The Infrastructure Problem in HCI

W. Keith Edwards[1], Mark W. Newman[2], and Erika Shehan Poole[1]

[1]GVU Center & School of Interactive Computing
College of Computing
Georgia Institute of Technology
85 5th Street NW
Atlanta, GA 30308 USA
{keith, erika}@cc.gatech.edu

[2] School of Information and
Dept of Electrical Engineering/Computer Science
University of Michigan
1075 Beal Avenue
Ann Arbor, MI 48109 USA
mwnewman@umich.edu

ABSTRACT
HCI endeavors to create human-centered computer systems, but underlying technological infrastructures often stymie these efforts. We outline three specific classes of user experience difficulties caused by underlying technical infrastructures, which we term *constrained possibilities, unmediated interaction,* and *interjected abstractions*. We explore how prior approaches in HCI have addressed these issues, and discuss new approaches that will be required for future progress. We argue that the HCI community must become more deeply involved with the creation of technical infrastructures. Doing so, however, requires a substantial expansion to the methodological toolbox of HCI.

Author Keywords
Infrastructure, human-centered design, toolkits

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Design, Human Factors

INTRODUCTION

A central goal of HCI is to understand how to design *human-centered* computer systems. But even the best-intentioned user experience designers fight an uphill battle against layers of underlying technological infrastructure that may not be designed with the full range of human-centered concerns in mind. Most user interfaces are not built in isolation, but sit atop a collection of software libraries, toolkits, protocols, and standards typically inaccessible to user-centered design processes. Traditionally, HCI has had little to say about these layers, instead focusing on creating compelling user experiences *within the constraints posed by that underlying infrastructure*. Infrastructure presents a fundamental tension for HCI. For user-centered design to provide realistic, useful, deployable, and economical solutions, designers rely heavily on existing technological infrastructures. At the same time, this dependency may at times restrict their ability to fully address user needs and capabilities. We argue that such lower level concerns, while not traditionally considered within the scope of HCI, significantly affect user experience, determining what sorts of functionality can be delivered, the logic by which functions are organized, and the interdependencies among these functions. In this paper, we ask what it would mean for HCI to consider the infrastructures upon which applications are built and explore possible ways that HCI might have more impact on discussions surrounding the design of infrastructure.

This paper makes two key contributions to our understanding of the tension between HCI and infrastructure. First, we identify three ways infrastructure can negatively impact user experience, and illustrate them with case studies. Second, we examine a number of possible approaches to address the infrastructure problem in HCI. By organizing these approaches into a framework organized by the level at which they engage the infrastructure, we identify new possibilities for how HCI can engage infrastructure development more effectively.

WHAT IS "INFRASTRUCTURE?"

"Infrastructure" is a broad term that can be applied to any system, organizational structure, or physical facility that supports an organization or society in general [54]; this broad term has been used to describe interconnect systems that sink into the background of everyday life (e.g., roads, sewers, or telecommunications networks), conceptual abstractions (e.g., disease classification schemes), and more complex relationships between politics, individuals, organizations, and technical systems [31, 35, 55, 57]. In this paper, however, we take a constrained view of infrastructure, focusing solely on the domain of software systems. By this definition, infrastructure comprises system-level software providing functions, capabilities, or

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.

services to *other* software. Operating systems, libraries, toolkits, frameworks, services, protocols, and interoperation standards are common examples of the infrastructure we take as our focus in this paper. We focus on these software infrastructures particularly because as software applications become increasingly complex, distributed, and interconnected, reliance on infrastructure increases. Despite the impact of infrastructure on the user experience, there is little understanding within the HCI community as to how we may contribute to the development of software infrastructures leading to positive user experiences.

Infrastructure Design Tensions

Software infrastructures are a necessity; they provide economic and technical benefits of *reuse, separation of concerns,* and *interoperability*. If many applications require a particular software capability, it makes economic sense to build a reusable infrastructural component. For example, the development of GUI toolkits relieved application developers of the burden of creating common widgets such as lists, panels, and resizable windows [40]. Infrastructure also supports a *separation of concerns,* enabling specialized capabilities to be implemented by developers with the most appropriate skills. For example, security-related functions typically are provided via pre-existing libraries (e.g., the secure string functions in C), as it is unlikely that most application developers would correctly and efficiently implement these functions on their own. A third benefit of infrastructure is *interoperability*. Whenever an application communicates with software created by another entity, some kind of infrastructure must exist in order to facilitate the connection between those parties. For example, for a web browser to be able to retrieve and display a web page from a server, the browser and server must conform to predefined standards specifying, at a minimum, an application protocol (HTTP), a transport protocol (TCP), and a data format (HTML or XML), among others.

Although the economic and technical benefits of infrastructure are considerable, they are removed from the immediate concerns of users. In and of themselves, for example, such economic and technical benefits do not guarantee other desirable properties that we may wish from an infrastructure, such as maintainability, conceptual clarity, simplicity, or support for troubleshooting. While technical and economic considerations certainly can coexist with such human-motivated considerations, our own experience, as well as evidence from a growing body of literature [5, 9, 22, 37], suggests that an exclusive focus on traditional criteria does not reliably lead to positive user experience outcomes. In the next section, we discuss previous work examining the relationship between HCI and infrastructure, and present examples of user experience breakdowns occurring as a result of conventional approaches to infrastructure design.

