Ψ—Pervasive Services Infrastructure

Dejan Milojicic[1], Alan Messer[1], Philippe Bernadat[1], Ira Greenberg[1],
Olaf Spinczyk[2], Danilo Beuche[2], and Wolfgang Schröder-Preikschat[2]

[1] Hewlett-Packard Labs, MS 3U-18, 1501 Page Mill Road, Palo Alto 94304, California, USA
[dejan, messer, bernadat, iragreen]@hpl.hp.com

[2] Otto-von-Guericke Universität Magdeburg, Fakultät für Informatik, Universitätsplatz 2, 39106 Magdeburg, Germany
[olaf, danilo, wosch]@ivs.cs.uni-magdeburg.de

Abstract. Future systems have been characterized as ubiquitous, pervasive, and invisible. They will consist of devices that are diverse in size, performance, and power consumption. Some of these devices will be mobile, posing additional requirements to system software and applications. The focus will move from technology to deployment and ease of use of services. Consequently, traditional paradigms for reasoning about, designing, and implementing software systems and services will no longer be sufficient.

We believe that this future vision will rely on a three-tier infrastructure consisting of back-end servers, infrastructure servers, and front-end clients (mobile or static, handheld or embedded). The critical question for future systems will be how to deliver services on demand from back-end servers to resource-constrained clients. If we can handle the new requirements of these systems, we can enable this computing infrastructure to offer significantly more services to users in a more pervasive way.

1 Introduction

The future of computing has been painted by many visionaries. It was coined as ubiquitous computing by Mark Weiser [37], and D.A. Norman introduced invisible computing [26]. IBM promotes pervasive computing [14], Sybase calls it mobile embedded computing [33], and Sun uses the term Post-PC era [32]. HP Labs' vision is presented in CoolTown [18]. Several umbrella projects in major universities are also exploring these topics, such as Aura at CMU [25], Portolano at the University of Washington [8], Endeavour at Berkeley [11], and Oxygen at MIT [7]. The government is investigating Ubiquitous Computing [6] and Composable High Assurance Trusted Systems [5]. Finally, there are numerous startups in this area, such as StreamTheory [31], Transvirtual [34], and WordWalla [39].

Common to most of these visions are the ideas of blending computers into the infrastructure and providing user-friendly services to non-expert users. The center of gravity is moving from technology to users and services. Mobile and wireless are becoming common rather than the exception. Connectivity and bandwidth are improving, approaching 5-20Mbps for 4G networks in 2005. We believe that the focus of future technology will be in the intersection of the Internet, on-line services, and mobile wireless communication. The environment will consist of globally distributed high-end servers hosting services (e.g., Oceanstore [19]), mid-point servers caching and otherwise complementing service delivery to clients (e.g., Akamai), and a variety of client devices.

Under services, we assume a variety of applications and underlying support (description, look-up, storing state, etc.). Examples include traditional desktop applications, enterprise applications (e.g., project management, expense reporting), personal information management, and various vertical market applications, such as retail, health care, financial, entertainment, and travel.

We are investigating a Pervasive Services Infrastructure (PSI—Ψ in Greek) for delivering Internet services to (wireless) users. The Ψ vision is “Any service to any client (anytime, anywhere)”. We envision that in the future it will be possible to deliver services to clients on their mobile, handheld devices in the same way as it is possible at the desktop today. In addition to the traditional challenges of mobile computing [30], we believe that the biggest challenges of this environment are in adapting services to diverse client devices and in delivering services to clients.

Most devices are resource constrained compared to desktop systems. They differ in many ways, such as user interfaces, CPU, memory size, and power constraints, all of which will require service delivery to be adapted in some way. We are investigating how offloading parts of applications to *mid-point servers* can enable and enhance service execution on a resource-constrained device. Dynamic delivery of services to devices is required to eliminate the need for pre-installed services, to enable the downloading of dynamically composed services, and to support system evolution. For Ψ, client devices and infrastructure servers act as caches for delivering services from back-end servers, thereby improving the performance of remote access.

The rest of the document is organized as follows. Section 2 discusses adaptive offloaded services. Services on Demand is described in Section 3. In Section 4, we present some initial results. In Section 5, we compare our project to related work. We summarize the paper and present future work in Section 6.

