Abstract
This article discusses the challenges in computer systems research posed by the emerging field of pervasive computing. It first examines the relationship of this new field to its predecessors: distributed systems and mobile computing. It then identifies four new research thrusts: effective use of smart spaces, invisibility, localized scalability, and masking uneven conditioning. Next, it sketches a couple of hypothetical pervasive computing scenarios, and uses them to identify key capabilities missing from today's systems. The article closes with a discussion of the research necessary to develop these capabilities.

Pervasive Computing: Vision and Challenges
M. Satyanarayanan, Carnegie Mellon University

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

The goal of this article is to help us understand the challenges in computer systems research posed by pervasive computing. We begin by examining its relationship to the closely related fields of distributed systems and mobile computing. Next, we sketch two pervasive computing scenarios, and ask why they are fiction rather than fact today. From that starting point, we delve deeper into some key research problems. To preserve focus on computer systems issues, we avoid digressions into other areas important to pervasive computing such as human-computer interaction, expert systems, and software agents.

Related Fields
Pervasive computing represents a major evolutionary step in a line of work dating back to the mid-1970s. Two distinct earlier steps in this evolution are distributed systems and mobile computing. Some of the technical problems in pervasive computing correspond to problems already identified and studied earlier in the evolution. In some of those cases, existing solutions apply directly; in other cases, the demands of pervasive computing are sufficiently different that new solutions have to be sought. There are also new problems introduced by pervasive computing that have no obvious mapping to problems studied earlier. In the rest of this section we try to sort out this complex intellectual relationship and to develop a taxonomy of issues characterizing each phase of the evolution.

Distributed Systems
The field of distributed systems arose at the intersection of personal computers and local area networks. The research that followed from the mid-1970s through the early 1990s created a conceptual framework and algorithmic base that has proven to be of enduring value in all work involving two or more computers connected by a network — whether mobile or static, wired or wireless, sparse or pervasive. This body of knowledge spans many areas that are foundational to pervasive computing and is now well codified in textbooks [2-4]:

- Remote communication, including protocol layering, remote procedure call [5], the use of timeouts, and the use of end-to-end arguments in placement of functionality [6]
- Fault tolerance, including atomic transactions, distributed and nested transactions, and two-phase commit [7]
- High availability, including optimistic and pessimistic replica control [8], mirrored execution [9], and optimistic recovery [10]
- Remote information access, including caching, function shipping, distributed file systems, and distributed databases [11]
- Security, including encryption-based mutual authentication and privacy [12]

Mobile Computing
The appearance of full-function laptop computers and wireless LANs in the early 1990s led researchers to confront the problems that arise in building a distributed system with mobile clients. The field of mobile computing was thus born. Although many basic principles of distributed system design continued to apply, four key constraints of mobility forced the development of specialized techniques: unpredictable variation in network quality, lowered trust and robustness of mobile elements, limitations on local resources imposed by weight and size constraints, and concern for battery power consumption.

Mobile computing is still a very active and evolving field of research, whose body of knowledge awaits codification in textbooks. The results achieved so far can be grouped into the following broad areas:

- Mobile networking, including Mobile IP, ad hoc protocols, and techniques for improving TCP performance in wireless networks
- Mobile information access, including disconnected operation, bandwidth-adaptive file access, and selective control of data consistency
- Support for adaptive applications, including transcoding by proxies and adaptive resource management
- System-level energy saving techniques, such as energy-aware adaptation, variable-speed processor scheduling, and energy-sensitive memory management
- Location sensitivity, including location sensing and location-aware system behavior
Earlier in this article, we characterized a pervasive computing environment as one saturated with computing and communication capability, yet so gracefully integrated with users that it becomes a "technology that disappears." Since motion is an integral part of everyday life, such a technology must support mobility; otherwise, a user will be acutely aware of the technology by its absence when he moves. Hence, the research agenda of pervasive computing subsumes that of mobile computing, but goes much further. Specifically, pervasive computing incorporates four additional research thrusts into its agenda, as illustrated by Fig. 1.

