The Experience of Computation

Paul Dourish
Information and Computer Science/Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
jpd@ics.uci.edu

From Usability to Experience

Moore's Law sets the pace for computational life. Originally noted in 1964 by Intel founder Gordon Moore, it observes that the achievable density of components on a silicon substrate doubles roughly every 18 months. It has many corollaries and alternative formulations. One is that computers get twice as fast every eighteen months; another is that they get half as expensive. My personal favorite interpretation is that the total of all the advances in computer power since I was born (or you were born, or man first walked on the moon) will be exceeded by the progress of the next eighteen months. Although Moore's Law is conventionally regarded as a technical observation, its implications are much broader. The exponential advances in computer technology that we experience quickly mount up to changes not just in number, but in kind; computers become not just be cheaper and more plentiful, but become quite different kinds of things.[1] Computers reinvent themselves continually, and as a consequence, our expectations of them – of when and where and how we will encounter them, of what they will be expected to do, and of how we expect to be able to incorporate them into daily life – are also continually changing. So it's unsurprising that even in its short history, Human-Computer Interaction (HCI) has also undergone repeated changes in emphasis and approach. Early concern with input devices and display technology gave way to explorations of theories of cognition and action; in turn, once those became systematized, we became more concerned with the development of generic user interface technologies, and then with design approaches that would allow us to keep up with the pace of technical development.

One of the more interesting changes in recent years has been the rising influence of design, and the emergence of Interaction Design as one of the central strands within HCI practice. Design has always been an important part of HCI, of course, with designers represented within both research and development organizations, but in the past few years, it has taken on a much more prominent role. High-profile designers have played a more active role in conferences like CHI (e.g. John Thakara's keynote at CHI 2001 in Den Haag); courses and books incorporating the idea of Interaction Design have become more popular (e.g. Preece, Rogers and Sharp, 2002); the DIS (Designing Interactive

[1] Consider, as an example, that multimedia and networking facilities are relatively recent additions to the standard PC, while today, a computer that wasn't capable of connecting to the Internet and of playing MP3s and DVDs would be regarded as almost useless. The computer has become a network terminal and a multimedia hub.

Systems) conferences have become more firmly established as a major venue for research presentations, one that fuses technical and design concerns; and the manifestations of design on Web sites and in new interfaces such as Apple's Aqua (in MacOS X) and Microsoft's newer Windows XP design have become important criteria for selling and distinguishing between systems.

One of the reasons to draw attention to the increasing prominence of design in HCI is that it marks a departure from some earlier conceptions of the role of HCI and interactive technology. While there are many different schools of design and different perspectives on its role, one thing that most would agree on is that the incorporation of design is a move beyond a traditional narrow focus usability. Where the notion of "usability" evokes images of experimental evaluation, laboratory settings, and formal metrics of effectiveness, design addresses the much richer concerns of products in use. Designers are concerned not simply with how people use products, but also with what they experience in the process; they are concerned with the meaning that those products and interactions hold. From this perspective, usability is a critically important criterion, but the only way to understand usability is to place it within this wider frame of the meaning that technology holds and experience that it affords.

Experience and Practice

It is easy to dismiss this perspective as frivolous – to suggest, perhaps, that it's all "marketing hype" and that such issues as experience are merely ways to distinguish one product from another, irrelevant to the foundational concerns of technology in use. However, we know from other studies of technology in real settings that related issues of meaning and experience are critically important to successful design and adoption. Usage studies point out that individuals and groups find new ways to use technology that the designers had never intended; they find ways to make the technology useful to themselves by developing new sorts of practice and new ways of incorporating technology into that practice. The success of a technology depends not least on how well it can be incorporated into the practice of those who use it, and how well the technology and the practice can coevolve. The relevance of this here is that *practice* and *meaning* are intimately interconnected.

Etienne Wenger suggest that "practice is, first and foremost, a process by which we can experience the world and our encounters with it as meaningful." (Wenger, 1998:51). Wenger's concern is with "communities of practice," the social groups whose common experience of action provides a framework for shared meaning. As an illustration, consider an apprentice shoemaker. On his first day in the workshop, he finds himself surrounded by leathers of different sorts, but he lacks the skills to distinguish between them except on broad levels such as size, color, or flexibility. Over time, as he learns to use the tools of the trade, and as he learns to work the leather, he learns how to "read" the leather in new ways. He learns how to experience leather as amenable to certain sorts of operations, workable with certain sorts of tools and techniques, or holding the capacity to be used in different kinds of shoes or products. In other words, learning the practice of shoemaking means learning how to encounter the world as meaningful for that practice. This isn't unique to shoemaking, by any means. Sculptors sometimes talk of "releasing" a sculpture from the rock (rather than carving it); and part of the process of teaching computer scientist is training students to see the world as computer scientists see it, as amenable to certain forms of representation and proceduralization (teaching them to see algorithms, iteration, and encapsulation.) The relationship between practice and meaning is universal.

