A System Architecture for Pervasive Computing

Robert Grimm, Tom Anderson, Brian Bershad, and David Wetherall

Department of Computer Science and Engineering
University of Washington, Seattle

{rgrimm, tom, bershad, djw}@cs.washington.edu

Abstract

Pervasive computing, with its focus on users and their tasks rather than on computing devices and technology, provides an attractive vision for the future of computing. But, while hardware and networking infrastructure to realize this vision are increasingly becoming a reality, precious few applications run in this infrastructure. We believe that this lack of applications can be attributed to three characteristics that are inadequately addressed by existing systems. First, devices are heterogeneous, ranging from wearable devices to conventional computers. Second, network connectivity often is limited and intermittent. And, third, interactions typically involve several autonomous administrative domains. In this paper, we introduce a system architecture that directly addresses these challenges. Our architecture is targeted at application developers and administrators, and it supports mobile computations, persistent storage, and resource discovery within a single, comprehensive framework.

1 Introduction

Pervasive computing [9, 34] promises a computing environment that seamlessly and ubiquitously supports users in accomplishing their tasks and that renders the actual computing devices and technology largely invisible. Hardware and networking infrastructure to realize this vision are increasingly becoming a reality. Cell phones now include a processor and storage; palm-sized computing devices are widely available and increasingly powerful; and, devices with new form factors, such as pads, are about to be commercially released. Moreover, wireless networking standards, such as IEEE 802.11 or Bluetooth [3], provide local connectivity for mobile nodes, while the Internet provides world-wide connectivity.

At the same time, precious few distributed applications run in this infrastructure. Notable exceptions are electronic mail for communication and the World Wide Web as a medium for electronic publishing and as a client interface for multi-tier applications. We believe that this lack of applications can be attributed to three main reasons. First, hardware devices span a wide range of platforms, computing power, storage capacity, form factors, and user interfaces. Second, network connectivity generally is limited, intermittent, and, at best, fluctuating. Third, this infrastructure represents an amalgam of autonomous and independently managed administrative domains. Taken together, these three characteristics create an environment in which it is difficult to design, build, and deploy distributed applications.

A number of research projects have been exploring how to build distributed applications in a global computing infrastructure. Several efforts focus on an object-oriented programming model and infrastructure [11, 20, 33] as well as on extending traditional operating system abstractions to the wide area [32]. But, these systems are targeted at conventional computers running conventional operating systems and are unlikely to scale over the range of devices in a pervasive computing environment. Other efforts focus on managing replicated state [15, 17, 29] and on managing autonomous mobile computations, or agents [23]. While these technologies are clearly useful, they either address how to manage distributed state or how to manage distributed computations. They thus lack an integrated framework for building applications and require additional services, which, presumably, are provided by some operating system.

We argue that existing operating system abstractions and services are neither sufficient nor necessarily appropriate for a pervasive computing infrastructure. Rather, to overcome the three challenges posed by this infrastructure, it is necessary to define a comprehensive new computing platform, or system architecture, that runs across all devices. This architecture needs to be simple, so that it can be implemented across the range of devices. It needs to, at a minimum, integrate support for mobile computations and persistent storage, so that applications can work reliably under limited connectivity. And, it needs to be easy to encapsulate, so that applications can be effectively secured and managed while also preserving local autonomy.

To this end, we present a first cut at a system architecture for pervasive computing that directly addresses the concerns of heterogeneous devices, limited connectivity, and autonomous administration in order to make it feasible to design, develop, and deploy applications on a global scale. Our architecture is targeted at application developers and administrators, and it is based on three main abstractions: Tasks represent computations, tuples represent persistent data, and environments provide structure and control.

The rest of this paper is structured as follows. Section 2 motivates our research with an example scenario and describes the challenges posed by a pervasive computing environment in more detail. Section 3 develops a strategy for overcoming these challenges and describes the core functionality of a system architecture for pervasive computing. Section 4 presents the design of our solution. Finally, Section 5 concludes this paper with a discussion of future work.