2 Adaptive Offloaded Services

Pervasive systems bring a proliferation of devices and infrastructure with differing capabilities and capacities. We believe the scale and diversity will lead to several problems in supporting services on these devices. Consider Personal Digital Assistants (PDAs). While fundamentally doing the same task, specifications have varied greatly for screens, processors, and memory capacities. Providing even a simple service to these similar devices presents a complex task for the software manufacturer who must provide software for the lowest common denominator or must provide separate versions. When computing is pervasive or multiple services are run simultaneously, this problem becomes more acute.

We believe such device limitations can be relieved when devices are universally networked. Instead of hitting resource constraints when supporting a service, other parts of the infrastructure could cooperate to offload parts of the service from devices. However, offloading services in a networked environment is difficult because the available device resources and each device's location may change dynamically. Therefore, we believe that if services are to best exploit this environment, they will also need to adapt to changes in it.

Scalability also poses interesting problems when providing services to so many devices. With just three to five networked devices per user and several million users, the problem of running complex services such as multimedia or interactive services can easily outgrow a single central server cluster. Proxy caches and companies such as Akamai have shown that a multi-tier approach can be used for data to overcome this problem of scale. We believe that service provision can similarly benefit by a multi-tier approach for caching and execution, allowing service offloading into the infrastructure.

To illustrate our vision, consider using a PDA to edit a digital photograph sent to you by URL. This poses a problem for your device, because despite having a color screen there is little spare capacity. Yet, you would like to edit the photograph even if you cannot fully execute an editing application, such as Adobe Photoshop. Instead, the device's runtime uses its understanding of the service, its surrounding devices, and back-end servers to allow the device access to the service, independent of the limitations of the device. For example, the main execution may be performed on a nearby server leaving the device to handle performance sensitive operations and I/O (see Fig. 1). A more capable device might run the main execution and user interface, and use the server to hold swapped memory and to perform intensive image processing.

We believe that dividing service responsibility can be achieved by borrowing resources from mid-point servers (e.g., memory swapping), or by constructing the service from multiple components that are placed and executed separately. Services developed in compositional frameworks are therefore good candidates for our vision. Unfortunately, except for very coarse granularity, few services have been constructed this way. However, we believe that some automatic decomposition may be possible using additional system support in modular systems (e.g., Java).

We believe that infrastructure support for service offloading and adaptivity will allow scalable, high-performance services for a multitude of differing devices in a mobile environment. However, several questions will have to be answered. What service framework requirements would be needed for such distributed execution? Can existing services be automatically split and efficiently placed? Can placement be per-

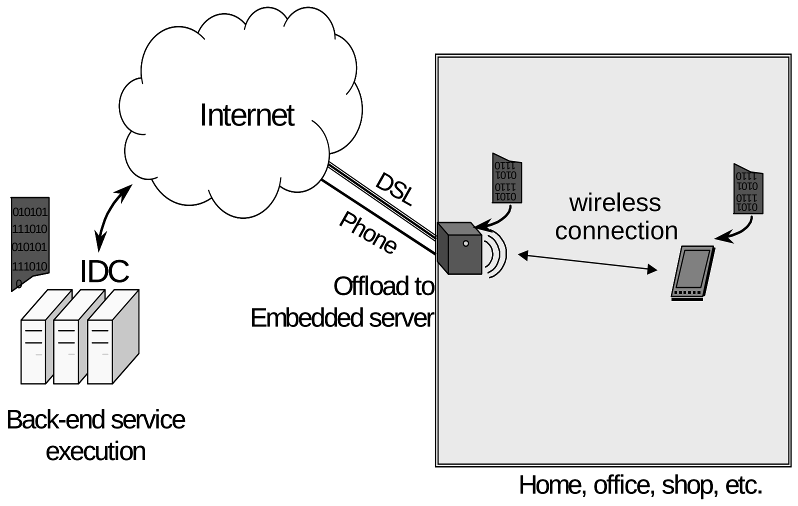

Fig. 1 Adaptive Offloaded Services

formed transparently or should services and runtimes interact? Can services be performant and scalable when distributed across a multi-tier environment? Can service offloading effectively cope with the widely varying characteristics of different devices? Can users roam and still effectively obtain service? What if the service infrastructure decides to migrate parts of the service onto another node, or there are communication errors? If operation is not transparent, it may be necessary to do a service-specific cleanup, such as closing temporary files or restarting, requiring extra application code.