So, the concern with meaning and experience in use that design approaches draw our attention to are relevant not just for web pages or physical artifacts, but for all aspects of information systems. Further, these topics are not niceties or factors that supplement the basic stipulations of “usability engineering” – they go to the heart of what it means for a system to be usable in any practical sense. Experience, meaning, and practice are central to interactive system development and use.

Two Problems: Fragmentation and Alienation

It should be heartening, then, to see design take a more prominent place in HCI, and indeed it is. However, the effectiveness of this attempt to give users a richer experience in using technology is potentially limited by being conducted at the wrong level. As HCI designers, we tackle design and user experience problems largely at the level of applications and frameworks – the design of specific web pages, specific interactive widgets, or specific applications. Further, in doing these, we look outside of the computer to find metaphors to make these experiences meaningful. The result is a cacophony of concepts presented side-by-side on a single computer screen. At any given moment, a user might have to contend with “windows,” “palettes,” “menus,” “icons,” three-dimensional objects that stretch and twist, hourglasses, and even faux street signage all present on the screen at the same moment.

These metaphors have been designed to make the technology easier to for us to use and make sense of, by drawing analogies between technical practice and the everyday world. They encourage us to think of interactive systems not as pieces of software, but as artifacts that embody aspects of desktops, spatial environments, social institutions (in the case of the traffic signs) and the everyday physical environment. In doing so, though, they introduce some other difficulties. I will briefly suggest two here.

The first is *fragmentation*. The nature of the way that we design computer systems today – the structure of interactions between operating systems, modules, and applications – is forced on us by a combination of technical necessity and historical precedent. It makes it possible to harness human resources to build large-scale technical artifacts under the constraints that operate on most software development exercises. On the other hand, though, the separation of software systems into different components, developed by different (groups of) people, subject to different sorts of demands, requirements, and interpretations of need, working under different organizational strictures, makes it hard to manifest a system-wide coherence of interaction. Application development efforts are largely independent, and while software platform providers can do their best to promulgate standards, conventions, and design guidelines to promote consistency across these different applications, an absolute consistency is impossible to achieve.

The second difficulty, and the more serious here, is what I will call *alienation*. When faced with the tough challenge of creating a user interface to a new software system, designers typically look outside of the computer system itself to find models and metaphors with which to make the system seem more familiar and natural. While this offers a fruitful model for "first-time" or "walk-up" use, the effect over time is to prevent or interfere with the user gaining a deeper understanding of how the computer system is operating in response to their actions. The level at which the computer system is experienced is deliberately and systematically distanced from the level at which it actually operates. In addition, through the process of abstraction, many quite different technical phenomena may be presented through the same interface (such as the ubiquitous "folder" that might represent a floppy disk, part of a fixed disk, a digital camera, or a networked server hundreds of miles away – each of which, clearly, have very different characteristics). This barrier frustrates the user's attempt to understand what is going on, and to relate the system's response (or lack of response) to the actions that they take. Even so-called "naïve" users, or those who have no interest in finding out how computers work, nonetheless do form models of causality that furnish them with the basic means to interpret the system's behavior. However, the accuracy of these models is limited by the way that conventional design systematically alienates users from the technology that acts on their behalf.

FORMULATING AN AGENDA

These problems, fragmentation and alienation, have elements in common. Essentially, they are each the consequence of barriers erected within computer systems. Where fragmentation arises because of the barriers between different parts of the system, alienation is a product of the barriers that are placed between users and the underlying computational medium. Further, they have a number of consequences for the questions of meaning and experience as central elements in interactive system design. Drawing on social studies of technology use, it was suggested that successful system adoption depends on being able to fit technology into working settings. Although this is a widely recognized concern which has motivated a great deal of work on user-centered design and on workplace studies to inform system development, what I want particularly to point out here is the dynamic nature of this fit. Technology must fit into working settings, but working settings and forms of practice evolve over time, and technologies coevolve alongside. Studies such as those of Mackay (1990) have illuminated the coadaptive relationship between technology and practice, and have emphasized the importance of understanding how technologies can be made amenable to this sort of adaptation and appropriation. The question to be asked, then, is not simply, "how can we design technology to fit the working practice that we have observed?" but rather, "what properties of computer systems allow them to support the emergence and continual evolution of practice?"