Previous Reflections on HCI and Infrastructure

A number of studies have examined the uncomfortable relationship between infrastructure and human experience. These studies have looked at those disillusioned or disenfranchised by civic infrastructure [35], home networking challenges [9, 22], and cognitive difficulties with security software [60], among others. In addition, a number of authors have made recommendations about how particular kinds of infrastructure technologies might be built to better account for human concerns. These recommendations have included suggestions for better infrastructure design processes [17], better abstractions for exposing infrastructure capabilities [14], appropriate considerations for evaluating user interface systems research [44], and architectural patterns for supporting usability [27]. We extend this prior work by identifying *general* problems with infrastructure, providing a framework for understanding the strengths and weaknesses of previous approaches to addressing HCI's infrastructure problem, and suggesting new approaches that have not been previously addressed in the literature.

FACETS OF THE INFRASTRUCTURE PROBLEM

In this section we consider three ways in which technical infrastructures shape user experience. In particular, we discuss:

- **Constrained possibilities:** Design choices taken by the infrastructure may preclude entirely certain desirable user experience outcomes.
- **Interjected abstractions:** Technical abstractions in the interface may appear in the conceptual model exposed to users.
- **Unmediated interaction:** Users may have to interact directly with the infrastructure to accomplish their goals.

We present a set of case studies to illustrate each of these manifestations of the infrastructure problem.

Case Study #1: UPnP and Constrained Possibilities

We define *constrained possibilities* as situations in which the technical capabilities of an infrastructure preclude the ability to create certain desirable user experiences. For example, consider Universal Plug and Play (UPnP), a technology supporting interoperability among networked consumer electronics devices, particularly media-oriented devices [38]. UPnP simplifies device setup by removing tedious, manual configuration of networked devices; in theory, when users plug new devices into their networks, these devices should work immediately. However, UPnP is limited to supporting a set of pre-defined device types and is therefore brittle in the face of evolving user needs and technical capabilities.

To illustrate, consider the case of a user who purchases a new UPnP device, but also has a number of older UPnP devices in her home. Upon installation, she discovers that her existing devices are *not* interoperable with this new

device's advanced functionality. Perhaps the new device supports a UPnP profile not yet defined at the time the earlier devices shipped. Perhaps the older devices simply support an earlier version of the profile that the new device uses. She has three options: return the new device, replace the old equipment already in the home, or hope that vendors of all of the existing equipment make a patch that allows them to work with the new device (which, of course, necessitates the headache of upgrading the software on all the existing devices.) This experience represents the kind of broken expectation noted by Bly et al. that is common when infrastructures fail to provide capabilities anticipated by a user [5].

What has gone wrong with UPnP? We argue that UPnP's failure to deliver the promised experience is, in fact, inherent in the infrastructure itself. UPnP, like other interoperation standards such as USB and Bluetooth, takes an "ontological" approach to interoperability in which *service profiles* describe each type of device. UPnP, for example, defines profiles for a set of device types, including media display devices, media storage devices, HVAC components, and printers. These service profiles are standardized by committees and specify details of how to communicate with a given device, including which operations each class of device can perform.

In the approach taken by UPnP, standardization at this level of detail is necessary for interoperability (a client application that encounters a MediaRenderer "knows" that it will be able to communicate with it using the standard set of operations). Further, such standardization affords a sort of polymorphism (MediaRenderers from any vendor can be simply "plugged in" to the network and used by existing clients without writing any additional software). Yet this approach to interoperability comes at a cost. Standardization at this level means that new *types* of devices *are unusable by existing applications*. New sorts of devices (say, for example, a MediaUploader associated with an online photo sharing service), with new functionality and semantics, require a different profile. Existing software—including the software on devices already installed on the network—will not be written against this new profile and thus will be unable to use the new device type [19]. Even new *versions* of existing device types face this problem, as new revisions of the standards define new features unusable by devices or applications created before the revision. Moreover, the evolution of an ontological standard is a slow process. For UPnP, new device types (or revisions to existing device types) must be standardized by the UPnP Forum, then implemented and shipped by device manufacturers, and then finally acquired and deployed by consumers.

Fundamentally, then, the design choices in UPnP privilege interoperability among a handful of *existing* (and slowly evolving) categories of devices, but at the cost of accommodating entirely new types of devices. This design choice presents an inherent user experience conundrum: because of UPnP's approach to interoperability, whenever users wish to introduce a new type of device into an existing network, they are faced with incompatibilities or the need for wholesale upgrades of existing devices. It is often difficult to anticipate these incompatibilities in advance, as doing so would require specialized knowledge of the infrastructure. Thus, UPnP users are left with difficult choices: do they upgrade everything in their homes to accommodate new devices? Do they abandon hopes of having a new device for the time being? Once there are software patches available, do they bother with the potentially tedious task of upgrading all of their existing equipment to support the new device?

Is the implicit prioritization made by UPnP the "correct" one from a user experience perspective? Are there other approaches that might have mitigated these downsides? A different set of infrastructure design decisions in UPnP (and in other similarly designed infrastructures such as USB and Bluetooth) may have led to a different set of trade-offs, enabling more "ad hoc" interoperability while preserving existing user experience benefits. For example, self-describing formats such as tuplespaces [26], or mobile code-based approaches [19] have the potential to enable easier interoperability among devices with less *a priori* knowledge of each other—meaning that the user experience enabled by these infrastructures could be one of more fluid, dynamic use of new devices, without the hassles of continuous software updates or device replacement.

