

The Event Horizon User Interface Model for Small Devices

Antero Taivalsaari

The Event Horizon User Interface Model for Small Devices

Antero Taivalsaari

SMLI TR-99-74

March 1999

Abstract:

One of the scarcest resources in today's computing devices is screen space. This is particularly evident in the emerging pocket-sized devices such as PDAs, cellular tele- and dataphones, two-way pagers and wireless web browsers. In this paper, we introduce a new approach, called the *event horizon* user interface model that allows us to overcome some of the limitations. The key idea of the proposed model is that the display can be compressed and expanded by moving objects radially farther away or closer to an event horizon in the middle of the screen. In order to study the proposed model in detail, we have built an implementation called *Radius* for the Palm™ connected organizer. The current Radius implementation will be illustrated and some future directions outlined.



M/S MTV29-01
901 San Antonio Road
Palo Alto, CA 94303-4900

email address:
antero.taivalsaari@sun.com

© 1999 Sun Microsystems, Inc. All rights reserved. The SML Technical Report Series is published by Sun Microsystems Laboratories, of Sun Microsystems, Inc. Printed in U.S.A.

Unlimited copying without fee is permitted provided that the copies are not made nor distributed for direct commercial advantage, and credit to the source is given. Otherwise, no part of this work covered by copyright hereon may be reproduced in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

TRADEMARKS

Sun, Sun Microsystems, Java, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@eng.sun.com>.

The Event Horizon User Interface Model For Small Devices

Antero Taivalsaari

Sun Microsystems Laboratories
901 San Antonio Road, MS MTV29-117
Palo Alto, California 94303

1. Introduction

The processing power and storage capacity of computing devices has increased by several orders of magnitude in the past twenty years. In the early 1980s the average desktop computer had a few tens of kilobytes of RAM and maybe a few hundred kilobytes of (floppy) disk space. In 1999 a typical personal computer has 32-128 megabytes of RAM and 4-16 gigabytes of hard disk space. However, at the same time, the average computer screen size has barely doubled in linear size. This has led to an interesting dilemma: while the virtual space inside the computers has expanded rapidly, the physical windows through which we look into that space allow us to see a relatively smaller and smaller portion of the space. For instance, in the early 1980s a typical desktop computer contained no more than a few hundred files, and listing all of them using a simple command line operation or a visual file manager tool took no more than a few seconds. In 1999, personal computers may contain tens or hundreds of thousands of files, and listing them all could take hours or days. The problem is exacerbated by the fact that today's computers are increasingly networked, allowing access to millions and millions of files on the Internet, making information finding and visualization even more challenging.

In general, while the development of computers and software has traditionally been guided primarily by the need for faster processors and more memory, today development is increasingly being driven by other issues, such as the need for easier and faster network access, or better mobility. At the same time, it is becoming apparent that one of the scarcest resources in today's computing devices is screen space. There is simply not enough space on a typical computer screen to visualize the rich information content characteristic of today's computers and computer networks. This problem is particularly evident in the emerging pocket-sized devices such as personal digital assistants (PDAs), cellular telephones and dataphones, two-way pagers and wireless web browsers. These devices typically have a screen size of only a few inches across—yet these devices are more and more commonly being used for display-hungry activities such as browsing the web, accessing large corporate databases, or even capturing and displaying live video. What is even more problematic is that these devices are increasingly being used as

replacements for personal computers for storing large amounts of business-critical information such as contact lists, memos, business contracts, e-mail, fax messages, etc. Organizing and rapidly finding such information can be extremely cumbersome on a screen that is only a few inches across.

In this paper we introduce a new *collapsible* user interface concept that allows us to overcome some of the limitations arising from the small screen size. The proposed concept is general and can be used not only in small devices but also in full-size desktop computer user interfaces. The concept can be used either separately or in conjunction with other user interface concepts such as the traditional desktop metaphor. To illustrate the proposed concept, we have built an implementation for the Palm connected organizer—the most popular personal organizer device today. The current implementation replaces the standard Memo Pad application of the Palm device, but it could easily be extended for other purposes as well.

The rest of this paper has been organized as follows. We start by briefly discussing the evolution of user interfaces in commercial computer systems and by examining alternative user interface concepts that have been proposed in the research world (Section 2). In Section 3, we present our new model and discuss its main benefits and drawbacks from the conceptual viewpoint. In Section 4, we introduce our current implementation of the new user interface for the Palm connected organizer. Finally, in Section 5 we summarize our experiences with the new model.

2. Comments on user interface evolution

2.1. The evolution of commercial user interfaces in the 1980s and 1990s

Considering the dramatic advances in the processing power and storage capacity of computer systems in the past two decades, it is somewhat surprising how little progress has been made in the area of commercial computer user interfaces in the past fifteen years. While the 1980s were characterized by a rapid transition from character-based user interfaces to graphical user interfaces—inspired by the success of Apple Computer’s popular Macintosh product line—in the 1990s relatively few advances have been made. The basic WIMP (Windows, Icons, Menus and Pointers) approach and the familiar desktop metaphor have been augmented by a few additional concepts—such as floating toolbars, wizard dialogs, extensive use of drag-and-drop editing—but otherwise the basic user interface concepts and look-and-feel have remained largely unchanged from the mid-1980s. From the research perspective, most of these concepts hark back to the 1960s and early 1970s.

The only real challenge to the status quo in computer user interfaces in the past few years has been the introduction of web browsing. The rapid popularization of the World-Wide Web and the growing success of electronic commerce have made the web browser one of the most frequently used computer applications. Unfortunately, this transition has not

been an entirely positive phenomenon. While web browsers are generally capable of supporting far richer information content and mosaic of media (graphics, video, audio) than traditional computer user interfaces, web browsers have some fundamental flaws. In particular, the page-oriented, server-centric interaction and navigation model of the web is taking us away from the direct manipulation model [Shn83] popularized by personal computers and workstations in the 1980s. Rather than dealing with application objects directly, the user is manipulating data in a very indirect fashion by sending and receiving pages one at a time.

Even worse, rather than manipulating *objects*, the user is dealing with low-level abstractions described and accessed using arcane and extremely cumbersome facilities such as HTML and CGI. In general, currently the World-Wide Web is still suffering from the fact that it is a poorly-abstracted, poorly-organized, page-oriented world—instead of being a world of objects and services as it could be. Fortunately, Java™ and Jini™ technologies are gradually changing this by enabling the development of networked services in a more object-oriented fashion. However, neither of these technologies directly addresses the user interface issues pertaining to small devices.