Elsewhere, I have termed this continual integration into practice "appropriation" (Dourish, in press). Appropriation of technology, in this sense, is not simply tailorability and customization, but also the emergence of new working practices that arise around technology in use, based on evolving understandings of its capabilities and potential. We see it not only in explicit customization, but also in the emergence of conventions by which new meanings can be created through differentiation and repetition.[2] Technologies that lend themselves to this sort of adaptation and appropriation are not only ones that are malleable, but ones that are translucent, allowing users to see the consequences of their actions, and understand how those actions can be transformed to yield different results. Flexibility must be accompanied by reversibility, directness, and accountability (Dourish, 1995), allowing for continual adaptation, not as a separate activity (e.g. programming), but seamlessly woven into use.

The vision this suggests is of a fluid, flexible computational medium, malleable and responsive enough to allow the development of new structures and practices. At a high level, this is similar to the "tool paradigm" for interactive systems – the idea that a system should present itself to users as a tool without overly constraining how the tool is to be used. On a more detailed level, though, this approach breaks with the traditional interpretation of the tool paradigm by focusing at the level of computation rather than applications. The tool paradigm has normally been realized at the level of applications, giving us tools for reading, for writing, and for communicating. However, the problem of alienation suggests that we need to go further – not just to mimic existing tools, or to create new sorts, but to recognize the inherent nature of *the computer as a tool.* Computation is representational, is active, is responsive. A solution to the problems of fragmentation and alienation will depend on giving people *an experience of computation.*

What are some of the properties of interactive systems that give an experience of computation?

Perceptual. Directness is a key property for this approach. Our goal is to support embodied interaction, where embodiment denotes a sense of participation and presence (Dourish, 2001). An important design concern here is to make information directly available and perceptible to the user, without requiring indirect manipulation or interpretation. Much of our current work (see below) is based on visual approaches, which aim to move processing from the cognitive to the perceptual system (Robertson et al., 1993), relying on what has been called "external cognition" (Scaife and Rogers, 1996)

Continuous input and output. An important characteristic of our experience of the world is that it is continual and ongoing. The world around us may fade into the background as we concentrate on some task, but it remains there, available and accessible. The world is always available for interaction, and always presents itself directly. Similarly, we believe that an important building block for giving an experience of computation is continual input and output. Instead of building systems around a model of commands and results, it will be necessary to build them around a model where users shape the unfolding of computational behaviors, continually changing, revising, shaping and directing them.

[2] A simple example is the use of README files; everyone knows what sort of information is likely to be in a README file, and where it is likely to be found. The README file is meaningful to certain sets of system users even though these "semantics" are never encoded in the filesystem. It is a convention that arises amongst people who understand facets of each other's work and can recognize how the system will appear to them (and has, in the process, lost the sense of its original reference to "Alice in Wonderland".)

Concrete. The direct interaction style envisioned by this proposal leads to a concrete design style. By this, I mean that the entities affecting interaction should be directly available and accessible within the interaction. Rather than providing two worlds – one world of objects of attention, and one of abstractions that describe them – the design style is to present an integrated whole. Philosophers of language talk about the use/mention distinction – the difference between saying "hello" (use) and referring to the word "hello" (mention). So, one important characteristic that supports a direct experience of computations is that it integrates the two.

We present these characteristics here to give a flavor of the main idea, rather than as a definitive list of necessary or adequate conditions. They are intended to help sketch the outlines of the design space. To this same end, it is valuable to look at some current research activities that address some of the same questions.

Some Examples

These are broad, abstract ideas. To try to see how they might work out in practice, it's useful to look at some specific examples. I will introduce four current activities: *visualizing the behavior of software systems*, *visualizing security*, *populating the social workscape*, and *ad-hoc and emergent information management*. Each of the projects described here is ongoing work, and expresses some part of the approach outlined here. None is intended as an ideal instantiation; rather, they are designed to explore some of the issues outlined above.