Case Study #2: ACLs and Interjected Abstractions

The second category we explore is *interjected abstractions*, or situations in which low-level infrastructural concepts become part of the conceptual model of the interface. All technical infrastructures present abstractions to application developers. These abstractions may be in the form of software objects (e.g., modules, classes, and functions), or underlying concepts (e.g., reliable vs. unreliable transport protocols). Developers use these abstractions to create applications with user-facing features. In many cases, these infrastructural abstractions—perhaps designed to further technological priorities such as extensibility, modularity, or performance—become exposed to users through the applications built on top of the infrastructure.

For example, consider access control lists (ACLs), used by operating systems and middleware to regulate access to information and system resources. ACLs have a long history in computing, going back to the early 1970s [30]. They have a number of advantages: ACLs are easily specified in machine-friendly ways (lists of principals, along with the privileges accorded to those principals); they are also a highly efficient means of regulating access (checking whether a given process has the ability to read a file, for example, can be done by simply checking whether the process's owner has the "read" permission in the file's access control list) [51]. Thus, from a technical perspective, this abstraction is well designed, efficient, and simple.

However, users see this abstraction more or less "as is." The typical user interface to an ACL-based system is simply a textual list of users and their corresponding access rights, despite the fact that the HCI community has long identified and discussed many of the problematic aspects of ACLs. These include ACLs' relative inflexibility [1], their requirement for *a priori* configuration [7], and more general difficulties mapping access to the negotiated, contingent way that revelation and identity are managed in social settings [48]. How can we address these problems? One avenue is to tackle the redesign of the "classic" ACL interface [36]; this approach has been shown to yield substantial usability improvements, and we agree that such improvements are necessary. However, while it may be possible to create more usable interfaces on top of ACLs, such higher level interfaces are still constrained in that they must, at the end of the day, be expressible in the forms required by ACLs: lists of human principals with their access rights for specific objects.

Understanding how such features become visible in the user experience might have led to a different set of design goals, had they been known. For example, studies have shown that users' sharing preferences and practices are conditioned by the sharing context, contents of the shared data items, and sometimes-ephemeral aspects of the relationships with collaborators [46, 58]. Alternative interfaces to sharing such as those proposed by [58] would provide a solution, but would be difficult to make widely available due to the limitations of underlying abstractions such as the ACL.

The problem of interjected abstractions extends to other cases where infrastructure abstractions become part of the application's conceptual model. For example, Whitten and Tygar's analysis of the shortcomings of PGP 5.0 point out a number of places where the underlying security infrastructure's abstractions—such as public and private keys, and webs of trust—are unlikely to be understood or properly employed by end users, thus subverting the goal of enabling secure communication among users [60].

Case Study #3: Networks and Unmediated Interaction

A third way in which infrastructure impacts user experience is one we term *unmediated interaction*, in which users must interact with the infrastructure *directly*, without the mediation of some intervening application. While we typically think of the infrastructure as being a sort of hidden "plumbing" removed from the user experience by layers of application code, the infrastructure itself becomes a locus of interaction in some cases. As noted by Star, for example, this can happen when the infrastructure "breaks" in such a way that it is no longer hidden from the user [54].

However, there are a number of computing infrastructures that are poorly mediated at best, *even in those cases where the infrastructure may not necessarily be "broken,"* and which require that users interact directly with the infrastructure. Our case study for this section is networking in the home. The presence of complex home networks is growing, with some figures showing that 34% of US households have a home network, and 25% of households having a wireless network [25]. Notably, the protocols and architectures used in the home today are essentially the *same* protocols and architectures used in the global Internet, and which were developed during the 1960's and 1970's. Many of the assumptions of the original Internet architecture no longer hold true, and this creates problems in a number of contexts, not least of which is the home [4]. The concepts of the Internet Protocols (such as IP addressing, non-routable addresses, the Domain Name Service), and network topology (e.g., routers, bridges, switches, hubs, network address translation (NAT) facilities) are all present in the home, despite the fact that these concepts and underlying architectures were originally created for a world of trained network administrators, shared responsibility for the network as a whole, and assumptions of mutual trust. This mismatch between original goals and present uses is manifest through the many reported user experience issues with networking in the home [5, 6, 9, 18, 52].

Home network users today face unmediated exposure to the network infrastructure at many points. For example, effectively (and securely) installing a home network means that users must have some modicum of understanding of the basics of network topology, including the role that the home router plays in the network; for example, that it creates the notion of an "inside" of the home network that is separate and distinct from the "outside" public Internet.

Unmediated interaction also manifests itself at times when new devices are added to the home network. The Internet infrastructure requires that devices be configured with local state information in order to operate. At a minimum, they must be configured with a range of link-layer settings (e.g., SSID, WEP key) as well as network-layer settings (e.g., IP address, router, and DNS server). Further, certain *application requirements* may force users to interact directly with the network in order to configure it to support applications' needs. For example, running a service (such as a web server to host a blog) inside the home network cannot be done on most home networks without fairly extensive network-layer configuration such as setting up NAT forwarding, configuring firewall rules, or setting up a Dynamic DNS service.

Perhaps the most common form of unmediated interaction with the network comes at troubleshooting time, when the nature of the network means that no single node may have a complete picture of where a problem exists. The opacity of the network infrastructure means that the "interface" to troubleshooting these problems is typically only the physical interface provided by the network hardware itself—blinking LEDs that indicate connectivity status, for instance, and the connectivity of physical cables.

Our point in enumerating these cases is not to reiterate the many (and well-known) problems of networking in the

home. Rather, our point is to show how certain features of the network infrastructure in homes today *necessitate* that users be exposed to it. Furthermore, we wish to highlight that it is difficult, and in some cases impossible, to create applications that shield users entirely from this exposure. The degree of unmediated interaction required of users is inherent in the design decisions made by this infrastructure.