2 Motivation

A pervasive computing infrastructure provides an attractive environment for distributed applications other than electronic mail and the World Wide Web. To better illustrate the potential of such an infrastructure, consider the following scenario, which describes how Alyssa P. Hacker relies on this infrastructure to seamlessly and unobtrusively provide personal information management, in the widest sense of the term.

2.1 An Example Scenario

Since resigning from her position at MIT and moving from Cambridge to the Bay Area, Alyssa has been highly successful as a speaker and consultant on the social impact of computing technology. Her clients include many local high-tech companies, professional organizations, as well as governmental institutions around the world. As part of her job, Alyssa visits different organizations around the Silicon Valley almost daily and frequently travels across the world.

For managing her complex schedule and her numerous contacts as well as for communicating with clients she relies on a number of devices. At her office, Alyssa uses a conventional PC. In her opinion, keyboard, mouse, and a window-based interface still represent a very effective user interface, especially when writing reports and developing presentations. At home, her PlayStation 3 not only functions as a digital VCR and music jukebox, but also provides her with access to her contacts, schedule, and electronic mail. On the road, Alyssa prefers smaller, more portable devices. Her 4-band cell phone is the most important; in addition to global voice communications, it lets her access her personal information manager and provides her with a music player to entertain her during those long international flights. At a client's location, she typically uses the client's computing facilities to make presentations, and, when staying at a hotel, she uses the in-room information appliance to access her office workspace. But, for travel to less technologically developed parts of the world, she also relies on a pad with an optional keyboard.

Across all these devices, Alyssa can access basically the same information and functionality. Her schedule and contacts are automatically synchronized between them. And, when accessing her workspace from a client's computer or a hotel's information appliance, she is presented with *her* desktop, featuring the same applications, data, and customizations as those on her office PC. Furthermore, many of the tasks centered around personal information management are performed automatically and often without human interaction. For example, after agreeing on a meeting, the actual date and time for the meeting are automatically scheduled by her and her client's personal information managers. And, her manager reacts to a successfully scheduled out-of-town meeting by making the necessary travel arrangements, booking appropriate flights and hotels, all based on Alyssa's preferences, but without directly involving her (besides notifying her of the results).

2.2 Challenges

While this scenario represents an attractive application of pervasive computing, it also poses three major challenges in the form of heterogeneous devices, limited connectivity, and autonomous administration. Computing devices cover a wide range of form factors, ranging from traditional computers to information appliances to small, mobile devices, such as cell phones, and employ different user interfaces, ranging from window- and keyboard-based to pen-based to voice-based interaction. They also differ in computing platform and in available computing power, storage capacity, and network connectivity. At the same time, given current technology trends, we can realistically expect that *all* devices in this infrastructure command reasonable amounts of processing power, storage capacity, and networking bandwidth [9].

Connectivity is affected by several factors. First, in a global network the speed of light becomes tangible—a roundtrip between diametrically opposed points on earth takes at least 0.13 seconds. Second, many nodes, notably mobile devices using wireless technology, connect through relatively high latency, low bandwidth links and, for technical as well as economical reasons, cannot remain continuously connected. Third, as the overall networking infrastructure is composed of many smaller networks, yet used as a single, shared medium, changes in load or location typically result in fluctuations of available bandwidth. Finally, node and network failures [18, 26] are indistinguishable from lost connectivity and further limit access to resources. As a result, connectivity is limited, intermittent, and, at best, fluctuating. However, it is not necessarily unpredictable. For example, network connectivity for Alyssa's cell phone and pad depends on her current location and can thus be correlated to her schedule. More generally, overall Internet traffic patterns can be correlated to human factors, such as working hours and lunch breaks [30].

The need for autonomous administration is a direct result of the structure of a global computing infrastructure, which spans different organizations and is composed of many smaller networks. For example, in our scenario, Alyssa relies on the computing facilities in her home, at her office, at a client's location and at hotels. The different facilities typically cooperate in exchanging network traffic and in providing application services, for example, to let a personal information manager schedule meetings and make reservations. But, each network is usually owned by a separate organization and thus managed independently.