Visualizing the Behaviour of Software Systems

One of the centrally important features of computation is that, in its embodiment in conventional computers, it is *active* – it can do things for us. If it could not, there would be little point in using computers. However, what it's doing, and how, is typically not information that's available to us as users of computers. Interfaces are constructed in terms of requests and results, but not in terms of the mechanisms necessary to turn one into the other.

The reasons for this are simple and obvious (although not so obvious that HCI practitioners don't have a hard time explaining them to people sometimes). Most people, most of the time, don't want to "use computers" – they want to get their work done. Using the computer is simply a means to an end. They have no desire to be forced to understand the intricacies of the computer system design merely to print a letter, send an email message, or perform a mathematical analysis, any more than they would be interested in learning the physics of the internal combustion engine in order to get to work in the morning. This is a basic principle behind much of HCI, but it hides a more complex observation. Although nobody wants to learn about internal combustion in order to get to work, most of us do have available to us a set of cues that let us assess how our cars are performing – the sound (or, if we're unlucky, the smell) of the engine, the feel of the steering wheel or the clutch, or the whizzing of the scenery in our peripheral vision. We may not be engine mechanics, but we have a practical understanding of how cars work that guides our own activity. The sound of the engine straining prompts me to change gear; the feel of the steering in a turn prompts me to give the engine more gas. In general, we act everyday in the world as we find it, available to use for practical action, organized in ways that are more or less predictably meaningful. The world is available to us for examination and interpretation, even if the interpretations that we make are flawed or incorrect. Notably, this is not true of most computer interfaces; organized around requests and responses, they provide no insight into the operations that came in between. Web search engines exemplify this paradigm; you enter a query, a results page is returned, and it's up to you to figure out how the results are meant to be related to your request, and what to do about it (Muramatsu and Pratt, 2001).

One of our recent projects has been exploring the extent to which we can produce real-time visual representations of executing programs. An important criterion is to be able to visualize programs without making any changes to them. We have been developing VaVoom, a "visual virtual machine" for Java programs, which takes unmodified Java class files and executes them, producing a simultaneous visual representation of the program as it runs (Dourish and Byttner, 2002). Various aspects of the program's behavior are displayed in real-time alongside the program's execution, and these visual representations can be dynamically controlled and explored. At any moment, the user can shift attention from one representation to another, or display multiple different representations concurrently to observe any correlations in their behavior.

As a first step and proof-of-concept, the initial target is to be able to produce representations that give novice programmers insight into the execution of their programs. Our informal experiences working with programmers has been positive, and indeed the system has been more effective than we had hoped at allowing people to see external manifestations of otherwise purely internal behavior. However, aiming these presentations at programmers allows us to rely on their internal models of software construction, which, although often flawed, are clearly much more sophisticated than would be those of end users. Our intuition is that sufficiently carefully design visual displays (and, in particular, coupled visual displays that maintain temporal coherence) can provide people with meaningful interpretations of information that would otherwise be much lower-level than you would expect to make sense (much like the sound of the car engine is a representation of information about engine timing that is over my head); however, the representations that we have been working with so far are definitely too low-level for effective engagement with end users. This work is a first step, though, and the approach seems to have promise.

Visualizing Security

To use this visualization approach with end-users, our approach is to look for specific, focused application areas in which we can build specialized visualizations. This will allow us to better formulate design parameters for these active visualizations. One especially promising area that we are beginning to explore is security in networked applications.

Much work over the past few years has been directed towards the goal of providing reliable and robust mechanisms for implementing secure computer communications. The use of strong cryptographic technology allows mathematically provable security guarantees to be made about specific technical arrangements. However, practical security has proven to be an elusive goal for at least two reasons: first, that the nature of these mathematical guarantees is hard for people to understand, and second, that secure infrastructures are inherently brittle, requiring users to pay attention to myriad details in order to maintain them. The irony here is that absolutely, mathematically-provable strong cryptography is often more than people need. End users do not ask, "is this system secure?" but rather, "is this system secure *enough* for the task at hand?" Different tasks – sending an email message to a friend, submitting grades, executing a financial transaction – require different degrees and different sorts of security guarantees. In the everyday world, we might assess our immediate security by examining our surroundings, or by testing doors and windows. Computer systems tend not to provide users with the resources they need to make an informed decision about the adequacy of available security configurations and mechanisms.