To illustrate how other possibilities could exist for home networking, we note that other network infrastructures have taken radically different approaches, yielding radically different user experiences. The public switched telephony network (PTSN), for example, limits such unmediated interaction by embodying a different set of design priorities: in the PSTN, most of the infrastructure of the network is removed from the home entirely, end-user devices need not be configured in order to work (a landline phone, once plugged in, "knows" its phone number), and there is a minimum of troubleshooting required by users.

Summarizing the Problem

UPnP, ACLs, and home networks are three examples of infrastructure technologies that present user experience challenges. At the same time, we do not aim to discount the fact that all three enable enormously powerful capabilities for end-users. The technical skill and insight required to design each of these infrastructures is impressive and we do not intend to diminish their significance; rather, our aim is to highlight that that infrastructure design processes based largely around technical considerations in isolation are unlikely to avoid user experience problems such as the ones highlighted above. We do not propose to *replace* traditional considerations based around technical and economic issues with discussions of user experience, rather we seek to *add* to the perspectives already represented in the design of infrastructure technology.

THE WAY FORWARD

So far we have argued that technical infrastructures deeply affect the user experience of systems created on top of them. In this section, we consider various approaches to addressing the infrastructure problem in HCI. We find it productive to think of the various approaches in terms of the depth at which they engage infrastructure; in many ways, the layers of engagement that we present here echo Rodden and Benford's analysis based on Brand [49].

- *Surface* approaches focus on applying superficial layers of user-facing software in an attempt to shield users from unwieldy aspects of infrastructure.
- *Interface* approaches focus on the interface between the infrastructure and the applications it supports, endeavoring to reduce the problems caused by mismatches between conceptual models and system functionality.
- *Intermediate* approaches supply new infrastructure technologies that are more amenable to delivering a positive user experience, though they are often constrained by other, more fundamental infrastructure layers.
- *Deep* approaches seek to directly influence the architecture of infrastructure itself. In contrast to intermediate approaches, deep approaches *require* the engagement of multiple technical disciplines, most notably the systems specialists who have traditionally dominated discussions of infrastructure design. These are the most challenging approaches, and the least understood by the HCI community at present. They represent the next frontier in the struggle to overcome HCI's infrastructure problem.

Surface Approaches

Surface-layer approaches have been the path of much work in HCI. Accepting infrastructure *as given* and attempting to present a prettier picture to users via application software is often the most expedient development path, in particular when infrastructure is immutable due to technical or practical reasons. Masking underlying infrastructure can sometimes address *unmediated interaction* by providing abstractions that improve the match between user expectations and system functionality. It may also address *interjected abstractions*, though remapping abstractions often has unintended negative consequences when these superimposed abstractions break down. This approach, however, fails to address *constrained possibilities*, as it simply accepts the infrastructure *as is*.

Interface Approaches

Some researchers in HCI have proposed exposure of infrastructure in novel ways that allow greater transparency to users as well as flexibility to user experience designers. These approaches involve modifying the infrastructures to expose more accurate, or more appropriable, abstractions to developers or users, potentially in ways unanticipated by the infrastructure designers.

Seamful design

Bell et al. [2] argue for a "seamful design" approach, in which designers expose human-salient aspects of infrastructure rather than masking them. Fundamentally, the seamful approach is about accepting the notion that users will be exposed to infrastructure, but ensuring that that exposure is done in such a way that users can perceive and appropriate underlying abstractions, and share them with other users. In contrast to surface-layer approaches, infrastructure limitations become a resource for user reasoning and improvisation. This approach prefers the potential shortcomings of *unmediated interaction* to the pitfalls of *interjected abstractions*. It does not, however, directly tackle the issue of *constrained possibilities*, as seamful design proposes a new way to present existing infrastructure rather than new approaches to designing infrastructure itself.

Reflective architectures

A second interface-layer approach allows software developers to access the internal state of the infrastructure in new ways, including potentially for uses unforeseen by the original infrastructure developers. Such *reflective architectures* expose mechanisms that allow developers to not only use the infrastructure, but also reason about its current state and behavior, and potentially even modify its behavior in new ways. As an example, Dourish suggests an approach to reflective system design, called *accounts*, which allow applications to introspect infrastructure-layer abstractions to determine how they are working; such an approach, while not "fixing" the problems of poor infrastructure, allows the creation of applications that can access infrastructure in ways not available using traditional programming interfaces [15]. In some cases, reflective approaches go further—allowing application developers to modify underlying infrastructure implementation and behaviors. Such capabilities have been explored in domains ranging from programming languages (Kiczales' meta-object protocols [28] and "open implementation" approaches [34], most notably, but also the more prosaic mechanisms provided by the reflection APIs in the Java language) to collaborative toolkits [15] to real-time operating systems [53], to networks [3].

All of these systems share the ability for developers to reach into the infrastructure and to query and potentially even change its internal workings in ways that traditional approaches do not allow—even in ways that may have been unforeseen by the infrastructure's original developers. This ability can be used by applications to both adapt to the constraints of the underlying infrastructure, as well as to adapt underlying infrastructure to user-centered needs. Such approaches seem particularly suited toward overcoming challenges of *interjected abstractions* and *unmediated interaction*. While these approaches may offer potential for overcoming certain *constrained possibilities*, this potential is largely determined by how deeply into the infrastructure the reflective features reach.