**Effective Use of Smart Spaces** — The first research thrust is the effective use of smart spaces. A space may be an enclosed area such as a meeting room or corridor, or a well-defined open area such as a courtyard or quadrangle. By embedding computing infrastructure in building infrastructure, a smart space brings together two worlds that have been disjoint until now. The fusion of these worlds enables sensing and control of one world by the other. A simple example of this is the automatic adjustment of heating, cooling, and lighting levels in a room based on an occupant’s electronic profile. Influence in the other direction is also possible: software on a user’s computer may behave differently depending on where the user is currently located. Smartness may also extend to individual objects, whether located in a smart space or not.

**Invisibility** — The second thrust is invisibility. The ideal expressed by Weiser is complete disappearance of pervasive computing technology from a user’s consciousness. In practice, a reasonable approximation to this ideal is minimal user distraction. If a pervasive computing environment continuously meets user expectations and rarely presents him with surprises, it allows him to interact almost at a subconscious level. At the same time, a modicum of anticipation may be essential to avoiding a large unpleasant surprise later, much as pain alerts a person to a potentially serious future problem in a normally unnoticed body part.

**Localized Scalability** — The third research thrust is localized scalability. As smart spaces grow in sophistication, the intensity of interactions between a user’s personal computing space and his/her surroundings increases. This has severe band-
width, energy, and distraction implications for a wireless mobile user. The presence of multiple users will further complicate this problem. Scalability, in the broadest sense, is thus a critical problem in pervasive computing. Previous work on scalability has typically ignored physical distance — a Web server or file server should handle as many clients as possible, regardless of whether they are located next door or across the country. The situation is very different in pervasive computing. Here, the density of interactions has fallen off as one moves away; otherwise, both the user and his computing system will be overwhelmed by distant interactions that are of little relevance. Although a mobile user far from home will still generate some distant interactions with sites relevant to him, the preponderance of his/her interactions will be local.

Like the inverse square laws of nature, good system design has to achieve scalability by severely reducing interactions between distant entities. This directly contradicts the current ethos of the Internet, which many believe heralds the “death of distance.”

**Masking Uneven Conditioning** — The fourth thrust is the development of techniques for masking uneven conditioning of environments. The rate of penetration of pervasive computing technology into the infrastructure will vary considerably depending on many non-technical factors such as organizational structure, economics, and business models. Uniform penetration, if it is ever achieved, is many years or decades away. In the interim, there will persist huge differences in the “smartness” of different environments — what is available in a well-equipped conference room, office, or classroom may be more sophisticated than in other locations. This large dynamic range of “smartness” can be jarring to a user, detracting from the goal of making pervasive computing technology invisible.

One way to reduce the amount of variation seen by a user is to have his/her personal computing space compensate for “dumb” environments. As a trivial example, a system that is capable of disconnected operation is able to mask the absence of wireless coverage in its environment. Complete invisibility may be impossible, but reduced variability is well within our reach.

---

**Drilling Down**

Practical realization of pervasive computing will require us to solve many difficult design and implementation problems. Building on the discussion in earlier sections, we now look at some of these problems at the next level of detail. Our goal is only to convey an impressionistic picture of the road ahead. We make no claim of completeness or exclusiveness; this specific set of topics is merely a sampling of the problem space, presented in no particular order.

In this discussion we assume each user is immersed in a personal computing space that accompanies him/her everywhere and mediates all interactions with the pervasive computing elements in his/her surroundings. This personal computing space is likely to be implemented on a body-worn or handheld computer (or a collection of these acting as a single entity). We refer to this entity as a “client” of its pervasive computing environment, even though many of its interactions may be peer-to-peer rather than strictly client-server. As indicated by the discussion below, the client needs to be quite sophisticated and hence complex.

Figure 2 illustrates the structure of an Aura client, giving a concrete example of this complexity, showing the components of an Aura client and their logical relationships. The text in italics indicates the role played by each component. Coda and Odyssey were created prior to Aura, but are being modified substantially to meet the demands of pervasive computing.

In the case of Odyssey, these changes are sufficiently extensive that they will result in Chroma, a replacement. Other components, such as Prism and Spectra, are being created specifically for use in Aura. Additional components are likely to be added over time since Aura is relatively early in its design at the time of this writing. Server and infrastructure support for Aura are not shown here.