Again, we take a visualization-based approach to this problem (Dourish and Redmiles, 2002). The infrastructure we are developing rests on three foundations. One is a distributed event routing and monitoring architecture. This allows us to gather information from multiple system and network components, including being aware of activities in the user interface. The second is a visualization engine that can provide graphical depictions of security-relevant system actions. As in the previous case, the critical feature of these visualization is that they are provided dynamically, along with the system action that they describe; these are not intended as probes or analysis tools, but rather as ways of making system and network activity visible and manifest to users as part of the computational experience. The third component that links these two together is a set of security-specific interpretation agents, which use heuristic reasoning to attempt to determine the security implications of particular configurations of technology. For instance, observing that an encryption technology such as *stunnel* or *ssh* is currently running, and is targeting internet port 25 (assigned to the SMTP mail transport protocol) allows an agent to determine that email may be being tunneled securely (subject to further verification).

Populating the Social Workscape

Most interactive systems are designed for single users, individuals sitting at their own computers. In contrast, very many of the activities we carry out at computers are related to other people. We write documents to send to others, we read email that other people sent us, we develop software systems for others to use, and we create Web pages for others to read. Our working lives are suffused with other people, but the computer systems through which these lives are lived are stubbornly individual. The research domain of Computer-Supported Cooperative Work has developed a range of systems that provide explicit support for collaborative and collective activities, but this approach separates the domain of "collaborative" applications from that of "individual" ones. In our richly interconnected everyday worlds, even single-user applications are used to carry out tasks that relate to multiple individuals. I may be a single user of this word processor, but I write with a sense of the others who have commented on my document and who might read it in future. As in the examples encountered earlier, system design has erected barriers – this time, between a person sitting at a computer and the rich social workscape that extends beyond it. One of our current projects is attempting to remedy this problem.

The solution we are developing is based around social networks. In social science, social networks refer to patterns of connection between individuals that bind them into larger collectivities. Social networks can be constructed around many different relations. For instance, a network can be defined around the people who talk to each other on any given day. Person A talks to persons B, C, and D; person D talks to persons E and F, and so on. By iteratively exploring this relationship, a "network" can be drawn up which maps out the patterns of contact between individuals. Other networks might be drawn around people who frequent the same cafes, people who know each other, people who work together on projects, and so on. The network makes it possible to measure and compare various characteristics of the social group, such as subgroups, degrees of connectedness and centrality. Social network analysis is a common technique in the social sciences, an approach that analysts use to understand social groups. In our work, though, we are exploring the role that social networks can play for end users of computing systems. Can we automatically determine features of the social networks in which people are enmeshed, and can we use those features to "populate" the electronic workspace, turning it into a window onto the social workscape beyond? For example, can my filesystem be organized and displayed in ways that show me what other people are associated with the activities represented by my files?

There are three parts to this research. First, we are developing an infrastructure for determining and mapping social network information, which will then support the design of awareness and other socially-aware facilities to be integrated into the everyday user experience. We use a variety of sources of information, including patterns of electronic mail contact, other sources of electronic communication, filesystem activity, etc., to create a rich information base for exploring potential social networks. Second, we are building analysis tools that allow us to explore this information to find interesting relationships. We need to find the particular measures of social network activity that are especially relevant to people – perhaps different measures for different times or for different groups. Third, we are designing ways to embed this information into the conventional user experience. Again, as in the previous examples, we wish to avoid the problems of fragmentation and alienation by making this information pervasively available as a part of the user experience, rather than providing a new "social network browser" application.

One interesting observation that has arisen from this work is that social network information may be most useful when combined with other forms of information. In particular, we are interested in the use of temporal information in order to gain more insight into the social network statistics that we can gather. Social networks provide information about, for instance, who might talk with whom; by augmenting this with temporal information, we can also begin to understand when this might happen. This allows us to develop more nuanced presentations of information. For example, if I am working on a paper with a colleague, this information might allow the system to make the icon for that document increasingly prominent as we get closer to the time of day when my colleague normally goes home for the evening, making it clearer to me that I should work on this soon. The issue of temporal patterns of activity is increasingly receiving attention in the research community (e.g. Hudson et al., 2002; Reddy and Dourish, 2002; Begole et al., 2002); our Social Workscape activities provide us with opportunities to explore the technical and design implications of this research.