Support for intelligibility

Intelligibility, or explaining system configuration and state to users in an understandable way, is another approach to overcoming mismatches between users' mental models and the system's model of operation. For example, Dey et al. describe infrastructural abstractions [13] and end-user inspection facilities [33] that support end-users' comprehension of the behavior of context-aware systems. Support for intelligibility has the ability to overcome the problem of *interjected abstractions* by providing users with a more understandable view of the how the system works. It can also address *unmediated interaction* by essentially providing a new layer of mediation. Taken as an independent approaches (i.e., separate from the construction of *new* infrastructure, which is discussed next), intelligibility does not address the issue of *constrained possibilities*.

Intermediate Approaches

A number of HCI researchers have developed frameworks, toolkits, and libraries aimed at supporting certain specific user experiences. For example, platforms for context-aware computing [23, 50], tangible computing [29], flexible document management [16], and distributed user interfaces [42, 45] strive to overcome the limitations of existing infrastructure and ease the construction of novel applications. We refer to these as *intermediate approaches* because they typically sit atop of a layer of more fundamental infrastructure (e.g. existing operating systems, networking protocols, security mechanisms). In addition, these approaches tend to be focused enough that they can be developed unilaterally by HCI researchers seeking to support a particular style of application or a specific interaction facility, without the need to engage with the systems, networking, or security communities. This approach embodies much of the work of the "technical HCI" community, and has engaged discussions of the relationship between HCI and infrastructure.

New infrastructure technologies

Frequently, these infrastructure components follow a "top down" approach, in which applications are created in an ad hoc manner before the need for a general infrastructure is identified. For example, if several constructed applications share common needs, then often this shared functionality may be abstracted out into an infrastructure to support better reuse [24, 29]. Alternatively, as necessary features become apparent at the application layer, this provides a set of requirements for the next layer in the software stack, and so on. As an example, consider an application to allow secure exchange of content among users on an ad hoc network (along the lines of the Casca system reported in [20]). At certain points, the application may require that a dialog box appear that specifies who is requesting what content. For this dialog to be created, the underlying system must be capable of providing the user-facing application with information such as association of a device with its human owner, human-readable names to describe the content being requested, and so forth. These requirements, in turn, may argue for certain security protocols that can support these technical features. Thus, after identifying user-facing needs, general capabilities can be pushed down into infrastructure in a top-down manner. Assuming each application was built and evaluated to ensure a good fit with human needs and capabilities (not always a safe assumption), it is likely that the infrastructure will be similarly suitable. However, this approach breaks down when the existing infrastructure cannot support certain applications. In such cases, changes to the infrastructure must precede application development, and alternative measures must be employed to ensure that the infrastructure does not fall prey to the pitfalls discussed earlier.

New infrastructure processes

In situations where infrastructure needs to be built before it is possible to build the applications it enables, some have

argued for involving the practice of human-centered design and evaluation in the creation of infrastructure itself [17]. Incorporating such methods, it is hoped, will ensure that the underlying concepts of the infrastructure can more directly support the desired user experiences, whether of applications built upon the infrastructure, or through unmediated exposure to the infrastructure itself. This approach, however, is difficult given the current state of the art in HCI. Our traditional methods prove most useful when we have a well-known task, an application that supports that task, and an understanding of the user who will use that application. Our methods are less suited to situations where aspects of the system which we seek to design, or evaluate, are far removed from the application itself, or when there may be no specific task for which to design.

Currently, to the degree that human concerns are systematically brought to bear on technical infrastructure, it is done in an ad hoc fashion. For example, multiple lightweight prototypes might be built on an infrastructure; these prototypes serve as "proxies" for how real applications built on the final infrastructure might work, and can provide feedback about certain infrastructure features, albeit indirectly (e.g. this is the approach in [17]).

This approach could be developed more fully, for example by more directly aligning underlying abstractions of the infrastructure with the conceptual model we wish to expose to users. To the degree that these models can be aligned, there would be no "gap" between the veneer of user interface abstractions and the concrete reality of the system's underlying behavior. This certainly may not be possible in many cases; in others, it may not be desirable (perhaps for technical reasons, such as performance or security). However, such an arrangement would likely enhance the ability of users to work with the infrastructure in an unmediated way, and mitigate the abrupt step change between application-layer concepts and infrastructure-layer concepts, leading to infrastructure that is more actionable and predictable. Such an approach could allow users to more readily form actionable conceptual models about how the behavior of their infrastructure, reducing some of the well-known "gulfs" in user experience [43].

Intermediate approaches have the potential to avoid all three of the pitfalls we have been discussing with respect to the user experience of infrastructure, albeit usually limited to their restricted domain of focus. Even when an application is built directly atop a well-designed infrastructure component, it will typically rely on other, inherited infrastructure components that are out of either the application developers' or component developer's control. For example, a context-aware application built on top of an infrastructure like the Context Toolkit [12] will *also* be required to interact with common networking protocols, windowing toolkits, file systems, etc. As such, it will be susceptible to the user experience limitations of the infrastructure on which it depends.

Deep Approaches

The deepest and most pervasive aspects of software infrastructure are the layers that, to this point, HCI has had the least ability to influence, because they are the purview of the systems, networking, and security communities. However, as we have argued throughout this paper, ignoring these layers imposes significant constraints on our ability as a community to deliver compelling user experiences. While there may not be a difference in kind between the infrastructure technologies created through what we have termed "intermediate" and "deep" approaches, there is an important difference in the degree to which each approach engages stakeholders outside the HCI community—most importantly those in the computing disciplines who have traditionally been the most concerned with issues of software infrastructure.