---

**Figure 2.** The structure of an Aura client.
gy, with the surrogate serving as the mobile computer’s networking gateway to the Internet. When an intensive computation accessing a large volume of data has to be performed, the mobile computer ships the computation to the surrogate; the latter may cache data from the Internet on its local disk in performing the computation. Alternatively, the surrogate may have staged data ahead of time in anticipation of the user’s arrival in the neighborhood. In that case, the surrogate may perform computations on behalf of the mobile computer or merely service its cache misses with low latency by avoiding Internet delays. When the mobile computer leaves the neighborhood, its surrogate bindings are broken, and any data staged or cached on its behalf are discarded.

Cyber foraging opens up many important research questions. Here are some examples:

- How does one discover the presence of surrogates? Of the many proposed service discovery mechanisms such as JINI, UPnP, and BlueTooth proximity detection, which is best suited for this purpose? Can one build a discovery mechanism that subsumes all of them for greatest flexibility?
- How does one establish an appropriate level of trust in a surrogate? What are useful levels of trust in practice? How applicable and useful is the concept of caching trust [34]? Can one amortize the cost of establishing trust across many surrogates in a neighborhood?
- How is load balancing on surrogates done? Is surrogate allocation based on an admission control approach or a best-effort approach? How relevant is previous work on load balancing on networks of workstations?
- In typical situations, how much advance notice does a surrogate need to act as an effective staging server with minimal delay? Is this on the order of seconds, minutes, or tens of minutes? What implications does this requirement have for the other components of a pervasive computing system?
- What are the implications for scalability? How dense does the fixed infrastructure have to be to avoid overloads during periods of peak demand?
- What is the system support needed to make surrogate use seamless and minimally intrusive for a user? Which are the components of this support that must be provided by the mobile client, and which by the infrastructure?

Adaptation Strategy

Adaptation is necessary when there is a significant mismatch between the supply and demand of a resource. The resource in question may be wireless network bandwidth, energy, computing cycles, memory, and so on. There are three alternative strategies for adaptation in pervasive computing.

First, a client can guide applications in changing their behavior so that they use less of a scarce resource. This change usually reduces the user-perceived quality, or fidelity, of an application. Odyssey [23, 24] is an example of a system that uses this strategy.

Second, a client can ask the environment to guarantee a certain level of a resource. This is the approach typically used by reservation-based quality of service (QoS) systems [35]. From the viewpoint of the client, this effectively increases the supply of a scarce resource to meet the client’s demand.

Third, a client can suggest a corrective action to the user. If the user acts on this suggestion, it is likely (but not certain) that resource supply will become adequate to meet demand. An example of this approach was described earlier in the article: in Scenario 1, Aura advised Jane to walk to Gate 15 in order to obtain adequate wireless bandwidth. While conceptually promising, no real system has implemented this approach yet.

All three strategies are important in pervasive computing. The existence of smart spaces suggests that some of the environments encountered by a user may be capable of accepting resource reservations. At the same time, uneven conditioning of environments suggests that a mobile client cannot rely solely on a reservation-based strategy — when the environment is uncooperative or resource-impoverished, the client may have no choice but to ask applications to reduce their fidelities. Corrective actions broaden the range of possibilities for adaptation by involving the user, and may be particularly useful when lowered fidelity is unacceptable.

Many questions remain to be answered:

- How does a client choose between adaptation strategies? What factors should a good decision procedure take into account? How should different factors be weighted? What role, if any, should the user play in making this decision? How can smooth and seamless transitions between strategies be ensured as a user moves?
- At first glance, it appears that the second strategy (reservation-based QoS) is always superior from the viewpoint of the user, since he/she is required to neither accept lower fidelity nor perform a corrective action. Is this true in all circumstances? What are the hidden costs and “gotchas,” if any, in a widely deployed system?
- How will the implementation of a smart space honor resource reservations? What are the most appropriate admission control policies when there are competing requests from multiple clients? What resources beside wireless network bandwidth is it meaningful and useful for a smart space to reserve? What are the application programming interfaces (APIs) and protocols necessary to negotiate these reservations?
- Is adaptation using corrective actions practically feasible? Do users find such a strategy intrusive or annoying? What is the best way to communicate potential corrective actions to users? What are the programming models and APIs necessary to support corrective actions? Can existing applications use this approach? If so, how substantial are the modifications to them?
- What are the different ways in which fidelity can be lowered for a broad range of applications? Are existing APIs, such as that of Odyssey [23], adequate? How should those APIs and programming models be revised in the light of extensive usage experience? In particular, what is the negative impact of lowered fidelity on users, and how can this be minimized?