3 Strategy

Individually, heterogeneous devices, limited connectivity, and autonomous administration raise the specific issues of how to effectively implement applications across the range of devices in a pervasive computing infrastructure, how to reliably provide application functionality even when connectivity is limited and intermittent, and how to safely and securely deliver application services across administrative domains while also preserving local autonomy. Taken together, heterogeneous devices, limited connectivity, and autonomous administration make us question the very way applications are structured.

The approach used for client-server and multi-tier computing statically partitions application logic amongst nodes and relies on distributed state for coordinating the different nodes. But, limited connectivity and autonomous administration exacerbate the problems associated with managing distributed state [24] in a statically partitioned system and raise serious doubts about the scalability of this approach in a global computing infrastructure.

Several projects [11, 20, 33] address the inflexibility of a static application partitioning by using an object-oriented programming model, together with an infrastructure specifically designed for wide-area computing. As a result, these systems can hide some of the complexity of building globally distributed applications and offer more flexibility than client-server or multi-tier computing. But, the very flexibility of a largely unconstrained object-based framework easily leads to unmanageable systems. Furthermore, as these projects are mostly targeted at conventional computers running conventional operating systems, it is unclear how they would scale over the range of devices in a pervasive computing environment.

Mobile code systems, such as Java [13, 21], make it possible to dynamically move functionality between nodes and can thus be used to overcome the limitations of a static application partitioning. Furthermore, by providing a platform-independent execution environment, they represent an attractive solution to the heterogeneity of devices found in a pervasive computing infrastructure. At the same time, mobile code systems, by themselves, lack a comprehensive programming model. Agent systems [23] provide such a programming model for mobile computations. But, they also lack integration with state-based services as well as specific infrastructure for building, deploying, and managing application on a global scale.

We argue that a viable alternative for structuring pervasive computing applications is to build them within a system architecture that provides a select few, yet integrated services and that is specifically targeted at this environment. The system architecture, in turn, relies on a mobile code system to provide the basic, platform-independent execution environment. Such an architecture needs to be simple, so that it can be implemented across the range of devices in a pervasive computing infrastructure. It also needs to be easy to encapsulate, so that applications can be effectively secured and managed while also preserving local autonomy. Furthermore, we argue that the core services of this system architecture should be support for mobile computations, persistent storage, and discovery.

Mobile computations address three major needs in a global computing infrastructure. First, they enable the automatic distribution and installation of applications as well as application upgrades. As a result, Alyssa does not need to worry about maintaining her personal information management software; rather, upgrades are distributed automatically across all her devices. Second, they let applications move between nodes. So, when Alyssa accesses her workspace from a hotel in India, her applications can actually move to the in-room information appliance. Third, mobile computations can serve as software agents and autonomously perform tasks on remote nodes. For example, Alyssa's personal information manager can use an agent to make travel arrangements for her, even while it is disconnected from the travel agency's site.

Persistent storage is obviously necessary to store application data, such as Alyssa's schedule and contacts. But, it is also useful for saving the state of mobile computations so that they can survive node failures as well as reliably migrate between nodes. And, persistent storage can serve as a repository for the exchange of data in a loosely coupled system, thus simplifying the interaction between mobile computations. These uses suggest that persistent storage should be implemented by an object store that provides functionality closer to tuple spaces [12, 35] than to the raw bytes accessible through conventional file systems. In particular, the store should preserve the structure of application data, provide associative access to this data, and support atomic operations as well as transactions.

Discovery [1, 2, 7] provides applications with access to resources whose location or name they do not necessarily know, but whose characteristics match their needs. In a pervasive computing environment, discovery lets mobile devices access resources on the network. For example, discovery lets Alyssa's cell phone access a hotel's in-room information appliance and synchronize schedule and contact information with it. But, discovery should also be applied to resources local to a device, excluding the services for mobile computations and persistent storage. For example, an instance of Alyssa's personal information manager can use discovery to determine the type of user interface supported by the hosting device and consequently adjust how information is communicated to Alyssa.