Thus the final set of approaches we argue for in this paper focus on influencing those who themselves create the technical infrastructures we rely upon, to help them to create substrate technologies that are more usable by and useful to users. While one way to effect such change may be through education (teaching software practitioners about human-centered practices, for example), here we focus on what we consider to be a key challenge for our discipline in how we communicate our results to those in other disciplines, and how we influence their work.

Learn to speak the language of infrastructure

Much technically-oriented research focuses on a handful of quantitative measures for evaluating the "goodness" of a system—in networking, for example, throughput, latency, and scalability are canonical metrics (and entire infrastructures have emerged just to support evaluation along these metrics, such as PlanetLab [10]); for security systems, metrics such as cryptographic security (the computational cost of the fastest known attack on an algorithm) may be used to assess the merit of a given system. In and of themselves, these measures provide useful input to system designers, allowing them to answer questions, for example, about whether a given architecture can support the bandwidth required for video applications, or whether a given security system is safe enough for military-grade work. However, when used by themselves, these metrics paint a skewed picture—they provide a set of technically-oriented metrics that can be optimized and traded off against one another, but *without counterbalance from the human side of the equation.* In the absence of some way to represent user-facing concerns in the technical design process, pure technical capability is the primary (even sole) driver of technical infrastructure research and development.

Thus, the current state of the practice of technical infrastructure development too often focuses on isolated measures that present a reductionist view of the concerns that should face creators of such infrastructure. Far from being a problem that the technical disciplines should be expected to address in isolation, we argue that this is a

challenge for the HCI community: to foster the creation of new means for communication across disciplines. One such means of engagement would be for HCI to more effectively master the language of metrics that is spoken by the systems community.

Currently, the only major human-oriented metric that has been widely adopted outside of the HCI community is performance-oriented: namely, the human performance guideline that indicates that 100 millisecond response time is perceived as "instantaneous" [8, 39] is used as a technical goal in many systems ([47, 56], among many examples). Although the 100 millisecond number may itself be debated [11], there is little doubt that this metric has assumed the mantle of conventional wisdom, becoming widespread as a performance upper bound for systems designers from a range of disciplines.

However, there are far more aspects of the user experience that may be salient: concerns such as installability, evolvability, predictability, and intelligibility may all have great impact on the user experience—taken from a holistic perspective, perhaps even greater than mere response time. Distilling down such aspects of the user experience so that they can be accounted for in the design of technical infrastructure, however, is not an easy task. Producing metrics for these aspects—intended to serve as a counterbalance to existing, technically-oriented metrics—may be misguided at best, and counter-productive at worst: many of these aspects of the user experience are nuanced and multi-faceted, reflecting the complexity of human behavior and experience. Simplistic translation into some quantitative measure may do as much harm as good, obscuring nuances and subtleties that lead to yet more inappropriate infrastructure design choices.

While voices within HCI have argued for a greater emphasis on quantitative metrics for comparison of alternatives since the inception of the field [8, 41], we argue that there are distinct challenges with providing such metrics to inform the design of infrastructure. First, obtaining comparative data or establishing benchmarks for many aspects of user experience would be difficult: for infrastructural capabilities that enable new interactions, there may be no existing systems that form an appropriate basis for comparison. Further, benchmarks for qualities such as "match with users' conceptual model," "ease of maintenance," and "fits well with existing systems and practices" would be impractical to establish, and in any case would mean very little when isolated from each other. Second and most importantly, many of the most important aspects of the user experience of infrastructure unfold over long periods of time and are subject to future conditions that cannot be reliably forecast prior to the implementation and deployment of the system. Unlike isolated system metrics such as throughput, scalability, and key strength that can be computed analytically or modeled in accepted ways, user experience criteria will likely remain fuzzy and, to some extent, speculative for the foreseeable future.

Change the conversation

In the absence of easily computable quantitative metrics, how can we effectively engage with the technical disciplines on which HCI depends? Other examples of cross-disciplinary interactions may offer suggestions. In this section we look to the field of Environmental Planning as a provocative example. While we do not argue that the case here is directly transferable to computing, it does illustrate new approaches for integrating diverse concerns in design.

Environmental Planning grew out of Urban Planning and the 1960's environmental movement as a way to broaden the discussion about new urban development projects to include factors of "environmental impact" that were traditionally left out of mainstream urban planning discussions. To be specific, environmental planning seeks to introduce concerns such as land use, air pollution, noise pollution, effects on wildlife habitats, socio-economic impacts, and visual impacts of particular projects into discussions that were dominated by traditional, "functional" urban planning concerns such as economic development, transportation, and sanitation [59]. In the United States, the environmental planning agenda has been greatly facilitated by the 1969 passage of the National Environmental Policy Act (NEPA) which mandates the assessment of environmental impacts for certain types of development projects [21]—similar laws have been subsequently passed in other countries as well. The effect of NEPA is that for the affected development projects, an Environmental Impact Assessment (EIA) must be performed, and its results made a matter of public record prior to the granting of permission to proceed with a new project. For the purposes of our discussion, the most remarkable aspect of the EIA process is its breadth of assessment criteria.