In general, it is interesting to notice that even though web browsers were initially intended to be platform-independent across different kinds of computing devices, the chosen lowest common denominator was still far too high to enable web browsing from small consumer and embedded devices. For this reason, a number of proprietary techniques have been developed. For instance, the recently introduced *web clipping* technology (http://www.palm.com/devzone/palmvii/tutorials/tutorial_web.html) by 3Com Corporation allows a mobile web browser to extract only a specific set of the information from a given web page, eliminating the less important information as specified by the creator of the web site. Obviously, this technology addresses only the problem of making web pages smaller. The more fundamental problem—how to efficiently visualize and organize a large amount of information and objects on a small screen—remains unsolved.

2.2. Alternative user interface models

There are two general approaches to solving the problems that arise from limited screen space. The first approach is to try to reduce the number of objects that are visible on the screen simultaneously. This can be accomplished, e.g., by creating a hierarchy of objects: instead of showing all the objects at once, the system displays only one level of the hierarchy at a time. This approach has been utilized extensively in traditional graphical user interfaces. For instance, the desktop/folder metaphor familiar from the Macintosh computers uses this principle. Also, instead of displaying full objects, it is possible to use icons or other more compact representations to conserve space.

The second general approach is to make the screen larger. This can be accomplished either physically, for instance by buying a bigger screen, or virtually by extending the virtual display space inside the computer. Obviously, there are technological and

economic constraints as to how big the physical screen can be made, and in small devices this approach is not feasible in the first place. Therefore, we are left with the latter possibility: enlarging the virtual display space.

Zooming user interfaces. In recent years several interesting user interface concepts have been presented. For instance, there are several systems that utilize *zooming* as the main technique for extending the virtual screen space. Examples of such systems include *Pad++* [BeH94, BSH94], *Workscape* [LuS94, Bal94], *Web Forager* [CRY96] and *Data Mountain* [RCL98]. In these systems, the user can control the number of objects visible on the screen either by pushing and pulling selected objects farther or closer, or by zooming the whole display in and out to decrease and increase the number of objects that are visible to the user. By zooming the display out, the user can see a much larger number of objects than would be possible in a regular, non-zooming user interface. By zooming the display in, the user can focus on the details on individual objects. Some systems, such as *Workscape*, even use zooming as a substitute for object opening, i.e., unlike in traditional user interfaces in which the user typically has to explicitly click or double-click an object to open or edit it, objects in *Workscape* are “always open” [LuS94]. The user who wants to work on a particular object can simply zoom in to that object and start working on it immediately. Since no explicit opening of objects or applications is necessary, it is also possible to work on an object or multiple objects “from the distance”. In some systems zoom rates are set automatically based on the size of the objects the user is currently working on [WLS98].

Zooming user interfaces have some potential problems. For instance, insufficient screen resolution can make it difficult to view objects that have been zoomed far out. If the user frequently needs to move objects back and forth to improve legibility or to see more objects, the value of the user interface will decrease substantially. Also, it is tricky to implement zooming in a realistic and aesthetically pleasant fashion without addressing some thorny implementation issues such as object collision. For instance, it is not obvious what should happen when an object is moved from behind another object to the front of that object. Should the other object give way or are objects allowed to pass through each other? Furthermore, to implement zooming smoothly and realistically, quite a lot of computing power is required.

Focus+context displays. There are several other interesting techniques that can be viewed as variations of zooming. For instance, the *document lens* model introduced by Xerox PARC [RoM93] allows the user to focus on a particular object or part of the screen and magnify the focused area to cover most of the visible screen area. At the same time, the user can still view some of the surrounding context. This technique is particularly useful in situations in which the user does not want to lose the global context information about the document while focusing on a specific part of the document. Such a requirement is often common, e.g., when reading large legal documents, reference manuals, or other books containing numerous cross references. User interface techniques that allow the user to focus on a particular object without losing global contextual information are often referred to as “*focus+context*” displays [RoM93].

Variations of focus+context displays include *fish-eye views* [Fur86, SaB92, Gre96], *image magnifiers* [WaL95], *magic lenses* [BSP94, Fox98] and *hyperbolic views* [LaR94, LRP95, HGD95, LaR96]. Some systems utilize *radar views* [GGR96]: Rather than zooming or otherwise manipulating the main view, the system creates an additional view or window that allows the user to see a miniaturized version of the full virtual display while the main view displays only a certain part of the full display. For instance, the Kansas system discussed below uses the radar view technique.

Infinitely large virtual displays. An alternative to zooming is to make the virtual display space infinitely large. The *Kansas* distance collaboration system [SHH98] developed at Sun Microsystems Laboratories uses this approach. In *Kansas*, all the objects reside in the same, large, flat, two-dimensional, shared space (hence the name *Kansas*). The user can move about in the space by panning the display left/right and up/down. *Kansas* is a multi-user system, so multiple users can coexist in the same space, share objects and work together collaboratively. There is no hierarchical structuring of user interface space: all objects are assumed to reside in the same world. To facilitate navigation and understanding of global context, additional tools such as the radar view are provided.

The *Kansas* system was originally built as a user interface for the *Self* programming language [UnS87], and it is based on research ideas experimented earlier in the *ARK* (Alternate Reality Kit) system at Xerox PARC [Smi86, Smi87]. A number of other systems utilize the same principle. For instance, the collaborative design system *TDE* of Nokia Corporation [TaV97] uses the same shared world/infinitely large flat space metaphor in combination with hierarchical folders and zooming capabilities.

Graph-based displays. A popular approach to visualizing large amounts of information is to use different kinds of graph representations. For instance, the graphical user interfaces of the popular operating systems such as the MacOS or Windows have traditionally provided a *tree view* for visualizing and manipulating directory hierarchies. However, the use of graph-based representations goes far beyond simple tree views. For instance, the *cone tree* display used in the *Information Visualizer* system [RMC91] combines graph-based representation with an animated three-dimensional display.

Perhaps the most relevant graph-based user interfaces for small devices are so-called *hyperbolic views* [LaR94, LRP95, HGD95, LaR96] already briefly mentioned above. In hyperbolic views, the user interface is arranged in a circular fashion so that the object that is currently the center of attention is located at the center of the display. The objects that are next relevant are then displayed radially around the object at the center. In a way, the display contains several “rings” of objects, each ring containing objects whose “conceptual proximity” to the object at the center is about the same. Interconnections of objects are represented visually as graph edges so that objects at the first ring have a visual link to the object at the center, objects at the second ring have a visual link to a related object at the first ring, and so on. The navigation in a hyperbolic view takes place by clicking any of the surrounding objects. Whenever the user clicks an object, the object

is brought to the center of the display and the rest of the display is reorganized accordingly. The object that was previously the center of attention becomes part of some surrounding ring.