Initially, we intend on investigating the division, placement, and execution support for services. We have started manually offloading part of a service's storage and execution across a two-tier environment (mid-point and client) to study the performance and scalability effects of distributed execution with an educated split in a simple scenario. At the same time, we are examining the characteristics of services (active memory footprint/access patterns, execution paths) to determine whether automatic splitting is appropriate for services. Using these results, we will investigate the effect of roaming in such an environment and the effects of placing the same service on alternative resource-constrained devices.

3 Services on Demand

It is becoming increasingly important to provide services on demand with minimal support from users. In general, users will be less computer savvy and will want to focus on using services, not administering them. Users will connect to the Internet with a wide variety of devices running a diverse collection of system software (e.g. operating systems, browsers). Many services will become available and will be updated frequently. Users will want to access their preferred services and environments from any location and any device.

These trends call for a new system software infrastructure to support services on demand. In this approach, no “a priori” service installation will be required. Instead, desired services will be dynamically located and retrieved, perhaps with service brokers, based on available resources. A trust framework will be needed so that services with an appropriate level of trust can be obtained, and services with different levels of trust can work together. Appropriate billing models will have to be integrated into the infrastructure so that clients can be dynamically billed for the services they use. Because user devices will have volatile storage, user environments and data will have to be securely and persistently stored on storage servers, with support for privacy and integrity. Services, user environments, and data will have to be enabled and loaded on a user's device wherever the user is located, even if the user is moving. The infrastructure should also support disconnected operation for services that can operate in that manner.

Consider the following scenario. While listening to the news, you hear about a new financial service that makes it possible to simulate some investment model. You connect to the Internet, request the service, and start it. The service might also be located by type from a look-up service provided by a service broker. As opposed to a Web-server-based service running remotely, the service may run partially or entirely on your device. In reality, the service can be any type of Java software, and it will be seamlessly installed (or cached) on your device, if it fits. Otherwise, it will be off-loaded to a support server.

Assume that the service saves simulation results as files. Your data is managed by a storage provider, and the files are transparently updated there, at reconnection time if required. You decide to move and will either carry your PDA or use another device at your destination (hotel, airport, etc.). When you log in on this new device, you will provide a user key to the storage provider. The service will be downloaded (if it is not already cached), and your private files and environment will be retrieved (see Fig. 2).

Achieving these goals poses several fundamental challenges. How should device resources be characterized (primary and secondary storage, user interface, peripherals, etc.), and how should service resource requirements be characterized? How can services be acquired and composed with appropriate levels of trust? What service look-up strategies make sense, and how should users be billed for services? What service-dependent reconciliation strategies can be used for disconnected operation?

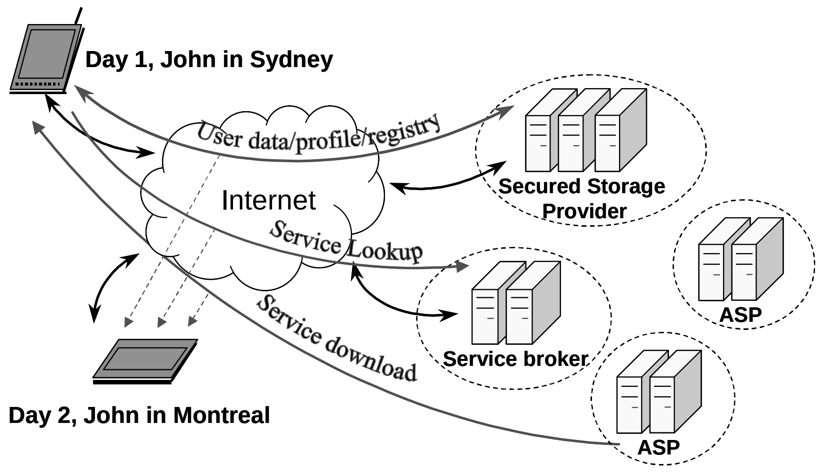

Fig. 2 Services-on-Demand Infrastructure

To answer these questions, we modified Java virtual machine to investigate performance and techniques to interpose file system operations. Asking users to switch to system software that noticeably degrades performance is unacceptable. We want to determine the real cost of using a virtual machine, interacting with a service broker, and downloading services on demand, and to see how much caching helps. We will experiment with client devices that are small, portable, resource-constrained, and JVM-enabled, such as HP Jornada personal digital assistant. Additionally, we will setup a service broker with a sufficiently large number of services to allow us to experiment with real-life situations. In the long term, we will investigate resource characterization and disconnected operation, and investigate a security and trust framework in this context.