The EIA mechanism accommodates a wide range of criteria that are measured in radically different ways. Indeed, the completion of an EIA often requires calling upon the skills of various disparate specializations, including ecologists, economists, sociologists, and experts in cultural heritage. The Leopold matrix [32] is commonly used in EIAs to represent the various activities involved in a project and their effects on the criteria of interest. The matrix is a simple table that plots attributes of a proposed project against the estimates of the impacts that each attribute will have on the environmental factors of interest. Each cell of the table contains two numbers: an assessment of the *magnitude* of the impact and an assessment of its *importance*. What is significant about this representation is its ability to serve as a *boundary object* among multiple disciplines involved in Environmental Planning. Indeed, the NEPA specifically requires that federal agencies "use 'a systematic and interdisciplinary approach' to ensure that social, natural, and environmental sciences are used in planning and decision-making," and in particular to develop methods so that "presently unquantified… values may be given appropriate consideration in decision-making along

with traditional economic and technical considerations" [21].

One may posit an "Infrastructure Impact Assessment" for software as a thought experiment. Like the EIA, it could accommodate the perspectives, methods, and assessment criteria of multiple disciplines. User experience criteria could live alongside traditional the criteria of system performance, cost of implementation, security, and so forth. HCI specialists would employ a range of techniques in order to generate assessments, including scenario generation, prototype building, user testing, ethnographic studies, user modeling, as well as new techniques that would likely evolve to address questions specific to infrastructure evaluation. Note also that such a mechanism might also support the inclusion of other important perspectives often missing from infrastructure discussions in addition to usability, such as environmental sustainability, access for disabled users, and impacts on economic competition.

We do not argue that notions like an Infrastructure Impact Assessment are necessarily the right solution to the infrastructure problem in HCI, nor even that they are entirely workable. Nevertheless, mechanisms like the EIA offer examples of how others have approached the challenge of involving multiple perspectives, from radically different disciplines, into the process of designing complex systems intended to be long-lived and robust. At a minimum, mechanisms such as the EIA surface and make visible and explicit concerns that, today, are often invisible or implicit in the creation of software infrastructures.

CONCLUSION

We have argued that HCI should expand its methodological toolbox to address the design of system infrastructures typically considered outside of the realm of "the user experience." Technical infrastructures not only influence user experience but in many cases may preclude entirely certain desirable possibilities. Certainly, there will always be constraints to what can be built; we do not argue that such constraints will somehow disappear when HCI becomes involved in the lower layers of the software stack. However, our lack of involvement in the lower layers of software has led to systems that—while "working" from a technical perspective—present less-than-ideal abstractions to users, require tedious and error-prone troubleshooting, and limit their applicability to new situations.

We have argued for a number of possible ways to mitigate these effects, but the challenges ahead remain substantial. Fundamentally, we see the need to involve HCI practitioners more deeply with the creation of technical infrastructure, and doing so requires new methodological approaches to allow us to both understand the user experience implications of infrastructure design decisions, as well as to guide the development of infrastructure features given a desired user experience outcome. Adopting processes from other fields that have faced analogous challenges integrating diverse stakeholders may be informative. To have impact beyond the application layer, engagement with the technical concerns of infrastructure is a necessity for the HCI community.

REFERENCES

1. Ackerman, M.S.: The Intellectual Challenge of CSCW: The Gap Between Social Requirements and Technical Feasibility. Human-Computer Interaction, 15(2-3):179-203, (2000)
2. Bell, M., et al.: Interweaving Mobile Games with Everyday Life. ACM CHI 2006 417-426
3. Bhattacharjee, S., Calvert, K.L. and Zegura, E.W.: An Architecture for Active Networking. Proceedings of the Seventh IFIP Conference on High Performance Networking, White Plains, NY, USA (1997) 265-279
4. Blumenthal, M.S. and Clark, D.D.: Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World. ACM Trans. on Internet Tech., 1(1):70-109, (2001)
5. Bly, S., et al.: Broken Expectations in the Digital Home. ACM CHI 2006 568-573
6. Calvert, K.L., Edwards, W.K. and Grinter, R.E.: Moving Toward the Middle: The Case Against the End-to-End Argument in Home Networking. ACM HotNets-VI (2007)
7. Cao, X. and Iverson, L.: Intentional Access Management: Making Access Control Usable for End-Users. ACM SOUPS 2006 20-31
8. Card, S.K., Moran, T.P. and Newell, A.: The Psychology of Human-Computer Interaction. Erlbaum (1983)
9. Chetty, M., Sung, J.-Y. and Grinter, R.E.: How Smart Homes Learn: The Evolution of the Networked Home and Household. Ubicomp 2007 127-144
10. Chun, B., et al.: PlanetLab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev., 33(3):3-12, (2003)
11. Dabrowski, J.R. and Munson, E.V.: Is 100 Milliseconds Too Fast? : CHI 2001 317-318
12. Dey, A.K., Abowd, G.D. and Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Hum.-Comput. Interact., 16(2):97-166, (2001)
13. Dey, A.K. and Newberger, A.: Support for Context-Aware Intelligibility and Control. ACM CHI 2009 859-868
14. Dourish, P.: Developing a Reflective Model of Collaborative Systems. ACM ToCHI, 2(1):40-63, (1995)
15. Dourish, P.: Accounting for System Behaviour. Computers and Design in Context (1997) 145–170
16. Dourish, P., et al.: Extending Document Management Systems with User-Specific Active Properties. ACM Trans. Inf. Syst., 18(2):140-170, (2000)
17. Edwards, W.K., et al.: Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Infrastructure. ACM CHI 2003:297-304, (2003)
18. Edwards, W.K. and Grinter, R.E.: At Home With Ubiquitous Computing: Seven Challenges. Ubicomp 2001 256-272
19. Edwards, W.K., et al.: Experiences with Recombinant Computing. ACM ToCHI, 16(1):1-44, (2009)