Real-world metaphor displays. It is very typical for user interface designers to try to map the user interface concepts to some real-world phenomena. The well-known *desktop/folder* metaphor used by various operating systems is a good example of this approach. Other common metaphors include the *card deck*, *pile*, *book/bookshelf* and *room* metaphors. The card deck metaphor was popularized by Bill Atkinson's successful Apple *Hypercard* program released in 1987, and has been used also, e.g., in the experimental web browser *DeckScape* [BrS95, BrS96]. The pile metaphor was investigated in [MSW92]. The book and bookshelf metaphors have been used, e.g., in various calendar and memo pad applications (see [GFC98]) and in the experimental web browser *Web Forager* [CRY96]. The room metaphor is characteristic of collaborative systems such as *TeamRooms* [RoG96] (see also [HeC86]).

Data-type specific displays. Rather than using metaphors, it is sometimes a good idea to adapt the user interface to the data types that are being manipulated. For instance, there are user interface concepts that have been tailored particularly to viewing tables and other grid-like structures [RCJ92, RaC95]. *Treemaps* [Joh92] are particularly well-suited to visualizing large hierarchical structures such as file directories in association with other attributes such as file size and type; for instance, the commercial *DiskMapper* system by Micro Logic Corporation (www.miclog.com) uses this approach.

Other user interface concepts. There are a number of relatively new user interface concepts that utilize *time* as the main organizing criterion. For instance, in the *Lifestreams* system [FFG96] the user can create several "streams of objects" for organizing objects, and the system then automatically keeps the most recent objects in each stream on top of the display. A similar approach has been used in the *LifeLines* system [PMR96]. *Three-dimensional views* and *virtual reality displays* are also becoming more feasible as the available computing power increases, but these are beyond discussion here since these techniques are not generally very applicable to small devices. For further discussion on characterizing and classifying different user interface concepts, refer to [Sta93, Twe97].

2.3. User interface concepts for small devices

Small computing devices such as personal digital assistants, cellular telephones, pagers, wristwatches and bicycle computers typically have a screen that is much too small to accommodate a traditional desktop user interface. Limited computing power, poor screen resolution, and strict memory constraints make it difficult to utilize more advanced techniques such as zooming, scaling, or infinitely large virtual displays. Further, many of the small devices are generally intended to be manipulated with one hand only, imposing limitations on the input methods that can be used. For instance, the traditional WIMP (Windows, Icons, Mouse and Pull-Down Menus) approach requires that the user can

reserve one hand for operating a pointing device such as a mouse or a stylus. This may not be feasible when operating devices such as cellular telephones or automobile computers.

Given the large number of user interface proposals in the literature and the growing popularity of small computing devices, it is somewhat surprising how little research effort has been devoted to designing user interfaces for small devices. This may be partly because user interfaces for small devices have traditionally been rather specialized—a cellular telephone needs a rather different user interface than a pager or a PDA. Also, it is generally rather difficult for academic researchers to work in this area because the interfaces of many of the small devices have been proprietary. Nevertheless, there have been some interesting proposals. For instance, Kamba et al [KEH96] have suggested the use of *partially overlapping, semi-transparent widgets* in order to utilize screen space more efficiently. Fitzpatrick et al [FPS98] have proposed the use of single line *ticker tapes* like those seen in stock market displays. Lieberman [Lie94] has provided some interesting insights into visualizing very large information spaces, and introduced the notion of *macroscoping*: the use of translucent layers in conjunction with zooming and panning.

In contrast with user interface proposals, there are a large number of proposals for new input techniques for small devices. For instance, new pen input approaches have been studied in [VeN94, Mas98, Per98]. Examples of new kinds of input devices can be found in [SuT96, KaI98].

3. New proposal: the event horizon user interface model

Imagine a flat, unlimited two-dimensional space that can store all kinds of objects typically found in the computer screens (such as icons, pictures, graphical elements, text, etc.) The user can move objects about the screen in a traditional fashion by using a mouse, pen or other pointing device.

In the middle of the screen there is a small circular area known as the "*event horizon*" (or more briefly: the "*sink*"). The event horizon serves as an unlimited container in which objects are stored when the display is "compressed" (see the description below).

Only one basic user interface navigation operation, known as the "*expand/compress*" function, is provided to the user. This function can be operated by using two simple buttons (such as the up and down buttons of the Palm connected organizer), or by using a simple rocking switch.

When the display is *compressed*, all the objects on the screen move radially closer towards the circular event horizon as long as the compress button is pressed (see Figure 2). Those objects that become close enough to the event horizon are "sucked in" the sink

and will disappear from the screen (hence the term "event horizon"—it operates like a black hole).

When the display is *expanded*, all the objects on the screen move radially farther away from the event horizon as long as the expand button is pressed (see Figure 3). At the same time, the event horizon will release the objects in it in the original order they were previously sucked in, making them visible again and restoring the original view.

During the "expand" operation, many objects may move beyond the physical limits of the computer screen. This is perfectly normal: objects can be brought back to the visible part of the screen later by using the "compress" operation again. In general, the expand/compress process is entirely reversible and repeatable. The compress operation allows the user to gradually fold complex, visual displays of information away from the screen altogether, storing them in the sink. Conversely, the expand operation can be used to make more space or to restore the view as it was before compression.

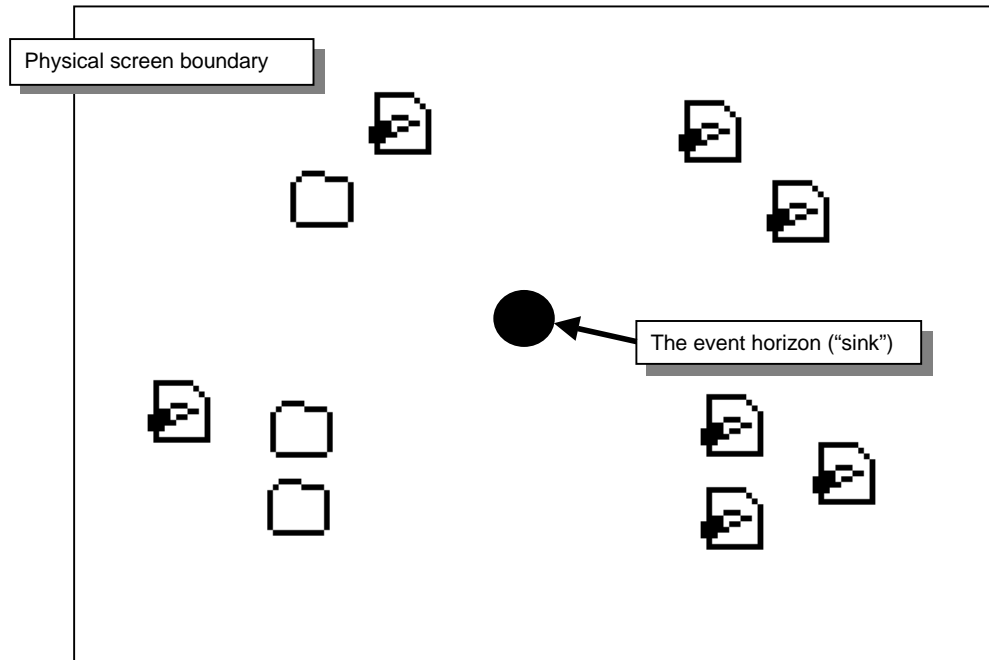


Figure 1. Initial example situation

The compress/expand operations are illustrated in Figures 1, 2 and 3. Figure 1 shows the initial display with ten visible objects. In Figure 2, the user has pressed the compress button, and all the objects have moved radially closer to the event horizon. One of the objects shown in Figure 1 has disappeared in the sink, and another object that was previously outside the visible screen area has emerged in the top right corner of the display. Notice that the dash-line arrows in Figure 2 and 3 are used for illustration purposes only to visualize the movement of objects, i.e., they are not part of the actual displays.

