaction with the user, with the RR-sequences being carried out as a result of the application's understanding of what the user has instructed it to do. Thus the application's response to user inputs can ve revealed. Since the application's actions are often hidden, and therefore unintelligible to the user, they leave the user puzzled. By visualising this activity — making visible what is invisible [20] — promoting a better understanding of application functioning.

In providing any feedback which portrays user interaction history, we need to avoid information overload. We aim to provide tools which will allow the user to get extra information as required [5, 30]. For visualising a great deal of data, Shneiderman [27] advises the use of the *overview and zoom* approach. This approach displays a broad overview and allows the user to zoom in on items of interest, as applied in [12, 25, 26]. The following section will describe the prototype implementation of this framework.

4 HERCULE

HERCULE[1] was developed, using Java, to be a generic prototype facility for visualising application activity. The visualisation provided by the prototype implementation will be discussed in this section. The cost of accessing the information thus visualised is made up of the cost of finding it, and the cost of assimilating it [2]. To reduce the first cost, it should be available at a glance, and to address the second, the information being depicted should not be ambiguous. The user should be left in no doubt of which particular action it refers to. The visualisation will be provided in a window independently of the application.

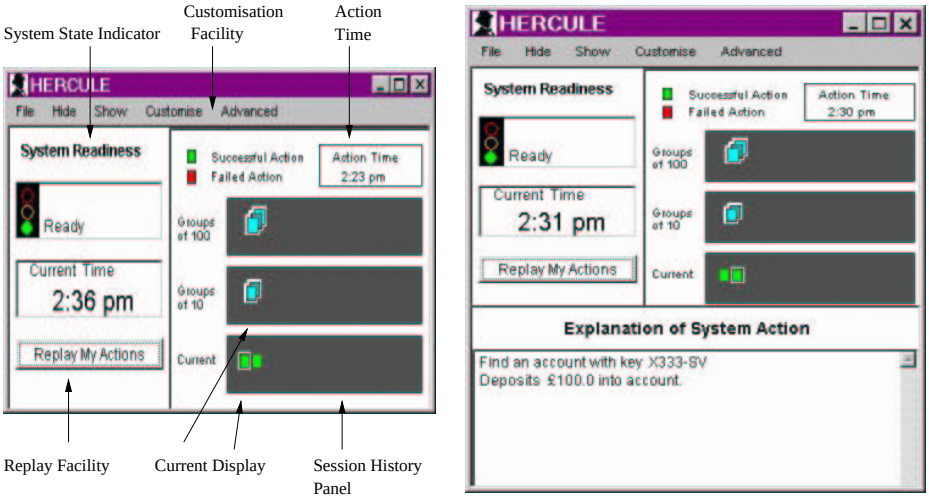

Figure 6: The HERCULE Display

To supply the required visualisation independently of the application, application activity is observed by means of the dynamic insertion of proxies into the system, one tracking user interaction with the application, and another

[1]Named after *Hercule Poirot*, Agatha Christie's legendary detective. It was named after a detective since I reasoned that it watched all activity and tried to make sense of it.

tracking application interaction with the environment. The component documentation for components using the system is also mined, in order to assign a meaning to requests and replies observed. Having observed this activity, we provide a visualisation of the application activity by relating application activity to the meaning of that activity. At runtime, HERCULE (Figure 6) provides the following information, which is dynamically updated as the user works:

1. A traffic lights widget depicting the current system state. This will display:
   - red when the application cannot be tracked. The legend beside the traffic light will display the result of HERCULE's attempt to diagnose the cause. This could be a server breakdown, a network problem, because the application hasn't yet started executing, or because the application has terminated;
   - orange when the server is busy servicing a request; and
   - green when HERCULE is in feedback mode, and waiting for application or user activity to occur.
2. A `Replay My Actions` button will summon a playback facility which allows the user to replay all UI-sequences as they took place. This is done in the form of the windows which were displayed by the application as shown to the user, one at a time, with a highlight on the action which caused the transition to the next window, as shown in Figure 7.

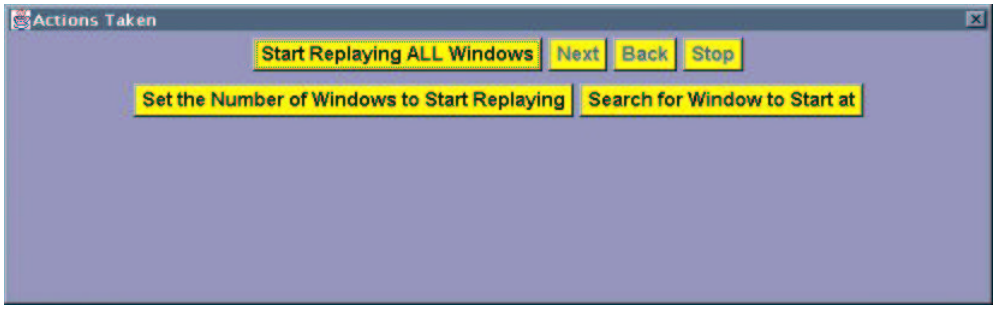

Figure 7: The Playback Facility

   To allow extra flexibility, the user can search for a particular window with some key phrase in it, or step back a certain number of windows, or replay all user interface activity from beginning to end. The replay will show the appearance of each window, as well as the input values entered into the text fields.