4 Preliminary Investigations

We performed two experiments to investigate whether service offloading is beneficial. The first experiment studied the trade-off between the performance/footprint implications of offloading and its benefits in a static scenario. The goal of the second experiment was to gain insight about the dynamic behavior of objects in a Java virtual machine.

The first experiment uses the jPure system [4] on top of the Pure OS [3] to measure the performance of statically offloading the Java runtime support to a server. The jPure system is designed to bring Java execution to systems that are typically too

Table 1 Java Execution Environment Comparison

Java Environment	Memory Consumption & Score		
	Total initial memory consumption (KB)	CaffeineMark overall score	kB per score point
Sun JDK 1.3 (JIT)	7476	1828	4.1
IBM JDK 1.3 (JIT)	8212	5155	1.6
GCJ-Linux 2.95	1416	3109	0.5
jPure (client+server)	312+800	2167	0.5

constrained to run a full JVM. It does so, by manually splitting the Java execution environment into a client (for execution) and a server (for runtime support), thereby reducing the footprint of the client. This system, therefore, represents a simple, static form of offloading from a resource-constrained device. We compared the performance/footprint trade-off of this system against that of a variety of JVMs and Java-to-native compiler-based solutions using the EmbeddedCaffeineMark benchmark. Table 1 illustrates that a fixed offloading of runtime support can offer a good performance/footprint trade-off compared to a traditional Java execution environment, and a similar trade-off to that of a monolithic Java-to-native compiler. The results for the jPure and monolithic Java-to-native compiler show similar performance/footprint trade-offs because they are derived from the same source, but the absolute performance score for jPure is lower because of the cost for remote access of runtime support functions on the server.

For the second experiment, we used the Kaffe [35] JVM to investigate the dynamic behavior of Java execution. The Java memory footprint can often be a limiting concern even with offloaded runtime libraries because applications in large systems can dwarf the runtime support. To investigate the execution behavior of JVM, we modified Kaffe to record access to the pages (4kB in size) used to implement the Java thread stacks and the Java heap for every 10,000 bytecodes executed. Fig. 3 presents the results for the EmbeddedCaffeineMark benchmark, which uses a 1538-page heap with between 125 and 225 pages of live objects, of which only between 6 and 70 pages are actually accessed during the benchmark's runtime. This simple example indicates that there is an opportunity to dynamically offload memory for services because very few pages are simultaneously used. A similar study of object access would probably show an even smaller working set because of false sharing in the pages being monitored.

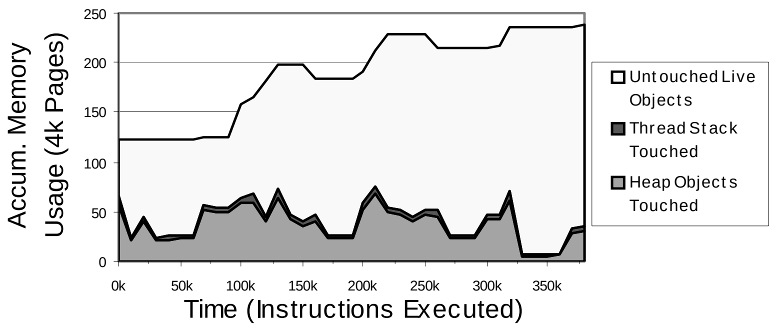

Fig. 3 The Java heap page working set of 'Hello World'