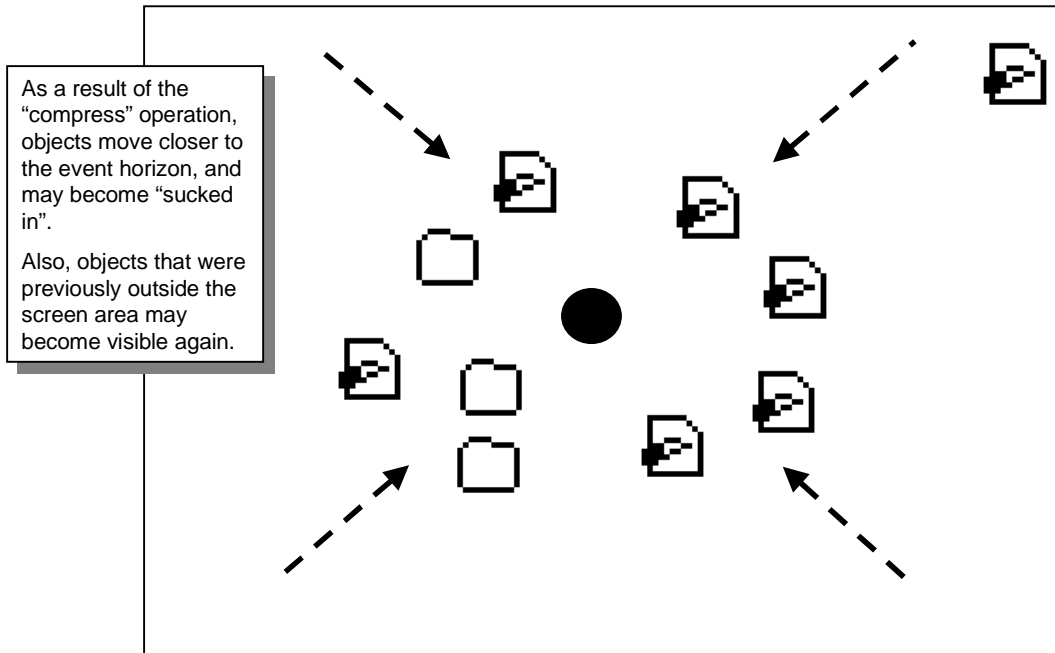


Figure 2. Using the compress operation

Figure 3 shows a situation in which the user has pressed the expand button while originally being in a situation depicted in Figure 1. All the objects have moved farther away from the sink, and a new, previously invisible object has emerged from the sink.

It is important to note that the compress/expand process is not an on/off operation, but it can be operated in a gradual fashion, sliding objects gradually closer and farther away from the sink. In a way, this is the equivalent of window scrolling in a traditional computer user interface, with the exception that objects are moved radially rather than horizontally or vertically. An important consequence of radial scrolling is that when objects are scrolled farther from the sink, the relative distance of individual objects increases. Conversely, when scrolling objects closer to the sink, the relative distance of objects decreases. This is something that does not happen in traditional scrolling. However, as with traditional scrolling, the relative positions of objects are always preserved, and all scrolling operations are always fully reversible and repeatable.

Benefits of the proposed approach. One of the best things about the proposed approach is that it allows an unlimited number of objects to be stored, visually organized and manipulated in a virtually large but physically very limited screen. The radial movement of objects ensures that the user will see all the objects stored in the sink while performing the expand operation to completion (i.e., until the sink has no more objects inside). Yet everything can be accomplished by using one-dimensional navigation only (compress/expand). This is different from traditional displays in which the user often has to scroll the display to two different directions (left/right and up/down) in order to find all the objects. In general, the proposed approach seems well suited to devices and situations in which the navigation operations have to be as simple and easy to operate as possible. Unlike traditional one-dimensionally scrolling displays such as list views, the user still

has the possibility of utilizing spatial memory and capitalizing on pragmatic and incidental cues for retrieval, such as “I left it over there”, “it was near the memo that I wrote last week”, or “it was the topmost object on the screen”.

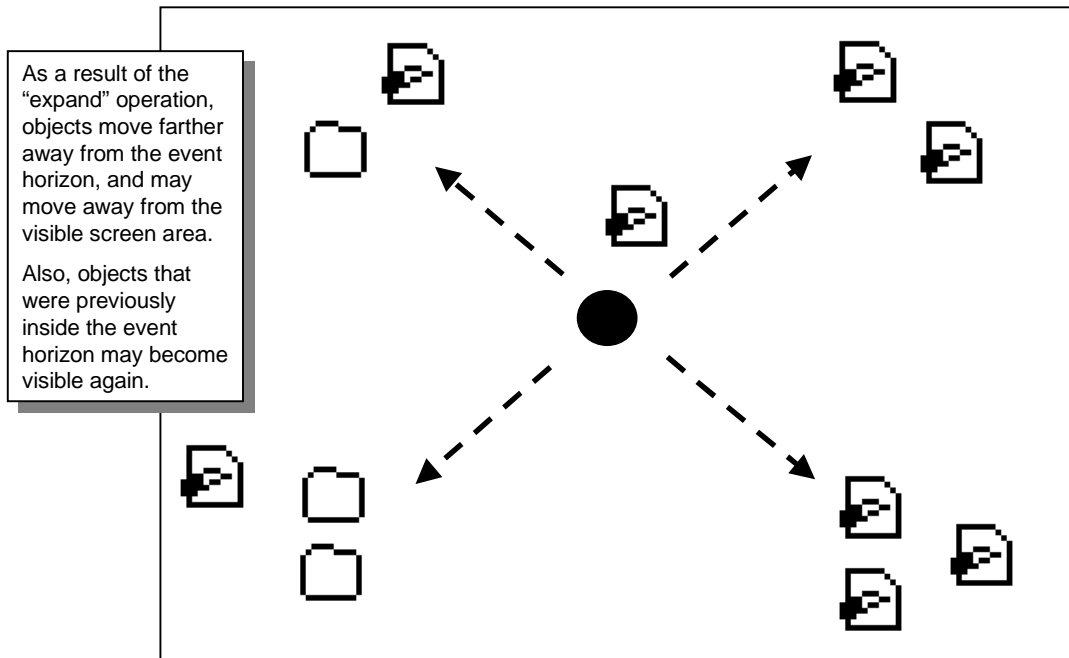


Figure 3. Using the expand operation

The proposed approach can be used in combination with other user interface concepts. For instance, in addition to simply having one sink for organizing objects, it is quite possible to have multiple hierarchically organized sinks. This approach has been used in our current implementation discussed in the next section. Also, it is quite obvious that other user interface operations, such as moving objects with a mouse or pen, can be supported as usual. In general, the proposed approach works well with persistent displays in which objects remain where the user places them. This principle allows the user to utilize spatial memory as described above.