4 Our Architecture

Our architecture is based on three main abstractions: Tasks represent computations, tuples represent persistent data, and environments provide structure and control. These abstractions are similar to those of the ambient calculus [5, 6]. But, while the ambient calculus defines a formal framework for reasoning about mobile computations in a global network, we focus on building practical systems for pervasive computing.

Tasks execute code, possibly, using multiple threads, and have their own, private state. The basic operations on tasks are create, start, and destroy, as well as check-point to save a task's execution state and move to transfer a task to a different location. Tasks, like processes in traditional operating systems, are isolated from each other, and local tasks interact by exchanging tuples. Tuples are immutable records and are strongly typed in that both their fields and the tuples themselves are typed. The basic operations on tuples are write, read, and take (which reads a tuple and deletes it in one atomic operation). Furthermore, tasks can request to be notified when (specific) tuples are added or written.

In addition to tasks and tuples, our architecture needs a way to group computations and data, to change such groupings, to support interaction between different groupings, and to move both computations and data. It also needs to control the resources consumed by tasks and tuples and to ensure the security of all operations. Environments address these concerns by providing structure and control.

Environments provide structure for computations and data by encapsulating tasks, tuples, and other environments. Tasks execute within an environment and, by default, access only tuples in the same environment. But, in order to enable remote interaction, tasks can explicitly request to access tuples in other environments. Besides the obvious create and destroy operations, the basic operations on environments are move, which moves an environment and all its contents either within a local hierarchy or to a different location, and open, which moves all contents of an environment into the enclosing environment and deletes the emptied environment. Furthermore, tasks can request to be notified when other environments are moved into or out of their environment.

Like nested process structures in other systems [4, 10, 31], the hierarchical nesting of environments provides control. In particular, environments use leases [14] and quotas to limit the resources available to enclosed tasks, tuples, and other environments and to facilitate the effective reclamation of these resources. To ensure security, they also subject all relevant operations to access control and auditing.

To simplify the task of building reliable applications, all operations on tasks, tuples, and environments are fully serializable and support transactions to group several operations into a single, atomic unit. Furthermore, environments provide persistent and recoverable tuple storage as well as the ability to restart tasks from saved check-points. Because operations are fully serializable, we assume that environments that directly contain tasks or tuples reside on a single node or across a local cluster of cooperating nodes. Applications that require data to be replicated across several sites thus need to implement their own replication algorithms, though we plan to support a default replication mechanism as part of our implementation. Environments that solely contain other environments may span several nodes. Such environments can be used to provide functionality similar to that of distributed virtual machines [27] within our architecture by enforcing a uniform policy across all nodes within an organization and by providing a single point of administrative control.

As described so far, our architecture represents a closed system in that it provides just the necessary abstractions to represent computations and data as well as to provide structure and control for them across the network. By default, all other abstractions and services, such as a user interface or an electronic mail service, are outside the scope of our architecture and inaccessible from within it. This *closed world assumption* [28] is a conscious design decision and trades off scalability and manageability (including security) against flexibility. But, as discussed in Section 3, tasks, as well as an implementation of the architecture itself, legitimately need to discover and access such resources, whether they are available locally or within the current networking environment. Environments thus include a well-defined interface for discovering and accessing external resources, which subjects access requests to the same hierarchical controls as those for internal resources.

Naming in our architecture generally is associative. As for tuple spaces, we use templates to describe tasks, tuples, environments, as well as external resources, and, if several resources match a specified template, any of the matching resources can be selected. Generally, a resource matches a template if its type is a subtype of the template and if all fields specified by the template match the corresponding fields of the resource. Field matching is by equality for arbitrary objects, though simple comparison operations, such as larger-than or matches-pattern, are supported for primitive types, such as numbers or strings. To simplify replication, tasks, tuples, and environments can also be named by a unique identifier [19].