High-Level Energy Management

Sophisticated capabilities such as proactivity and self-tuning increase the energy demand of software on a mobile computer in one’s personal computing space. At the same time, relentless pressure to make such computers lighter and more compact places severe restrictions on battery capacity. There is growing consensus that advances in battery technology and low-power circuit design cannot, by themselves, reconcile these opposing constraints — the higher levels of the system must also be involved [36, 37].

How does one involve the higher levels of a system in energy management? One example is energy-aware memory management [26], where the operating system dynamically controls the amount of physical memory that has to be refreshed. Another example is energy-aware adaptation [24], where individual applications switch to modes of operation with lower fidelity and energy demand under operating system control. Many research questions follow:

- In what other ways can the higher levels of a system contribute to managing energy? What are the relative strengths and weaknesses of these approaches? When should one method be used in preference to another?
- How does high-level energy management impact the goal of invisibility in pervasive computing? How intrusive or distracting do users find such techniques?
Can knowledge of user intent be exploited in energy management? If so, how robust is this approach in the face of imperfection in this knowledge?

Can smart spaces and surrogates be used to reduce energy demand on a mobile computer? What is the range of possible approaches, and what are their relative merits?

What is the role of remote execution in extending battery life? Under what circumstances does its energy savings exceed the energy cost of wireless communication? Can a system predict these savings and costs accurately enough in practice to make a significant difference?

Client Thickness

How powerful does a mobile client need to be for a pervasive computing environment? In other words, how much CPU power, memory, disk capacity, and so on should it have? The answer will determine many of the key constraints imposed on the hardware design of the client. In trade press jargon, a thick client is a powerful client, while a thin client is a minimal one.

Thick clients tend to be larger, heavier, require a bigger battery, and dissipate more heat—all negative factors from the viewpoint of the user who has to carry or wear the client. Over time, improvements in very large-scale integration (VLSI) and packaging technology can reduce the physical size and weight of a thick client. However, those improvements will translate to an even smaller and lighter thin client. For a mobile user, a client can never be too small or too light, or have too much battery life!

A wide range of feasible designs has been demonstrated. At one extreme are ultra-thin clients such as Infopad [38, 39] and SLIM [40]. These bare-bones devices are little more than high-resolution displays connected through high-bandwidth wireless links to nearby compute servers. At the other extreme are full-function clients capable of standalone or disconnected operation. Examples include the Navigator family of wearable computers [41] and laptops running as clients of the Coda File System [18]. Such designs can make use of wireless connectivity when available, but are not critically dependent on it. Handheld computers such as the PalmPilot represent design points between these extremes. They can operate in isolation, but run a limited range of applications.

For a given application, the minimum acceptable thickness of a client is determined by the worst-case environmental conditions under which the application must run satisfactorily. A very thin client suffices if one can always count on high-bandwidth low-latency wireless communication to nearby computing infrastructure, and batteries can be recharged or replaced easily. If there exists even a single location visited by a user where these assumptions do not hold, the client will have to be thick enough to compensate at that location. This is especially true for interactive applications where crisp response is important.

With a client of modest thickness, it may be possible to preserve responsiveness by handling simple cases locally and relying on remote infrastructure only for more compute-intensive situations. Alternatively, it may be possible to execute part of the application locally and then ship a much-reduced intermediate state over a weak wireless link to a remote compute server for completion. The hybrid mode of speech recognition in Odyssey [23] is an example of this approach. A nother approach would be for the client to recognize that a key assumption is not being met, and to alert the user with an intelligible message. The client could also suggest possible corrective actions such as moving to a nearby location that is known to be suitable for the application.