An additional benefit of the proposed approach is that the distance of objects from the sink serves as an implicit notion of time: the “deeper” in the sink the objects are, the more likely it is that they are older than the objects located farther from the sink, or vice versa. For instance, it is quite possible to utilize the sink for the versioning of objects: when creating a new version of an object, the old version is stored away in the sink from where it can be restored later, if necessary.

From the implementation viewpoint an important advantage of the proposed approach is that its implementation does not require much computing power. For instance, unlike with zooming displays, there is no need for scalable fonts or other scaling operations. Implementation aspects of the proposed model will be discussed in more detail in the next section.

4. Implementation for the Palm connected organizer

We have built an implementation of the proposed user interface concept for the Palm connected organizer. The work was originally motivated by our work on *Spotless* [TBS99]—a Java™ virtual machine for small devices such as the Palm connected organizer. In that system one of the interesting design problems was to devise a solution that would allow a large number of classfiles to be managed on a small device with a limited screen. The event horizon concept proposed in this paper grew out of that work. After studying the event horizon model in more detail, we realized that the model could be used also for many other purposes, and we decided to build a more general-purpose implementation. The resulting system, *Radius*, replaces the standard *Memo Pad* note editor application of the Palm connected organizer, but it could be extended and used for other purposes as well.

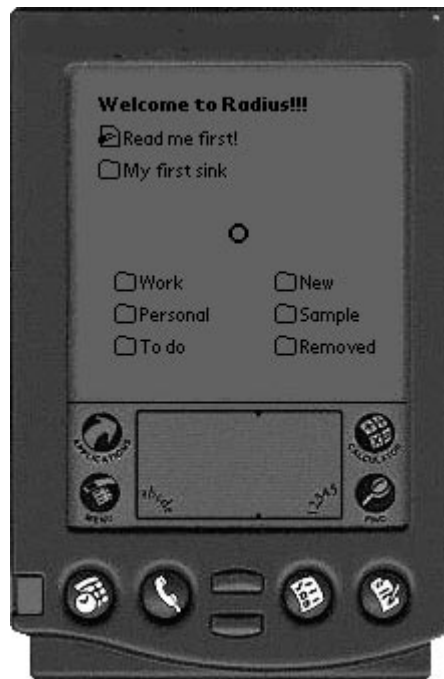


Figure 4. Radius running on the Palm connected organizer

The current Radius implementation for the Palm connected organizer is shown in Figure 4. As is characteristic of the event horizon model, in the middle of the screen there is a small circular icon representing the event horizon, surrounded by a number of objects created by the user. Apart from the event horizon icon, which is always located in the middle of the screen and which cannot be moved or removed, the user can select and move objects at will. The basic object selection, moving and editing capabilities are illustrated in Figures 5 and 6.

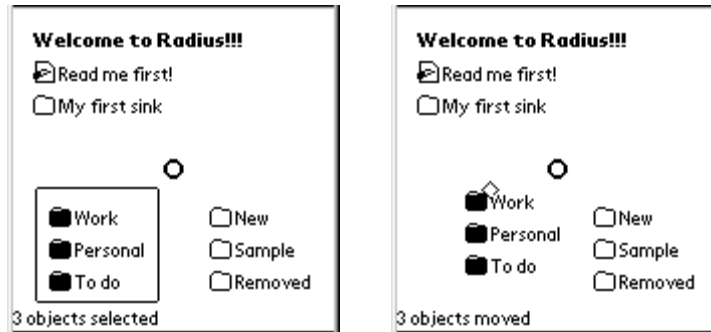
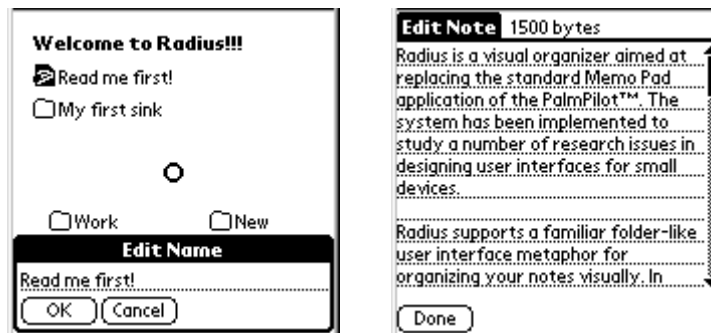


Figure 5. Object selection and moving



Figures 6a and 6b. Object name editing and note editing capabilities

The current implementation of Radius supports three user-level object types, illustrated in Figure 7. *Note* objects contain text that can be edited freely by the user. *Label* objects are one-line textual markers that can be added anywhere to provide further information to the user. Apart from their names, always shown in boldface, labels do not store any other information or have any other special meaning. *Sink* objects represent the subsinks of the currently visible sink. New sink objects can be created to build hierarchical sink structures. Creating subsinks is analogous to creating subfolders in a traditional graphical file system.

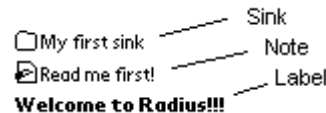


Figure 7. Radius object types

The names of objects can be edited by tapping the name of the object. A simple dialog box (illustrated in Figure 6a) will be opened to support name editing. By tapping the icon of a note object, the user can open a simple text editor (Figure 6b) to edit the note. By tapping the icon of a sink object, the user can open a subsink. The Radius system always shows the contents of only one sink at a time. The user can move up in the sink hierarchy by tapping the event horizon icon in the middle of the screen or by choosing the corresponding menu item.

Using the expand/compress functions. The expand/compress functions characteristic of the event horizon user interface model have been integrated tightly into the user interface of the Radius system. Normally the user operates these functions simply by pressing the *up* and *down* keys available on the Palm connected organizer (Figure 8). However, it is also possible to operate the functions by using menu operations. Both the expand and the compress operation can be applied either to all the objects in the current sink or to selected objects only.