To summarize, our architecture is based on three abstractions. Tasks represent computations, tuples represent data, and environments provide structure and control by encapsulating tasks, tuples, and other environments. Because of a closed world assumption, all other resources, by default, are outside the scope of our architecture; though, they can be made accessible through a well-defined discovery interface. Operations are atomic and may use transactions. Finally, naming is associative, using simple matching rules.

4.1 Discussion

Our design reflects the three challenges of heterogeneous devices, limited connectivity, and autonomous administration as follows. First, our architecture has a small application programming interface (API); it is based on only three main abstractions with relatively simple operations. The small API makes it possible to implement the architecture on devices with limited resources, such as cell phones. At the same time, it does not prevent high-performance implementations, such as a cluster of workstations that replicates all tuples in an environment and load-balances tasks across the cluster. Consequently, applications, including Alyssa's personal information manager, can be written against a single API and can use the same binaries across all devices; though, they need to support the different user interfaces.

Second, environments integrate computations with persistent data and provide mobility for both. Applications and their data can thus easily move between nodes and avoid slow links or a future loss of connectivity. So, as pointed out in Section 3, while Alyssa is staying at a hotel in India, her applications and data can actually move to the in-room information appliance. And, her personal information manager can send out a software agent to make travel arrangements for her return to the USA. Furthermore, the use of typed records, that is, tuples, and associative naming simplifies the exchange of data between applications. For example, Alyssa's personal information manager can easily exchange schedule information with a client's scheduling agent, simply by granting it (limited) access to the tuples representing Alyssa's schedule.

Third, nodes in our system are locally autonomous. Anyone can run an implementation of our architecture on any node and populate it with tasks, tuples, and environments. Different nodes can use discovery to connect with each other, and only the participants need to agree on granting each other access. Furthermore, the nesting of environments and discovery in an otherwise closed system makes it possible to effectively control applications. For example, the in-room information appliance in Alyssa's hotel room can use this nesting to ensure that her applications only consume a reasonable amount of resources, only interact with networked services she is authorized to use, such as the hotel's printer, and are purged from the appliance after Alyssa checks out of the hotel.

5 Future Work

We have presented a first cut at an architecture for pervasive computing. Our architecture combines the management of mobile computations, the management of persistent data, and resource discovery into a comprehensive framework and uses a small, yet powerful API based on tasks, tuples, and environments. To evaluate whether this design, in fact, meets the challenges of heterogeneous devices, limited connectivity, and autonomous administration, we obviously need to build an implementation as well as applications that run within it. As part of this effort, we plan to address three important issues, namely, security, routing, and validation.

First, to be effective, security in a global computing infrastructure must be cryptographically strong. This raises the questions of how to effectively manage the necessary keys [22], how to convey authorization [8], and how to ensure the integrity of mobile computations and their data [16]. Second, we expect the transfer of computations and data to be a relatively frequent event in a global computing infrastructure. But, while our architecture provides basic support for moving tasks and environments between nodes, it does not provide a dedicated routing service. This raises the question of how to efficiently support routing, especially when nodes are only intermittently connected, thus suggesting a store-and-forward architecture akin to the one used for electronic mail [25]. Third, while a qualitative evaluation certainly is an important part of an implementation effort, it is not sufficient, by itself, to validate a design. It is thus necessary to develop benchmarks which capture important performance aspects of a pervasive computing environment and which, ideally, also enable the comparison between different architectures.

Acknowledgments

We thank Mark Fiuczynski and Hank Levy for providing valuable feedback on earlier versions of this paper.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of the Seventeenth Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island Resort, South Carolina, Dec. 1999.

[2] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. Addison-Wesley, 1999.

[3] Bluetooth SIG. Specification of the Bluetooth system, Dec. 1999. `http://www.bluetooth.com`.