Uneven conditioning of environments implies that an extreme-thin-client approach will be unsatisfactory for pervasive computing in the foreseeable future. At the same time, there is considerable merit in not having to carry or wear a client thicker than absolutely necessary. Any research questions follow from this tension:

Can the concepts of client thickness and environmental conditioning be quantified? Are there “sweet spots” in the design space where a modest increase in client thickness yields considerable improvement in performance and usability?

Can a proactive system alert a user in a timely manner before he leaves a benign environment for a less hospitable one? In that context, can an application be transparently migrated from a thinner to a thicker client and vice versa? What are the kinds of applications for which such migration is feasible and useful? What is the impact on usability?

Is it possible to build cost-effective modular computers that can be physically reconfigured to serve as the optimal mobile clients under diverse environmental conditions? Can a proactive system advise a user to reconfigure when appropriate? Knowing higher travel plans, can such a system guide in configuring the system so that it is of adequate thickness at all times?

Can semi-portable infrastructure be carried with a user to augment less hospitable environments? For example, in a poorly conditioned environment, can a thin bodyworn computer extend its capabilities by wireless access to a full-function laptop brought by the user? This is analogous to carrying both a briefcase and a wallet when you travel; the briefcase is not physically on your person at all times, but it is close enough to provide easy access to things too large to fit in your wallet. Is this a usable and practical strategy to cope with uneven conditioning?

Context Awareness

A pervasive computing system that strives to be minimally intrusive has to be context-aware. In other words, it must be cognizant of its user’s state and surroundings, and must modify its behavior based on this information. A user’s context can be quite rich, consisting of attributes such as physical location, physiological state (e.g., body temperature and heart rate), emotional state (e.g., angry, distraught, or calm), personal history, daily behavioral patterns, and so on. If a human assistant were given such context, he or she would make decisions in a proactive fashion, anticipating user needs. In making these decisions, the assistant would typically not disturb the user at inopportune moments except in an emergency. Can a pervasive computing system emulate such a human assistant?

A key challenge is obtaining the information needed to function in a context-aware manner. In some cases, the desired information may already be part of a user’s personal computing space. For example, that space may include schedules, personal calendars, address books, contact lists, and to-do lists. More dynamic information has to be sensed in real-time from the user’s environment. Examples of such information include position, orientation, the identities of people nearby, locally observable objects and actions, and emotional and physiological state.

Implementing a context-aware system requires many issues to be addressed. For example:

How is context represented internally? How is this information combined with system and application state? Where is context stored? Does it reside locally, in the network, or both? What are the relevant data structures and algorithms?

How frequently does context information have to be consulted? What is the overhead of taking context into account? What techniques can one use to keep this overhead low?

What are the minimal services an environment needs to provide to make context awareness feasible? What are reasonable fallback positions if an environment does not provide such services? Is historical context useful?

What are the relative merits of different location-sensing technologies? Under what circumstances should one be used in preference to another? Should location information
be treated just like any other context information, or should it be handled differently?

Balancing Proactivity and Transparency
Proactivity is a double-edged sword. Unless carefully designed, a proactive system can annoy a user and thus defeat the goal of invisibility. How does one design a system that strikes the proper balance at all times? Self-tuning can be an important tool in this effort. A mobile user’s need and tolerance for proactivity are likely to be closely related to his/her level of expertise on a task and familiarity with his/her environment. A system that can infer these factors by observing user behavior and context is better positioned to strike the right balance.

Historically, the ideal in system design has been transparency. For example, caching is attractive in distributed file systems because it is completely transparent. Unfortunately, servicing a cache miss on a large file over a low-bandwidth wireless network takes so long that most users would rather be asked first whether they really need the file. However, a flurry of such interactions can overwhelm the user. Coda suggests a way to resolve this dilemma [19]. On a cache miss, the system consults an internally maintained user patience model to predict whether the user will respond positively to a fetch request. If this appears likely, the user interaction is suppressed and the fetch is handled transparently.

Many subtle problems arise in designing a system that walks the fine line between annoying proactivity and inscrutable transparency. For example:
• How are individual user preferences and tolerances specified and taken into account? Are these static, or do they change dynamically?
• What cues can such a system use to determine if it is veering too far from balance? Is explicit interaction with the user to obtain this information acceptable? Or would it be an annoyance too?
• Can one provide systematic design guidelines to application designers to help in this task? Can one retrofit balancing mechanisms into existing applications?