To better understand the possibility of offloading parts of applications, a prototype is being built to split Java applications at run-time and offload part of the application to a nearby server. Fig. 4 presents early results of splitting an execution of the JavaNote editor application (a Java version of notepad). The figure shows two graphs of the execution history of the JavaNote application. Each graph is composed of a set of nodes (Java classes) connected by interactions (method invocations, data accesses). The left portion of the figure shows the execution history of the application as a large text file is loaded. At this point, the application would normally run out of memory because of the client's 8-megabyte heap limit. The right portion of the figure shows

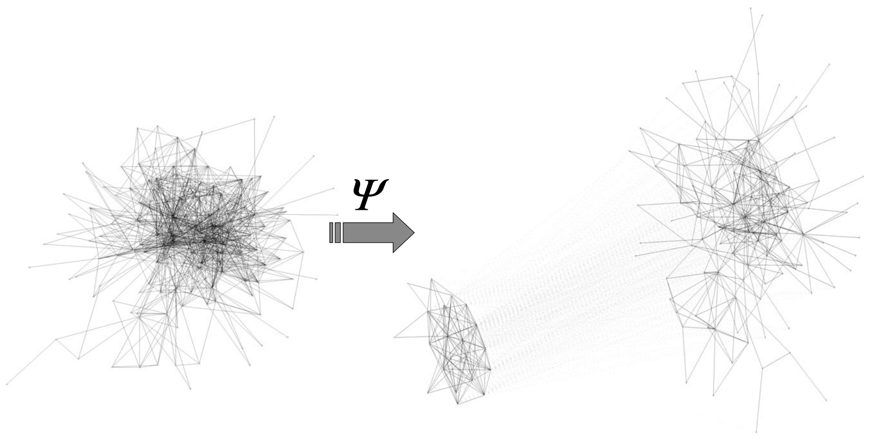

Fig. 4 Partitioning the JavaNote Application

Table 2 Services-on-Demand (SoD) Loading Overhead

Service	Local Loading (s)	SoD Loading (s)			Loaded URLs		Java Classes	
		Intranet	Wave	Dialup	Cnt	Size (B)	Cnt	Size (B)
calculator	0.1	~0	+0.1	+4.7	2	9520	2	9520
calendar	0.2	~0	+0.4	+4.7	4	12952	4	12952
editor	0.2	~0	+0.3	+5.2	7	15885	7	15885
game	0.2	~0	+0.3	+6.1	9	18739	9	18739
agenda	0.2	+0.1	+0.7	+8.4	1	35360	12	59846
ftp	0.6	+2.1	+3.2	+20.9	3	107725	22	142155
mail	2.5	+2.7	+15.1	+129.4	1	675046	138	365574

the execution history after a splitting algorithm split the application state to try to free at least 20% of the memory on the client. The splitting algorithm was based on a run-time analysis of the execution history. Two sub-graphs are now formed, representing local execution on each of the two machines. The interconnections between the sub-graphs represent potential RPCs between the machines. As a result, the Java heaps on both the client and the server would have plenty of free space to continue executing, even though the program has grown beyond the client heap's 8-megabyte limit. Work on this prototype is continuing with the aim of better understanding the splitting of applications in general and their performance.

To investigate the cost of downloading services on demand, a prototype system was instrumented to measure the elapsed time until a service is ready. Table 2 shows the performance of various service classes, ordered by download size. All services are downloaded from servers outside of the HP intranet. The reference is measured when the service is fully cached. The download overhead is only incurred when the service is first used. The relative performance (in addition to reference time) is measured uncached for three connection types:

- **Intranet** within HP (with Web proxy side effects).
- **Wavelan** connected to the HP intranet.
- **Dialup** telephone line.

These preliminary results indicate that, on a reasonable network link of the future, the overhead of downloading and installing services on demand is relatively small (a couple of seconds). Even for large applications that can be compressed in the

JAR format, performance is not unreasonable. However, we can see that for this application (mail) the code size is only 300KB leaving the rest as documentation and graphics. A service-on-demand version of this application might download these supplemental items incrementally or they may be cached to avoid noticeable delays. Overall, these results point towards the suitability of Java applications coupled with a good infrastructure to effectively present services on demand without adversely affecting performance.