[4] P. Brinch Hansen. The nucleus of a multiprogramming system. *Communications of the ACM*, 13(4):238–241, 250, Apr. 1970.

[5] L. Cardelli. Abstractions for mobile computations. In J. Vitek and C. D. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 51–94. Springer-Verlag, 1999.

[6] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 1998.

[7] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proceedings of the Fifth ACM/IEEE International Conference on Mobile Computing and Networking*, pages 24–35, Seattle, Washington, Aug. 1999.

[8] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, Internet Engineering Task Force, Sept. 1999.

[9] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next century challenges: Data-centric networking for invisible computing. In *Proceedings of the Fifth ACM/IEEE International Conference on Mobile Computing and Networking*, pages 256–262, Seattle, Washington, Aug. 1999.

[10] B. Ford, M. Hibler, J. Lepreau, P. Tullmann, G. Back, and S. Clawson. Microkernels meet recursive virtual machines. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, pages 137–151, Seattle, Washington, Oct. 1996.

[11] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[12] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley, 1999.

[13] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[14] C. G. Gray and D. R. Cheriton. Leases: An efficient fault-tolerant mechanism for file cache consistency. In *Proceedings of the Twelfth Symposium on Operating Systems Principles*, pages 202–210, Litchfield Park, Arizona, Dec. 1989.

[15] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, Montreal, Canada, June 1996.

[16] N. M. Karnik and A. R. Tripathi. Security in the Ajanta mobile agent system. Technical report, Department of Computer Science, University of Minnesota, May 1999.

[17] P. J. Keleher. Decentralized replicated-object protocols. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 143–151, Atlanta, Georgia, May 1999.

[18] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of Internet stability and wide-area backbone failures. Technical Report CSE-TR-382-98, University of Michigan, Ann Arbor, Michigan, 1998.

[19] P. J. Leach and R. Salz. UUIDs and GUIDs. Internet Draft draft-leach-uuids-guids-01.txt, Internet Engineering Task Force, Feb. 1998.

[20] M. Lewis and A. Grimshaw. The core Legion object model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 551–561, Syracuse, New York, Aug. 1996.

[21] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, second edition, 1999.

[22] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the Seventeenth Symposium on Operating Systems Principles*, pages 124–139, Kiawah Island Resort, South Carolina, Dec. 1999.

[23] D. Milojičić, F. Douglis, and R. Wheeler, editors. *Mobility—Processes, Computers, and Agents*, chapter 14, pages 457–641. ACM Press. Addison-Wesley, Feb. 1999.

[24] J. Ousterhout. The role of distributed state. In R. F. Rashid, editor, *CMU Computer Science—A 25th Anniversary Commemorative*, ACM Press, chapter 8, pages 199–217. Addison-Wesley, Jan. 1991.

[25] C. Partridge. Mail routing and the domain system. RFC 974, Internet Engineering Task Force, Jan. 1986.

[26] V. Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, Oct. 1997.

[27] E. G. Sirer, R. Grimm, A. J. Gregory, and B. N. Bershad. Design and implementation of a distributed virtual machine for networked computers. In *Proceedings of the Seventeenth Symposium on Operating Systems Principles*, pages 202–216, Kiawah Island Resort, South Carolina, Dec. 1999.

[28] P. D. Stout. *Wax: A Wide Area Computation System*. PhD thesis, Carnegie Mellon University, Dec. 1994. Available as Technical Report CMU-CS-94-230.

[29] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, pages 172–183, Copper Mountain Resort, Colorado, Dec. 1995.

[30] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, November/December 1997. Extended version available at `http://www.vbns.net/presentations/papers/MCItraffic.pdf`.

[31] P. Tullmann and J. Lepreau. Nested Java processes: OS structure for mobile code. In *Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, Sept. 1998.

[32] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. WebOS: Operating system services for wide area applications. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 52–63, Chicago, Illinois, July 1998.

[33] M. van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, 7(1):70–78, 1999.

[34] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, Sept. 1991.

[35] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.