20. Edwards, W.K., et al.: Using Speakeasy for Ad Hoc Peer to Peer Collaboration. ACM CSCW 2002 256-265
21. Glasson, J.: Introduction to Environmental Impact Assessment. Routledge (2005)
22. Grinter, R.E., et al.: The Work to Make a Home Network Work. ECSCW 2005 469-488
23. Hong, J.I. and Landay, J.A.: An Architecture for Privacy-Sensitive Ubiquitous Computing. ACM MobiSys 2004 177-189
24. Hong, J.I. and Landay, J.A.: SATIN: a Toolkit for Informal Ink-Based Applications. ACM UIST 2000 63-72
25. Horrigan, J.: Home Broadband Adoption 2009. Pew Internet & American Life, Washington, DC, USA (2009)
26. Johanson, B. and Fox, A.: Extending Tuplespaces for Coordination in Interactive Workspaces. J. Syst. Softw., 69(3):243-266, (2004)
27. John, B.E., et al.: Bringing Usability Concerns to the Design of Software Architecture. Engineering HCI and Interactive Systems. Springer (2005) 1-19
28. Kiczales, G., des Rivieres, J. and Bobrow, D.: The Art of the Meta-Object Protocol. MIT Press, Cambridge (1991)
29. Klemmer, S.R., et al.: Papier-Mache: Toolkit Support for Tangible Input. ACM CHI 2004 399-406
30. Lampson, B.: Protection. ACM SIGOPS Operating Systems Review, 8(1):18-24, (1974)
31. Lee, C., Dourish, P. and Mark, G.: The Human Infrastructure of Cyberinfrastructure. ACM CSCW 2006. 483-492
32. Leopold, L.B., et al.: A Procedure for Evaluating Environmental Impact. U.S. Geological Survey (1971)
33. Lim, B.Y., Dey, A.K. and Avrahami, D.: Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems. ACM CHI 2009 2119-2128
34. Maeda, C., et al.: Open Implementation Analysis and Design. SIGSOFT Softw. Eng. Notes, 22(3):44-52, (1997)
35. Mainwaring, S.D., Chang, M.F. and Anderson, K.: Infrastructures and Their Discontents: Implications for Ubicomp. Ubicomp 2004 418-432
36. Maxion, R.A. and Reeder, R.W.: Improving User-Interface Dependability Through Mitigation of Human Error. Int. J. Hum.-Comput. Stud., 63(1-2):25-50, (2005)
37. McDonald, D.W., Smith, K.A. and Karlova, N.: Problem Solving Probes: A Method for Discovering Conceptual Disconnects with Digital Living Technologies. ACM CSCW 2008 Designing for Families Workshop
38. Miller, B., et al.: Home Networking with Universal Plug and Play. IEEE Communications, 39(12):104-109, (2001)
39. Miller, R.B.: Response Time in Man-Computer Conversational Transactions. December 1968 Fall Joint Computer Conference (1968)
40. Myers, B., Hudson, S.E. and Pausch, R.: Past, Present, and Future of User Interface Software Tools. ACM ToCHI, 7(1):3-28, (2000)
41. Newman, W.M.: Better or just different? On the benefits of designing interactive systems in terms of critical parameters. ACM DIS 1997 239-245
42. Nichols, J., et al.: Generating Remote Control Interfaces for Complex Appliances. ACM UIST 2002 161-170
43. Norman, D.A.: The Design of Everyday Things. Basic Books (2002)
44. Olsen, D.R.: Evaluating User Interface Systems Research. ACM UIST 2007 251-258
45. Olsen, D.R., et al.: Cross-Modal Interaction Using XWeb. ACM UIST 2000 191-200
46. Olson, J.S., Grudin, J. and Horvitz, E.: A Study of Preferences for Sharing and Privacy. ACM CHI 2005 1985-1988
47. Oppenheimer, P.: Top-down network design. Cisco (1998)
48. Palen, L. and Dourish, P.: Unpacking "Privacy" for a Networked World. ACM CHI 2003 129-136
49. Rodden, T. and Benford, S.: The evolution of buildings and implications for the design of ubiquitous domestic environments. ACM CHI 2003
50. Salber, D., Dey, A.K. and Abowd, G.D.: The Context Toolkit: Aiding the Development of Context-Enabled Applications. ACM CHI 1999 434-441
51. Sandhu, R. and Samarati, P.: Access control: Principle and Practice. IEEE Communications, 32(9):40-48, (1994)
52. Shehan, E. and Edwards, W.K.: Home Networking and HCI: What Hath God Wrought? : CHI 2007 547-556
53. Stankovic, J. and Ramamritham, K.: A Reflective Architecture for Real-Time Operating Systems. Advances in Real-Time Systems:23-38, (1995)
54. Star, S.L.: The Ethnography of Infrastructure. The American Behavior Scientist, 43(3):377-391, (1999)
55. Star, S.L. and Ruhleder, K.: Steps towards an ecology of infrastructure: complex problems in design and access for large-scale collaborative systems. ACM CSCW 1994 253-264
56. Tolia, N., Andersen, D.G. and Satyanarayanan, M.: Quantifying Interactive User Experience on Thin Clients. Computer, 39(3):46-52, (2006)
57. Tolmie, P., et al.: Unremarkable computing. ACM CHI 2002. pp. 399-406
58. Voida, S., et al.: Share and Share Alike: Exploring User Interface Affordances of File Sharing. ACM CHI 2006 221-230
59. Westman, W.E.: Ecology, Impact Assessment, and Environmental Planning. Wiley (1985)
60. Whitten, A. and Tygar, J.D.: Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. USENIX Security Symposium (1999)