Ad-Hoc and Emergent Information Structures

The final example here deals with a set of general problems surrounding practices of information management. Managing information is clearly at the heart of what computers do, and what we do with computer systems. Much of our everyday activity involves information management, either in the large (processing database records) or in the small (navigating a filesystem or filing email). Most information tasks are handled by introducing structures into which the information will fit; information is managed by navigating and manipulating the structure. Examples of structures include hierarchies (as in the filesystem), schemas (as in databases), or graphs (for web pages). Structures are convenient mechanisms for systems, but research on users managing information has repeatedly shown that they can present significant obstacles to user interaction (see, e.g., Shipman and Marshall, 1999).

There are at least four common problems. First, information structures must be predefined; the structures are defined in advance of the information. So, a folder must already exist before an email message can be filed in it; a database schema must be defined before the first record can be stored. On the other hand, when people create structures, they do so out of the detail of the information they have on hand. When I decide how my email should be organized, it depends on the messages that I have to arrange. I might use different structures for different sets of information. Defining structures ahead of use puts the cart before the horse. Second, information structures are hard to revise. It’s easy to add a new folder for my email, but it’s very hard to revise my whole filing structure, to organize it by topic rather than by person. Most information structures have this property. Even when the structure is easy to change, it may not be easy to move my information from one structure to another. Third, information structures are uniform, while actual circumstances frequently involve exceptions and variations. Information structures work by making everything the same, while actual circumstances are often characterized by variety and difference. Fourth, information and structure typically exist on different levels. Most systems separate the tools by which we manage information from the tools by which we examine, define, and modify structure. Working on both information and on structure can often mean moving repeatedly back and forth between “definition” and “use.” Different information systems exhibit different problems, or exhibit them to different degrees, but these four problems are relatively common across a range of information systems.

Again, the problem is one of barriers. Information structures act as a barrier between users and their information. In the early days of computing, this was a worthwhile trade-off; computers were large, slow, and expensive, and so the extra effort required of users to manage their information according to fixed and static structures was repaid in the improved performance that computers could achieve with uniform information in well-formed structures. However, the inexorable advance of Moore’s Law has changed the balance. Computers are now much faster, and their time is no longer worth more than human time. It is now possible for computer systems to work with unstructured or semi-structured information, or to derive structure automatically, fitting it to the data and the circumstances. However, this suggests a change to how we manage and work with information.

We are exploring tools for ad hoc and emergent structure. By ad hoc structure, we mean systems in which information is related by multiple different information structures (or even no structure at all), and in which structure is developed on the fly, according to immediate needs and activities. By emergent structure, we mean those structures that are developed by engaging with and exploring the information. For example, in sensemaking tasks, in which users must sift through large amounts of information in order to develop an understanding of the whole, structure is emergent – it arises out of the interaction between people and information.

In the Placeless Documents project (Dourish et al., 2000), we developed information systems based on an attribute-system in which fixed structures such as hierarchies and schemas were replaced with flexible and open-ended (or even nonexistent) structures based on attributes that were meaningful to end-users. Users could annotate information objects with attributes, which the system would then use to create dynamic structures through a high-performance query engine. To complement this information infrastructure, we have more recently been working on interactive systems that attempt to provide equally flexible ways to interact with information.

The particular approach we have been exploring is *spatial hypertext*. Where conventional hypertext systems relate information through explicit links, spatial hypertext systems allow users to arrange information freely in a two-dimensional space, and then exploit spatial and other visual properties to relate information items. The system can automatically recognize potentially meaningful patterns, such as columns, tables, or piles. Structure is implicit in what users do, although they can use it to interact with object groups. There is no separation between using the information and defining structure – structure is defined by moving an object from one place to another. So, creating, destroying and manipulating structures is done in the course of regular interaction with the objects. Objects can live in many structures at once, or in none; the structure is provided to assist user tasks, but is not enforced or required.

Spatial hypertext systems have shown considerable promise for some time, but there remain a number of important open questions. Two of the questions we are attempting to address are, first, how can we scale these systems up to large amounts of information while maintaining the casual interaction style?; and second, how can we extend this style of interaction to collaborative activities? To this end, we have built a spatial hypertext system that supports multi-way collaborative interactions and is based on a zooming paradigm. The visual properties on which spatial hypertext rests are scale-independent, so a zooming model seems a natural way to extend the space; and by incorporating both synchronous and asynchronous interaction, we hope to support the emergent information practices of workgroups.