Besides service lookup and download, the SoD prototype demonstrates remote storage handling. Using Java bytecode editing techniques, accesses to local resources such as files are interposed and redirected to a storage provider. The infrastructure is not tied to any protocol for remote file systems; instead, it defines a framework where file storage handlers can be to plugged-in. The prototype also includes basic support for disconnection through the use of caches and a simple file inconsistency detection mechanism. We intend to handle reconciliation through the use of plug-ins, because synchronizing file content often depends on the file-type or application.

The current prototype implementation has been successfully demonstrated on laptop and hand-held devices running various operating systems (Windows, Windows CE, Linux) and virtual machines (JDK, Personal Java, Kaffe, and ChaiVM). The following services are currently supported:

- Basic service brokering
- Transparent download and execution of services in sandboxes
- Interposition of a local resource access (File IO) and redirection to storage provider
- Support for a third party FTP client storage plug-in
- Service and storage caches
- Support for disconnected operation
- Simple disconnection/reconciliation modules

5 Related Work

Due to limited space, we do not discuss related work in detail; instead we just compare it with our project. In addition, we omit some related work, such as Rover [15], Coda [23], and GAIA [29].

The **Odyssey** project defines a software platform for application-aware adaptation of diverse mobile applications [25]. This approach considers agile applications in varying fidelities, adapting to system variations, e.g., in network bandwidth. Our work takes a similar view to adaptability applied to services and the system. We are interested in adaptability in a different scope, such as adaptability using computation and storage placement.

The **Ninja** project (as well as earlier work by Fox [9]) investigates a software infrastructure for next generation Internet services [11]. Services are designed to be composable, customizable, and accessible from a variety of device types. The service components can be executed closer to the client to enable transcoding. Our approach takes locality of computation further by considering mid-point servers as locations for computation and storage used by service providers and service clients. In addition, our focus is on offloading services rather than altering service fidelity as with proxy transcoding.

The **Oxygen** project studies software environments for composable applications and systems [7]. Their approach is to use abstraction, specification, persistent storage, and transactions to support change through adaptation and customization.

The **Portolano** projects (Active Fabric and ARCaDE) focus on service provisioning for self-organizing, mobile, composable services; service migration; and automatic service management [8]. Oxygen and Portolano take an active networks approach. We instead consider the computation and storage in the infrastructure to be temporary service caches, with services ultimately originating from back-end service providers.

To support nomadic users, HPL's **CoolTown** project offers a model based on a convergence of Web technology, wireless networks, and portable devices [18]. CoolTown attempts to bridge physical and virtual worlds, whereas Ψ addresses resources and services. CoolTown addresses location dependency and connectivity, while Ψ emphasizes deployment and disconnection.

There are also related industrial standards. Universal Description, Discovery, and Integration (**UDDI**) is a specification that defines a way to publish and discover information about services [36]. Open Services Gateway Initiative (**OSGi**) explores Java platform independence and dynamic code-loading for small-memory devices [27]. Ψ can benefit from either standard.

Coign has investigated the possibilities and merits for the automated partitioning of Microsoft COM applications into client/server implementations [13]. By profiling the interactions between COM components, this work showed that it was possible to construct a tool that could create a good (if not better than human) client/server implementation automatically. Similar work in the **DAP** project at IBM has also considered the importance of caching in these client/server implementations [17]. From a dynamic runtime perspective, the **M-Mail** system has considered runtime offloading decisions for the creation of mail objects on the client or server to improve overall performance and throughput [12]. Our work on Adaptive Offloaded Services extends these approaches to apply offloading at run-time on generic application through the consideration of execution graphs and resource load information.

Finally, there are a number of technologies addressing execution of code on the server rather than on the client. Examples include **Active Server Pages** (ASP) [28], **Java Server Pages** (JSP) [2], and **PHP** [38]. Active Server Pages is a server-side scripting technology that can be used to create dynamic and interactive Web applications. ASP uses scripts which can contain COM components and XML. JavaServer Pages use XML-like tags and scriptlets written in Java to encapsulate the logic of the content for the page. PHP is an HTML preprocessor that enables the creation of dynamic web pages.

ASPs, JSPs, and PHP represent a reverse trend compared to Java applets. Motivation for this server-centric approach is based on software maintenance and administration, security, and performance. Compared to these systems, our work (AOS) addresses the execution of the applications (not necessarily applets) on the clients, as well as the installation and administration of these applications (SOD). We don't think that there is a right answer or optimal solution for all applications. Under some circumstances and for some applications, thin clients will be optimal (the server-centric approach), in other cases, it will be thick clients (the PSI approach). Disconnection, mobility, and the increased deployment of handheld devices will only complicate the trade-offs between these two approaches.