   This serves to remind the user of past actions to enable the rebuilding of context. By providing this functionality, HERCULE supports users by alleviating their weaknesses (such as limited working memory), and capitalising on, and utilising their strengths (swift pattern recognition, and retrieving relevant information about the meaning of these patterns quickly).

3. To provide the user with a memory aid to assist them in rebuilding context, they should be able to explore the history of their interaction with the application. This is provided for by means of a session history panel, which presents all Episodes in a hierarchical structure, displayed in three separate panels:
   - the bottom panel displaying the last ten Episodes,
   - the middle panel depicting groups of ten Episodes, and
   - the top panel depicting groups of hundreds of Episodes.

   Each distinct Episode is displayed as a coloured rectangle. This depicts the result of the *RR-sequence* resulting from the Episode *UI-sequence* as:
   - red if it failed — assumed if the server throws an exception,
   - yellow if the outcome is pending, and
   - green if it succeeded — assumed by the absence of an exception.
4. To provide for different types of user needs, HERCULE can be customised to give extended feedback. To provide for this, extra "feedback components" are added to, or removed from, the bottom of the display.

   To help the user to link their actions to the effects of the actions, a feedback component will display an explanation of an RR-sequence. In other words, what the system did as a result of a UI-sequence. This is met as shown in Figure 6. The information depicted is derived from the component documentation, and augmented by the end-user application programmer as required.

   Feedback is provided for the most recent Episode by default, allowing the user to request the display of the explanation of a previous Episode by clicking on one of the previous blocks in the lower panel. Feedback can be obtained for Episodes not shown in the lower panel by clicking on one of the grouped symbols, in the

middle and upper panels. You will note from Figure 6 that each panel displays some rectangles with the rectangle representing the current Episode being highlighted. The *Current Display* label in Figure 6 points to the highlighted rectangles in the history panels, indicating that the most recent Episode's RR-sequence explanations would be displayed (if any feedback components were visible).

The user's immediate querying requirements with respect to individual Episodes, as indicated by clicking on a block which represents a previous Episode, will be met by dynamically updating the explanation for that Episode in the visible feedback components. On the display shown in Figure 6, the Episode actions are explained as `Deposit £100.0 into account`. This is not the explanation of the most recent Episode's RR-sequence, since the highlighted rectangle is in the last but one position, indicating that the explanation belongs to the second last Episode.

The display, shown in Figure 6, satisfies visualisation requirements by depicting all Episodes for the entire application session in one window, by allowing access to any Episode at the click of a mouse, by depicting the visualisation in a small screen space and not intruding, but always being available as an icon or a window.

5 Conclusions and Future Work

This paper has discussed interruptions and asserted the value of a visualisation of application activity to assist the user in handling interruptions. I have proposed the use of application tracking to generate the visualisation. Some of the research done into visualising application activity benefits application developers — usually giving insights about the execution of the application program [1, 11, 17, 18]. The rest of the research is for the benefit of the end-user and depicts a user's interaction with the system in the form of a structure — usually a list – containing a representation of commands. Examples of this are the selective paste offered by `Emacs`, or the `history` command used in Unix. Only the agent implemented by Rich and Sidner attempts to provide a history of user interaction in collaboration with the application, rather than from within the application itself. They provide explanations of the system activity in a textual format, but do not explicitly show the link between user actions and system activities. They also expect the application to provide specific "hooks" which allow the agent to query the application. Our work provides a *link* between user actions and system activity *independently* of the application, something which has not been attempted before.

Finally, the prototype implementation, HERCULE, was described, and examples given of how the visualisation can be provided. The visualisation generated by this prototype is augmented by explanations of system activities, and is dynamically extensible to accommodate changing user requirements.

Acknowledgment

I acknowledge the valuable contributions of Phil Gray, Francis Jambon and Richard Cooper. This research is supported by a scholarship from the Association of Commonwealth Universities and a grant from the Foundation for Research and Development in South Africa, and the University of South Africa. I gratefully acknowledge the donation of the Tengah server from Weblogic/BEA (`weblogic.beasys.com`).

References

[1] M. Brown, J. Domingue, B. Price, and J. Stasko. Software visualization. *ACM SIGCHI Bulletin*, 26(4):32–35, 1994.

[2] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proc. ACM Conf. Human Factors in Computing Systems, CHI*, pages 181–188. ACM, Apr. 1991.

[3] J. V. Carlis and J. A. Konstan. Interactive visualization of serial periodic data. In *Proceedings of the ACM Symposium on User Interface Software and Technology. UIST'98*, Visualization, pages 29–38, San Francisco, CA USA, November 1 - 4 1998.