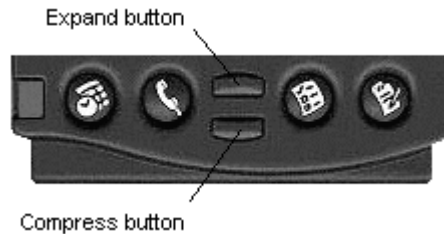


Figure 8. The Palm hard keys used for expand/compress functions

The use of the *expand* operation is illustrated in Figure 9. The picture on the left shows the initial situation. The picture on the right illustrates the situation after the user has applied the expand function a few times. As can be seen, all the objects on the screen have moved farther away from the event horizon, and there is now more room available in the middle of the screen. In this example, there were no objects inside the sink. Had there previously been objects inside the sink, those objects could potentially have become visible again.



Figure 9. Using the expand operation

The use of the *compress* operation is illustrated in Figure 10. We assume that the initial situation was the same as in the leftmost picture in Figure 9. The leftmost picture in Figure 10 shows the situation after the user has applied the compress function a few times. All the objects have moved closer to the event horizon, and some objects have already disappeared from the screen. The picture in the center shows the situation after the user has applied the compress function a few more times. Only one label is still visible, while the other objects have been sucked in the event horizon. Finally, the rightmost picture in Figure 10 shows a situation in which all the objects have disappeared inside the event horizon. Even though the display now looks radically different from the

initial situation, the user could restore the initial situation simply by applying the expand operation multiple times.



Figure 10. Using the compress operation

Note that the Radius system uses different icons to visualize the state of the current sink. In the current implementation, three different icons are used (Figure 11). A solid black circle indicates that all the objects in the sink are currently inside the event horizon. A “half-full” circle indicates that some of the objects in the sink are currently inside the event horizon. An “empty” circle indicates that there are currently no objects inside the event horizon.

- No objects inside the event horizon
- ◐ Some objects inside the event horizon
- All objects inside the event horizon

Figure 11. Icons for indicating sink status

In addition to regular expand and compress operations, the current Radius implementation also supports special functions called “expand fully” and “compress fully”. The *expand fully* function allows the user to rapidly unfold the sink and bring the innermost objects in the sink to the visible part of the screen. Conversely, the *compress fully* function allows the user to rapidly view the outermost objects of the sink, and pack the rest of objects into the sink.

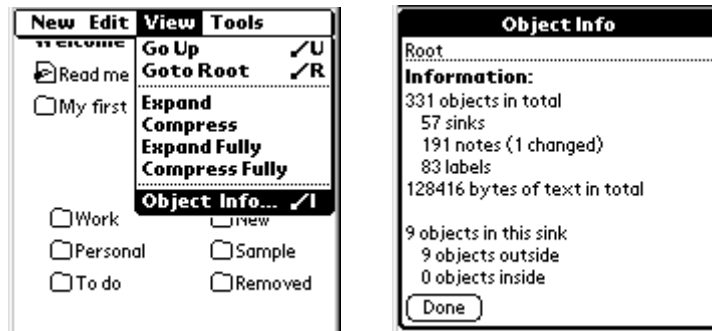


Figure 12. Object info display

Additional user interface features. The Radius system has an *object info* display that allows the user to view information about the current sink or selected objects (Figure 12). The information about the objects is calculated recursively starting from the current sink or selected objects, allowing the user to see the total number of sinks, notes and labels, as well as the total amount of text stored in these structures. In addition, the display also shows the number of objects that are currently located inside and outside the currently visible sink.

In order to provide compatibility with standard Palm OS applications, the Radius system includes *import and export capabilities* for moving data between Radius and the Memo Pad application. The import and export dialogs are shown in Figure 13. The Radius system keeps track of changed notes, so that it is possible to export only those notes that really have changed since the last export. A number of other options for importing and exporting data are provided.

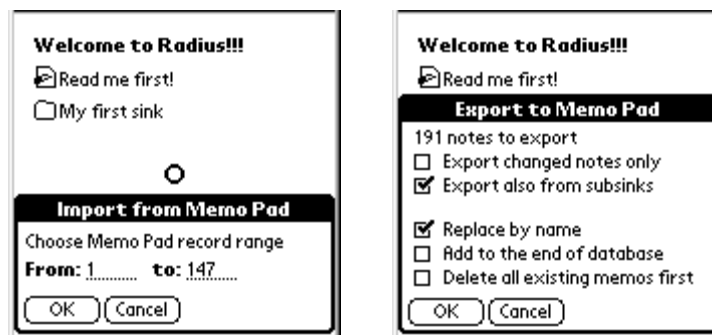


Figure 13. Import and export capabilities

Implementation. The Radius system has been implemented in C/C++ using the Metrowerks development environment for Palm OS. The source code base contains about 9200 lines of well-commented code, and the executable size is currently approximately 34 kilobytes. The implementation utilizes the Palm OS database manager, graphics APIs and event handling mechanisms extensively, but the overall application architecture is intended to be portable across a number of devices and platforms.

The Radius user interface implementation is based on the Model-View-Controller (MVC) design paradigm familiar from the Smalltalk-80 programming environment [GoR83]. Event handling (Control) has been separated from display management (Views) and database management (Model). The main controller utilizes a simple state machine for implementing drag-and-drop support, object selection and other features on top of the Palm event handling mechanisms.

With the exception of a few temporary variables, everything in the Radius system is a persistent object. A simple persistent object storage system has been implemented on top of the Palm OS database manager. The database supports two basic kinds of objects: primitives and application objects. Primitive data types include strings, simple object lists, and arbitrary data blobs. Application data types include sinks, notes and labels. Application objects are composed using the primitive types, and objects refer to each

other simply using Palm database record numbers. A number of predefined objects in the database have been created for persistently storing session information, object selection information and some user interface option flags.

5. Experiences

The Radius application has been in everyday use by a number of users for several months now. The application has generally proven to be useful and reliable. Notably the database part of the system has been very stable. The largest databases used so far have contained several thousands of objects and hundreds of notes storing hundreds of thousands of bytes of text. During the development of Radius, only a few small changes to the database structure were made after the initial version of the system was created.

The user interface of the Radius system has undergone some evolution and refinement. For instance, in the first versions of Radius it was very difficult to arrange objects on the screen in an aesthetically pleasant fashion. This was because the pen coordinates returned by the Palm OS are not 100% precise, but tend to vary by a few pixels even when holding the pen in a steady position on the screen. The problem was fixed by implementing an auto-align option using an invisible 8x8 pixel grid: Whenever the user moves an object, the system will automatically place the object at the nearest location on the grid if this option is turned on.