6 Summary and Future Work

We presented our vision of a pervasive services infrastructure. In particular, we addressed the two technologies required to achieve our vision: adapting services to execute on resource-constrained devices and installing services on demand. We believe that both are required to achieve ubiquitous systems. We also presented preliminary results indicating the benefits of offloading and downloading services. If the Ψ vision can be achieved, it may be possible to offer today's desktop services to a variety of resource-constrained devices in a mobile environment.

Our work is equally applicable to traditional wired networks and infrastructures as well as to wireless and mobile systems and services. In the presence of mobility, the traditional requirements for scalability, performance, reliability, and security are enhanced. Nevertheless, we believe that mobility opens up even more opportunities for Pervasive Services Infrastructure support in terms of disconnection, adaptivity to variances in network speed, and support for reliable storage.

We are also interested in investigating service composition. Promising techniques in this area are component-based computing [24] and aspect-oriented programming [16]. The ideal service would consist of a number of components glued together, and would separate placement and configuration from the core functionality, such as in Regis [22]. The service component boundaries would provide points for switching between local and remote execution control of the infrastructure runtime. Java RMI/JavaBeans, Puppeteer [20], and CANS [10] are other examples of component-based systems. Aspect-oriented programming takes a similar approach [16]. A tool called the aspect weaver can be used to connect the components with the implementations of their technical aspects. The result is a loose coupling, which leads to a high degree of configurability at either compile- or run-time. Examples include D [21] and work by Becker [1].

Acknowledgments

We are indebted to G. Candea, C. Karamanolis, K. Keeton, E. Kiciman, M. Mahalingam, D. Muntz, G. Snider, and J. Wilkes for reviewing the paper and/or otherwise contributing to the project. Their comments significantly improved the content and presentation.

Additional Information

For more information on the PSI project, please refer to the following Web site: http://www.hpl.hp.com/research/itc/csl/pss/psi/.

References

[1] Becker, C., Geihs, K., “Quality of Service — Aspects of Distributed Programs,” ICSE'98 Workshop on Aspect-Oriented Programming, 1998.

[2] Bergsten, H., “JavaServer Pages”, O’Reilly, December 2000 (see also http://java.sun.com/products/jsp/).

[3] Beuche, D., et al., “The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems,” Proc. 2nd IEEE Symp on OO Real-Time Dist Comp, StMalo, France, May 1999.

[4] Beuche, D., et al., “JPure - Purified Java Execution Environment for Controller Networks,” Proc. of the IFIP Workshop on Dist. and Parallel Embedded Systems, Paderborn, Germany, Oct 2000,

[5] Composable High Assurance Trusted Systems (CHATS), www.arpa.gov/ito/research/chats.

[6] DARPA ITO Ubiquitous Computing Program, www.arpa.gov/ito/research/uc.

[7] Dertouzos, M.L., “The future of computing, Scientific American,” July 1999. http://oxygen.lcs.mit.edu.

[8] Esler, M., et al., “Next century challenges: data-centric networking for invisible computing: the Portolano project at the University of Washington,” Proc of 5th ACM/IEEE Conf. on Mobile Computing and Networking, Aug 15-19, 1999, Seattle, WA. http://portolano.cs.washington.edu.

[9] Fox, A., et al., Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives,” IEEE Personal Communications, August 1998.

[10] Fu, X., et al., “CANS: Composable, Adaptive Network Services Infrastructure”, to appear at proc. of USENIX USITS, 2001.

[11] Gribble, S., “The Ninja Architecture for Robust Internet-Scale Systems and Services,” Special Issue of Computer Networks on Pervasive Computing, 2000. http://endeavour.cs.berkeley.edu/

[12] Hai Yan Lom, “M-mail: A case study of dynamic application partitioning in mobile computing,” Master's thesis, Dept. of Computer Science, University of Waterloo, May 1997.