THE EXPERIENCE OF COMPUTATION

What I have been arguing for here is a research agenda for HCI that concentrates on the "experience of computation." By "experience," I mean to draw attention to the direct, embodied, participative nature of interaction; this interaction style is concerned with the ways that computation is manifest in the course of working activities, and is made directly and continuously accessible to users. By "computation," I mean to draw attention to the way that, rather than adopting external metaphors or high-level abstractions, we attempt to focus on computation as a medium, representational and responsive. Each of the four ongoing projects introduced above addresses one or more aspects of this long-term goal, and is designed to provide some insight into the general problems to be solved. Much more work needs to be done, besides. We are accompanying these design efforts with ethnographic investigations of work, as well as foundational explorations of the relationship between computation and experience (Dourish, 2001), drawing on the work of others who have grappled with similar questions (e.g. Agre, 1997; Smith, 1996).

Our central concern, though, is one that unites these different activities and gives them a common focus. At the start of the twenty-first century, our experience of computation is dominated by models and approaches first developed half-way through the twentieth. Our goal is to follow through and develop the interactional consequences of the massive transformation in computation that has taken place since that time, and which continues to develop and evolve.

Acknowledgements

The work described here is being carried out in collaboration with a number of students and colleagues, including Johan Byttner, Danyel Fisher, Haimin Lee, Sameer Patil, Madhu Reddy, and David Redmiles.

References

Agre, P. 1997. *Computation and the Human Experience*. Cambridge: Cambridge University Press.

Begole, J., Tang, J., Smith, R., and Yankelovich, N. 2002. Exploring Work Rhythm Awareness: Coordinating Contact Among Colleagues. *Proc. ACM Conf. Computer-Supported Cooperative Work CSCW 2002* (New Orleans, LA) New York: ACM.

Dourish, P. 2001. *Where the Action Is: The Foundations of Embodied Interaction.* Cambridge: MIT Press.

Dourish, P. and Byttner, J. 2002. A Visual Virtual Machine for Java Programs: Exploration and Early Experiences. *Workshop on Visual Computing* (San Francisco, CA). New York: Springer.

Dourish, P. and Redmiles, R. 2002. An Approach to Usable Security based on Event Monitoring and Visualization. *Proc. New Paradigms in Security Workshop* (Virginia Beach, VA).

Dourish, P. In press. The Appropriation of Interactive Systems: Some Lessons from Placeless Documents. *Computer-Supported Cooperative Work.*

Dourish, P., Edwards, K. Lamarca, A., Lamping, J., Petersen, K., Salisbury, M., Terry, D., and Thornton, J. 2000. Extending Document Management Systems with User-Specific Active Properties. *ACM Transactions on Information Systems*, 18(2), 140-170.

Hudson, J., Christensen, J., Kellogg, W., and Erickson, T. 2002. "I'd be Overwhelmed, but It's Just One More Thing To Do:" Availability and Interruption in Research Management. *Proc. ACM Conf. Human Factors in Computing Systems CHI 2002* (Minneapolis, MN). New York: ACM.

Mackay, W. 1990. *User and Customizable Software: A Coadaptive Phenomenon*. PhD Dissertation, Sloan School of Management, MIT.

Muramatsu, J. and Pratt, W. 2001. Transparent Queries: Investigating Users' Mental Models of Search Engines. *Proc. ACM Conf. Research and Development in Information Retrieval* (New Orleans, LA.) New York: ACM.

Preece, J., Rogers, Y., and Sharp, H. 2002. *Interaction Design: Beyond Human-Computer Interaction*. Wiley.

Reddy, M. and Dourish, P. 2002. A Finger on the Pulse: Temporal Rhythms and Information Seeking in Medical Work. *Proc. ACM Conf. Computer-Supported Cooperative Work CSCW 2002* (New Orleans, LA). New York: ACM.

Robertson, G., Card, S., and Mackinlay, J. 1993. Information visualization using 3d interactive animation. *Communications of the ACM*, 36:57 – 71.

Scaife, M. and Rogers, Y. 1996. External Cognition: How do Graphical Representations Work? *Intl. Jnl. Human-Computer Studies*, 45, 185-213.

Shipman, F. and Marshall, C. 1999. Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the use of Formal Representations in Interactive Systems. *Computer Supported Cooperative Work*, 8(4), 333-352.

Smith, B.C. 1996. *On the Origin of Objects*. Cambridge, MA: MIT Press.

Wenger, E. 1998. *Communities of Practice: Learning, Meaning and Identity*. Cambridge: Cambridge University Press.