One potential problem of the event horizon user interface model is the behavior of objects near the event horizon. For instance, what should happen when a number of objects come close to the event horizon and start overlapping each other, or when objects have long names that reach from the left side of the screen to the right side of the screen? After exploring a number of solutions, such as sucking objects in “partially” or utilizing simple animation and sound effects to denote the proximity of the event horizon and collisions of objects, we decided to keep the implementation simple. When objects come close enough to the event horizon, they simply disappear. The possible problems in selecting overlapping objects are avoided by drawing the objects in the reverse order compared to the selection order of objects, i.e., those objects that are drawn last will always be searched first when selecting objects.

The solutions described above work well, and require no extra computing power when drawing the objects—this is something that is very important on a low-power device such as the Palm connected organizer. In spite of the simplicity of the implementation, the performance of the compress and expand operations can still be inadequate with large sinks that contain hundreds of objects. The Palm device simply does not have enough computing power to process hundreds of database write operations very rapidly.

From the implementation viewpoint one of the trickiest things during the development of Radius was to implement the radial movement efficiently in terms of both speed and space. We wanted to avoid the use of floating point operations because these operations

are not readily available on many small devices. For this reason, a simple integer trigonometric package consisting only of a few short data tables and functions was written. For efficiency, we also created a simpler implementation that moves objects in a 45° angle instead of truly radial movement. The resulting implementation is simple and efficient with no coordinate granularity problems.

The Palm connected organizer proved to be a good platform for prototyping new user interface concepts such as the event horizon model. The initial version of the Radius system was completed in just a few weeks. This is partly because of the availability of good development tools for the Palm platform but more importantly because of the easy-to-use graphics and database support that have been built in the Palm OS. The built-in database support makes it very easy to implement persistent applications—compared to full-fledged personal computers which typically require an external, heavyweight database system in order to be able to support persistent objects. The square screen of the Palm device was also a perfect fit for the event horizon user interface model.

6. Conclusions

In this paper we have presented a new *collapsible* user interface model for displaying and managing information on small computing devices. The new model is built around the idea of an *event horizon*: a circular area in the center of the screen into which all the other objects on the screen can be stowed in order to make more screen space available. Unlike traditional user interfaces in which the user typically has to scroll in at least two directions in order to see all the objects, the event horizon model requires only one navigation dimension, making the model particularly well suited to devices that need to be operated with one hand. The presented technique is general, and can be used in combination with traditional user interface techniques such as the desktop metaphor and hierarchical folders. Our current implementation of the event horizon model for the Palm connected organizer was presented, followed by a brief discussion on our experiences with the model.

References

- Bal94 Ballay, J.M., Designing Workspace™: an interdisciplinary experience. In *Proceedings of the CHI'94 Conference* (Boston, Massachusetts, April 24-28), 1994, pp.10-15.
- BeH94 Bederson, B.B., Hollan, J.D., Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology UIST'94* (Marina del Rey, California, November 2-4), 1994, pp.17-18.
- BSH94 Bederson, B.B., Stead, L., Hollan, J.D., Pad++: Advances in multiscale interfaces. In *CHI'94 Conference Companion*, ACM Press, 1994, pp.315-316.
- BSP94 Bier, E.A., Stone, M.C., Pier, K., Fishkin, K., Baudel, T., Conway, M. Buxton, W., DeRose, T., Toolglass and magic lenses: the see-through interface. In *CHI'94 Conference Companion*, ACM Press, 1994, pp.445-446.
- BrS95 Brown, M.H., Shillner, R.A., Deckscape: an experimental web browser. In *Proceedings of The Third International World-Wide Web Conference: Technology, Tools and Applications* (Darmstadt, Germany), April 1995.
- BrS96 Brown, M.H., Shillner, R.A., The DeckScape web browser. In *CHI'96 Conference Companion*, ACM Press, 1996, pp.418-419.
- CRY96 Card, S.K., Robertson, G.G., York, W., The WebBook and the Web Forager: an information workspace for the World-Wide Web. In *Proceedings of the CHI'96 Conference* (Vancouver, B.C., Canada, April 13-18), 1996, pp.111-117.
- FFG96 Fertig, S., Freeman, E., Gelernter, D., Lifestreams: An alternative to the Desktop metaphor. In *CHI'96 Conference Companion*, ACM Press, 1996, pp.410-411.
- FPS98 Fitzpatrick, G., Parsowith, S., Segall, B., Kaplan, S., Tickertape: awareness in a single line. In *CHI'98 Conference Summary*, ACM Press, 1998, pp.281-282.
- Fox98 Fox, D., Composing magic lenses. In *Proceedings of the CHI'98 Conference* (Los Angeles, California, April 18-23), 1998, pp.519-525.
- Fur86 Furnas, G.W., Generalized fisheye views. In *Proceedings of the CHI'86 Conference*, 1986, pp.16-23.
- GoR83 Goldberg, A., Robson, D., *Smalltalk-80: the language and its implementation*. Addison-Wesley, 1983.
- Gre96 Greenberg, S., Fisheye text editor for relaxed-WYSIWIS groupware. In *CHI'96 Conference Companion*, ACM Press, 1996, pp.212-213.
- GGR96 Gutwin, C., Greenberg, S., Roseman, M., Workspace awareness support with radar views. In *CHI'96 Conference Companion*, ACM Press, 1996, pp.210-211.
- GFC98 Gwizdka, J., Fox, M., Chignell, M., Electronic engineering notebooks: a study in structuring design meeting notes. In *CHI'98 Conference Summary*, ACM Press, 1998, pp.355-356.