[13] Hunt, Galen C. and Scott, Michael L., “The Coign Automatic Distributed Partitioning System, “Proc. of the Third Symposium on Operating System Design and Implementation (OSDI '99), pp. 187-200. New Orleans, LA, February 1999. USENIX.

[14] IBM Pervasive Computing http://www-3.ibm.com/pvc/.

[15] Joseph, A., et al., “Building Mobile Applications with the Rover Toolkit,” Proc. 15th SOSP, Copper Mountain Resort, CO, Dec. 1995, pp 165-171.

[16] Kiczales, G., et al., “Aspect-Oriented Programming,” Proceedings of the ECOOP 1997, Finland. Also available as Xerox PARC, TR SPL97-008 P9710042, Feb., 1997.

[17] Kimelman, D., Rajan, V. T., Roth, T., Wegman, M. N., “Partitioning and Assignment of Distributed Object Applications Incorporating Object Replication and Caching,” 3rd Workshop on Mobility and Replication, European Conference on Object-Oriented Programming (ECOOP), pp. 313-314. Brussels, Belgium, 1998.

[18] Kindberg, T., et al., “People, Places, Things: Web Presence for the Real World. “, Proceedings of the third WMCSA, 2000. see also HPL CoolTown, http://cooltown.hp.com/.

[19] Kubiatowicz, J., et al., “OceanStore: An Architecture for Global-Scale Persistent Storage”, Proc. of 9 ASPLOS, Nov. 2000.

[20] de Lara, E., et al., “Puppeteer: Component-based Adaptation for Mobile Computing,” to appear at proc. of USENIX USITS, 2001.

[21] Lopes, C.V., Kiczales, G., “D: A Language Framework for Distributed Computing”, Xerox PARC, TR SPL97-010 P9710047, Feb. 1997.

[22] Magee, J., et al. “A Constructive Development Environment for Parallel and Distributed Programs,” In IEE/IOP/BCS Distributed Systems Engineering, 1(5): 304-312, Sept 1994.

[23] Mummert, L.B., et al., “Exploiting Weak Connectivity for Mobile File Access,” Proc. of the 15th ACM SOSP, Dec. 1995, Copper Mountain Resort, CO, pp 143-155.

[24] Nierstrasz, O., Gibbs, S., and Tsichritzis, D., “Component-Oriented Software Development,” CACM, v 35, no 9, September 1992, pp. 160-165.

[25] Noble, B.D., et al, “Agile Application-Aware Adaptation for Mobility,” Proc of 16 SOSP, St. Malo, France, October 1997. Aura projects at CMU http://www.cs.cmu.edu/~aura.

[26] Norman, D. A., “The invisible computer,” Cambridge, MA, MIT Press, 1998.

[27] OSGI Service Gateway Specification, available at www.osgi.org.

[28] Powers, S., “Developing ASP Components”, O’Reilly, March 2001. (See also http://msdn.microsoft.com/library/default.asp?URL=/library/en-us/dnasp/html/asptutorial.asp, for tutorial on ASP)

[29] Roman, M., and Campbell, R.H., “Gaia: Enabling Active Spaces,” Proceedings of the 9th ACM SIGOPS European Workshop, Kolding, Denmark, September 2000.

[30] Satyanarayanan, M., “Fundamental Challenges in Mobile Computing”, Proc. of 15 ACM Symp. on Principles of Dist. Computing, May 1996, Philadelphia, PA, pp 61.

[31] StreamTheory, www.streamtheory.com.

[32] Sun Microsystem, “The .com Revolution Meets Consumer Appliances”, available at: www.sun.com/990106/ces/.

[33] Sybase white paper, “Enabling e-Business Anywhere, Anytime: the Sybase Strategy,” http://my.sybase.com/detail?id=1003164

[34] Transvirtual, www.transvirtual.com.

[35] Transvirtual, Kaffe (A clean-room, open source implementation of a Java virtual machine and class libraries). Available off of www.kaffe.org

[36] UDDI Technical White Paper, available at www.uddi.org.

[37] Weiser, M., “Some Computer Science Problems in Ubiquitous Computing,” Communications of the ACM, July 1993, 75-84.

[38] Welling, L., and Thomson, L., “PHP and MySQL Web Development”, SAMS, March 2001. (See also PHP, www.php.net).

[39] WordWalla www.wordwalla.com.