[4] J. M. Carroll, editor. *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction.* MIT Press, Cambridge, MA, 1987.

[5] M. Chalmers. Information visualization tutorial. In *25th International Conference on Very Large Data Bases VLDB'99*, Edinburgh, Scotland, 7th - 10th September 1999. Morgan Kaufmann.

[6] C. Chen. *Information Visualisation and Virtual Environments.* Springer, Singapore, 1999.

[7] A. Cypher. The structure of users' activities. In D. A. Norman and S. W. Draper, editors, *[21]*, chapter 12, pages 243–264. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[8] M. Czerwinski, S. Chrisman, and B. Schumacher. The effects of warnings and display similarities on interruption in multitasking environments. *SIHCHI Bulletin*, 23(4):38–39, October 1991.

[9] A. Dix, D. Ramduny, and J. Wilkinson. Interruptions deadlines and reminders: Investigations into the flow of cooperative work. Technical Report RR9509, School of Computing and Mathematics, University of Huddersfield, 1995.

[10] A. J. Dix. Closing the loop: modelling action, perception and information. In M. F. C. T. Catarci, S. Levialdi, and G. Santucci., editors, *AVI'96 - Advanced Visual Interfaces*, pages 20–28. ACM Press, 1991.

[11] S. G. Eick, J. L. Steffen, and E. E. Sumner. Seesoft — a tool for visualising software. *IEEE Transactions on Software Engineering*, 18:957–968, Nov. 1992.
[12] G. W. Furnas. Generalized fisheye views. In M. M. Mantei and P. Orbeton, editors, *Proceedings of the ACM Conference on Human Factors in Computer Systems*, SIGCHI Bulletin, pages 16–23. Association for Computer Machinery, New York, U.S.A., 1986.
[13] M. M. Gardiner. Principles from the psychology of memory. In *[14]*, chapter 5, pages 119–162. John Wiley & Sons, 1987. Part II. Episodic Memory.
[14] M. M. Gardiner and B. Christie, editors. *Applying Cognitive Psychology to User Interface Design*, Chichester, 1987. John Wiley & Sons.
[15] G. J. Hitch. Principles from psychology of memory. In *[14]*, chapter 5. John Wiley & Sons, 1987. Part I. Working Memory.
[16] F. Jambon. *Erreurs et interruptions du point de vue de l'ingénierie de l'interaction homme-machine.* Phd thesis, Université Joseph Fourier, 1996.
[17] D. F. Jerding. Visualizing patterns in the execution of object-oriented programs. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2 of *Doctoral Consortium*, pages 47–48, 1996.
[18] D. Kimelman, P. Mittal, E. Schonberg, P. F. Sweeney, K.-Y. Wang, and D. Zernik. Visualizing the execution of High Performance Fortran (HPF) programs. In IEEE, editor, *IPPS '95: 9th International parallel processing symposium — April 25–28, 1995, Santa Barbara, CA*, International Parallel Processing Symposium, pages 750–759, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.
[19] Y. Miyata and D. A. Norman. Psychological issues in support of multiple activities. In D. A. Norman and S. W. Draper, editors, *[21]*, chapter 13, pages 171–186. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.
[20] D. A. Norman. *The design of everyday things.* MIT Press, London, England, 98.
[21] D. A. Norman and S. W. Draper, editors. *User Centred System Design. New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.
[22] B. O'Conaill and D. Frohlich. Timespace in the workplace: Dealing with interruptions. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 of *Short Papers: Workplaces and Classrooms*, pages 262–263, 1995.
[23] J. R. Olsen. Cognitive analysis of people's use of software. In *[4]*, chapter 10, pages 260–293. MIT Press, 1987.
[24] R. J. Phillips. Computer graphics as a memory aid and a thinking aid. *Journal of Computer Assisted Learning*, 2:37–44, 1986.
[25] G. G. Robertson and J. D. Mackinlay. The document lens. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Visualizing Information, pages 101–108, 1993.
[26] M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss. Stretching the rubber sheet: A metaphor for viewing large layouts on small screens. Technical Report CS-93-39, Department of Computer Science, Brown University, Box 1910, Providence, RI 02912, Sept. 1993.
[27] B. Shneiderman. *Designing the User Interface.* Addison-Wesley, Reading, Massachusetts, 1998.
[28] H. A. Simon. *The Sciences of the Artificial.* The M.I.T Press, Cambridge, Massachusetts, 1969.
[29] L. Suchman. *Plans and Situated Actions.* Cambridge University Press, Cambridge, 1987.
[30] E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, U.S.A., May 1990.
[31] R. van Solingen, E. Berghout, and F. van Latum. Interrupts: Just a Minute Never Is. *IEEE Software*, 15(5):97–103, September/October 1998.
[32] Y. Waern. *Cognitive Aspects of Computer Supported Tasks.* John Wiley & Sons, Chichester, 1989.
[33] W. Walker and H. G. Cragon. Interrupt processing in concurrent processors. *Computer*, 28(6):36–46, June 1995.