- HeC86 Henderson, D.A., Card, S.K., Rooms: The user of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics* vol 5, nr 3 (Jul) 1986, pp.211-243.
- HGD95 Hovestadt, V., Gramberg, O., Deussen, O., Hyperbolic user interfaces for computer aided architectural design. In *CHI'95 Conference Companion*, ACM Press, 1995, pp.304-305.
- Joh92 Johnson, B., Treemap visualization of hierarchically structured information. In *Proceedings of the CHI'92 Conference* (Monterey, California, May 3-7), 1992, pp.369-370.
- KEH96 Kamba, T., Elson, S.A., Harpold, T., Stamper, T., Sukaviriya, P., Using small screen space more efficiently. In *Proceedings of the CHI'96 Conference* (Vancouver, B.C., Canada, April 13-18), 1996, pp.383-390.
- KaI98 Kawachiya, K., Ishikawa, H., NaviPoint: An input device for mobile information browsing. In *Proceedings of the CHI'98 Conference* (Los Angeles, California, April 18-23), 1998, pp.1-8.
- LaR94 Lamping, J., Rao, R., Laying out and visualizing large trees using a hyperbolic space. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology UIST'94* (Marina del Rey, California, November 2-4), 1994, pp.13-14.
- LRP95 Lamping, J., Rao, R., Pirolli, P., A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the CHI'95 Conference* (Denver, Colorado, May 7-11), 1995, pp.401-408.
- LaR96 Lamping, J., Rao, R., Visualizing large trees using the hyperbolic browser. In *CHI'96 Conference Companion*, ACM Press, 1996, pp.388-389.
- Lie94 Lieberman, H., Powers of ten thousand: navigating in large information spaces. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology UIST'94* (Marina del Rey, California, November 2-4), 1994, pp.15-16.
- LuS94 Lucas, P., Schneider, L., Workspace: A scriptable document management environment. In *CHI'94 Conference Companion*, ACM Press, 1994, pp.9-10.
- Mas98 Masui, T., Efficient text input method for pen-based computers. In *Proceedings of the CHI'98 Conference* (Los Angeles, California, April 18-23), 1998, pp.328-335.
- MSW92 Mander, R., Salomon, G., Wong, Y.Y., A 'pile' metaphor for supporting casual organization of information. In *Proceedings of the CHI'92 Conference* (Monterey, California, May 3-7), 1992, pp.627-634.
- Per98 Perlin, K., Quikwriting: continuous stylus-based text entry. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology UIST'98* (San Francisco, California, November 1-4), 1998, pp.215-216.
- PMR96 Plaisant, C., Milash, B., Rose, A., Widoff, S., Shneiderman, B., LifeLines: Visualizing personal histories. In *Proceedings of the CHI'96 Conference* (Vancouver, B.C., Canada, April 13-18), 1996, pp.221-227.
- RCJ92 Rao, R., Card, S.K., Jelinek, H.D., Mackinlay, J.D., Robertson, G.G., The information grid. In *Proceedings of the 5th Annual ACM Symposium on User Interface Software and Technology UIST'92* (Monterey, California, November 15-18), 1992, pp.23-32.

- RaC95 Rao, R., Card, S.K., Exploring large tables with the table lens. In *CHI'95 Conference Companion*, ACM Press, 1995, pp.403-404.
- RoM93 Robertson, G.G., Mackinlay, J.D., The document lens. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology UIST'93* (Atlanta, Georgia, November 3-5), 1993, pp.101-108.
- RMC91 Robertson, G.G., Mackinlay, J.D., Card, S.K., Cone trees: animated 3D visualizations of hierarchical information. In *Proceedings of the CHI'91 Conference* (New Orleans, Louisiana, April 27 - May 2), 1991, pp.189-194.
- RCL98 Robertson, G.G., Czerwinski, M., Larson, K., Robbins, D.C., Thiel, D., van Dantzich, M., Data Mountain: using spatial memory for document management. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology UIST'98* (San Francisco, California, November 1-4), 1998, pp.153-162.
- RoG96 Roseman, M., Greenberg, S., TeamRooms: network places for collaboration. In *Proceedings of the 1996 ACM Conference on Computer Supported Collaborative Work CSCW'96* (Boston, Massachusetts, November 16-20), 1996, pp.325-333.
- SaB92 Sarkar, M., Brown, M.H., Graphical fisheye views of graphs. In *Proceedings of the CHI'92 Conference* (Monterey, California, May 3-7), 1992, pp.83-91.
- Shn83 Shneiderman, B., Direct manipulation: a step beyond programming languages. *IEEE Computer* vol 16, nr 8 (Aug) 1983, pp.317-329.
- Smi86 Smith, R.B., The Alternate Reality Kit. In *Proceedings of the IEEE Workshop on Visual Languages* (Washington, D.C.), 1986, pp.99-106.
- Smi87 Smith, R.B., The Alternate Reality Kit: an example of the tension between literalism and magic. In *Proceedings of the CHI'87 + GI'87 Conference* (Toronto, Canada, April 5-9), 1987, pp.61-67.
- SHH98 Smith, R.B., Hixon, R., Horan, B., Supporting flexible roles in a shared space. In *Proceedings of the 1998 ACM Conference on Computer Supported Collaborative Work CSCW'98* (Seattle, Washington, November 14-18), 1998, pp.197-206.
- Sta93 Staples, L., Representation in virtual space: visual convention in the graphical user interface. In *Proceedings of the INTERCHI'93 (CHI'93 + INTERACT'93) Conference* (Amsterdam, The Netherlands, April 24-29), 1993, pp.348-354.
- SuT96 Sugimoto, M., Takahashi, K., SHK: Single hand key card for mobile devices. In *CHI'96 Conference Companion*, ACM Press, 1996, pp.7-8.
- TaV97 Taivalsaari, A., Vaaraniemi, S., TDE: Supporting Geographically Distributed Software Design with Shared, Collaborative Workspaces. In Olivé, A., Pastor, J. (eds): *Proceedings of the 9th Advanced Information Systems Engineering Conference CAiSE'97* (Barcelona, Spain, June 16-20), LNCS 1250, Springer-Verlag, 1997, pp.389-408.
- TBS99 Taivalsaari, A., Bush, B., Simon, D., The Spotless system: Implementing a Java™ system for the Palm connected organizer. Sun Microsystems Laboratories Technical Report SMLI TR-99-73, February 1999.
- Twe97 Tweedie, L., Characterizing interactive externalizations. In *Proceedings of the CHI'97 Conference* (Atlanta, Georgia, March 22-27), 1997, pp.375-382.

- UnS87 Ungar, D., Smith, R.B., Self: the power of simplicity. In Meyrowitz, N. (ed): *OOPSLA'87 Conference Proceedings* (Orlando, Florida, October 4-8), ACM SIGPLAN Notices vol 22, nr 12 (Dec) 1987. pp.227-241.
- VeN94 Venolia, D., Neiberg, F., T-Cube: A fast, self-disclosing pen-based alphabet. In *Proceedings of the CHI'94 Conference* (Boston, Massachusetts, April 24-28), 1994, pp.265-270.
- WaL95 Ware, C., Lewis, M., The DragMag image magnifier. In *CHI'95 Conference Companion*, ACM Press, 1995, pp.407-408.
- WLS98 Woodruff, A., Landay, J., Stonebraker, M., Goal-directed zoom. In *CHI'98 Conference Summary*, ACM Press, 1998, pp.305-306.

About the Author

Dr. *Antero Taivalsaari* is a staff engineer at Sun Microsystems Laboratories in Mountain View, California. His research interests include object-oriented programming and design, implementation of interactive programming languages, collaborative systems, mobile and wireless systems, and prototype-based programming. He has published a number of research papers in international journals and conferences, given invited lectures on various topics, and has organized several international workshops in the field of object-oriented programming.

Before joining Sun Labs in August 1997, Antero worked as a research manager at Nokia Research Center in Helsinki, Finland, where he lead a research group working on an advanced collaborative software design environment and managed some of Nokia's international research projects. Antero's doctoral thesis, entitled "A critical view of inheritance and reusability in object-oriented programming", received the award for the best doctoral thesis in computer science in Finland in 1994.