Privacy and Trust
Privacy, already a thorny problem in distributed systems and mobile computing, is greatly complicated by pervasive computing. Mechanisms such as location tracking, smart spaces, and use of surrogates monitor user actions on an almost continuous basis. A user becomes more dependent on a pervasive computing system; it becomes more knowledgeable about that user’s movements, behavior patterns and habits. Exploiting this information is critical to successful proactivity and self-tuning. At the same time, unless use of this information is strictly controlled, it can be put to a variety of unsavory uses ranging from targeted spam to blackmail. Indeed, the potential for serious loss of privacy may deter many from using a pervasive computing system.

Greater reliance on infrastructure means a user must trust that infrastructure to a considerable extent. Conversely, the infrastructure needs to be confident of the user’s identity and authorization level before responding to his/her requests. It is a difficult challenge to establish this mutual trust in a manner that is minimally intrusive and thus preserves invisibility.

Privacy and trust are likely to be enduring problems in pervasive computing. Any research questions follow. For example:
• How does one strike the right balance between seamless system behavior and the need to alert users to potential loss of privacy? What are the mechanisms, techniques, and design principles relevant to this problem? How often should the system remind a user that his/her actions are being recorded? When and how can a user turn off monitoring in a smart space?
• What are the authentication techniques best suited to pervasive computing? Are password-based challenge-response protocols such as Kerberos [42] adequate, or are more exotic techniques such as biometric authentication [43] necessary? What role, if any, can smart cards [44] play?
• How does one express generic identities in access control? For example, how does one express security constraints such as “Only the person currently using the projector in this room can set its lighting level”? Or “Only employees of our partner companies can negotiate QoS properties in this smart space”?

Impact on Layering
A recurring theme in the earlier sections of this article has been the merging of information from diverse layers of a system to produce an effective response. For example, scenario 1 showed the value of combining low-level resource information (network bandwidth) with high-level context information (airport gate information). Proactivity and adaptation based on corrective actions seem to imply exposure of much more information across layers than is typical in systems today.

Layering cleanly separates abstraction from implementation and is thus consistent with sound software engineering. Layering is also conducive to standardization since it encourages the creation of modular software components. Deciding how to decompose a complex system into layers or modules is nontrivial, and remains very much an art rather than a science. The two most widely used guidelines for layering are Parnas’ principle of information hiding [45] and Saltzer et al.’s end-to-end principle [6]. However, these date back to the early 1970s and early 1980s, respectively, long before pervasive computing was conceived. Many research questions follow:
• How can the benefits of layering be preserved while accommodating the needs of pervasive computing? What is the impact of these accommodations on efficiency and maintainability?
• Are existing layers best extended for pervasive computing by broadening their primary interfaces or creating secondary interfaces (e.g., the SNMP network management interface [46])?
• When creating a new layer, are there systematic guidelines we can offer to ensure compatibility with the needs of pervasive computing? How much harder is it to design and implement such a layer?

Conclusion
Pervasive computing will be a fertile source of challenging research problems in computer systems for many years to come. Solving these problems will require us to broaden our discourse on some topics, and revisit long-standing design assumptions in others. We will also have to address research challenges in areas outside computer systems. These areas include human-computer interaction (especially multimodal interactions and human-centric hardware designs), software agents (with specific relevance to high-level proactive behavior), and expert systems and artificial intelligence (particularly in the areas of decision making and planning). Capabilities from these areas will need to be integrated with the kinds of computer systems capabilities discussed in this article. Pervasive computing will thus be the crucible in which many disjoint areas of research are fused.

When describing his vision, Weiser was fully aware that attaining it would require tremendous creativity and effort by many people, sustained over many years. The early decades of the 21st century will be a period of excitement and ferment, as new hardware technologies converge with research progress on the many fundamental problems discussed in this article. Like the frontier of the American West in the early 19th century, pervasive computing offers new beginnings for the adventurous and the restless — a rich open space where the rules have yet to be written and the borders yet to be drawn.