Mobile Agents: A Key for Effective Pervasive Computing*

Roberto Speicys Cardoso Fabio Kon

Department of Computer Science
University of São Paulo, Brazil
{speicys,kon}@ime.usp.br
http://gsd.ime.usp.br

Abstract

The recent evolution of small-sized electronic devices and their growing computational power turned the concept of pervasive computing into a reality not very distant in the future. Researchers are currently developing systems to provide the basic software infrastructure needed for next generation pervasive computing environments. In these systems, however, the possibilities offered by the use of mobile agents are being overlooked. In this paper, we argue that mobile agents, due to its inherent flexibility, can bring a number of benefits to pervasive computing systems. Furthermore, we propose a novel architecture that uses mobile agents to perform three common tasks of pervasive computing more efficiently: system adaptation, component updates and QoS negotiation.

1 Introduction

In the last few years, researchers have been working on new operating system and middleware infrastructures for next generation pervasive computing environments [RC00, GDL+01, GWS01, IBM01]. The traditional approach for distributed computing has to be extended for the pervasive computing domain due to its particular features, thus creating a new paradigm.

The goal of the Gaia project at the University of Illinois [HRC02] is to create a computer system similar to a conventional operating system, but for the pervasive computing world. The Gaia system is responsible for managing a large collection of networked resources and for implementing a variety of services to facilitate application development in a pervasive computing environment. In this paper we focus on a few key issues that such a system must address to be successful.

Networking in pervasive systems is a fragile resource. Network connections of mobile devices are usually unstable and have high and unpredictable error rates. Besides, networking is one of the most energy consuming resources of a mobile computer [SH00] and energy is a scarce resource in mobile devices. As mobile computers are an essential part of pervasive computing, the limitations of mobile networking must be considered carefully.

Pervasive computing environments are always changing. New devices and users become part of the system and leave it frequently. High peaks of utilization are common as well as long idle periods with almost no system activity. The supporting system must have mechanisms to adapt itself to these constant changes; infrastructure services must be able to reconfigure themselves to accommodate changes in the execution environment. To be effective, this adaptation must be as spontaneous and automatic as possible, hiding from the user and administrators the tedious tasks of configuration and system management.

Thousands (or millions) of different devices will use the services provided by the computational infrastructure, which will run hundreds (or thousands) of different software components. It is obviously not feasible to manage this number of devices and components manually. Therefore, there must be a simple way to perform administrative tasks such as installing new software and updating existing components with little or no human intervention.

Finally, pervasive computing aims at using computational devices to augment the user's perception of the world and to help the user to perform traditional tasks more easily. To provide an effective pervasive computing experience, the integration between computers and the real world must occur as seamlessly as possible. Thus, services, applications, and user interfaces have strict quality of service requirements that must be met or otherwise the system will feel unresponsive to its users.

Mobile agents [JvRS95, LO98] can help to solve the above issues in a number of different ways. With asynchronous execution, they avoid long periods of connected activity, reducing network load [LA99], and thus saving energy. Mobile agents encapsulate data and code in a single mobile entity. Thus, an agent can be injected into the network to perform a task on a collection of distributed hosts, or perform tasks on the pervasive environment on behalf of an application, and then return the results of the requested actions on each host to the originating node. On each of these cases, the use of mobile agents can lead to great advantages in terms of performance, flexibility,

*This research is supported by grants from CNPq-Brazil (Kit Enxoval proc. 68.0118/01-2) and FAPESP-Brazil (proc. 01/03861-0).

and scalability.

In this paper, we describe a solution based on mobile agents for (1) system adaptation based on service migration, (2) automatic updates of software components, and (3) QoS negotiation. A prototype implementation of these ideas is underway.

The paper is structured as follows: Section 2 gives a brief overview of our previous work on pervasive computing. Section 3 describes the architecture of the proposed solution and Section 4 reports the current status of the project. Finally, Section 5 cites relevant related work, and Section 6 presents our conclusions on using mobile agents in pervasive computing.

2 Previous work

The work presented here is an evolution of the ideas presented in [KGA+00], where we demonstrate the use of mobile agents for the dynamic configuration of complex, large-scale distributed systems. In that paper, we described a model for automatic configuration of distributed systems, based on the use of push and pull techniques for component updates. In this paper we extend those ideas to the pervasive computing domain, and develop some previously unpublished ideas.

In a previous paper on pervasive computing [KHR+00], we discussed the requirements and problems that ubiquitous computing environments impose on the operating system and middleware and presented a prototype solution based on distributed CORBA services and reflective middleware [KCCB02].

In this paper, we use mobile agents to present an alternative solution for some of those problems, with performance, scalability and flexibility advantages.

3 System Architecture

The architecture we are developing aims at providing a flexible framework for managing three dynamic aspects of a pervasive computing environment, namely Adaptation, Software Evolution, and Quality of Service Negotiation. To achieve this goal, the architecture is divided into three subsystems.

The first subsystem is responsible for service adaptation. It detects changes on the environment that might affect the performance of a service, decides whether or not the service should migrate to another host, and actually migrates the service, if necessary.

The second subsystem controls the component updates throughout the environment. Whenever a new version of a component is released, this subsystem distributes the new version to all relevant hosts. Conversely, whenever an application requests a new version or an installation of a component in which it depends, this subsystem updates or installs the component requested.

The last subsystem is in charge of Quality of Service (QoS) negotiation. Application that can run on multiple QoS levels can use this subsystem to help it to negotiate its hardware and software requirements. The subsystem helps the user to find a suitable node on the pervasive computing environment willing to host the application with an acceptable quality of service level.

One can easily imagine a scenario where these three subsystems are used together to enable an effective pervasive computing experience: A service is running on a node of the distributed system when it perceives a loss of performance, such as a large response time due to an increase in the number of requests for that service. It, then, decides to renegotiate its QoS requirements to provide better quality to its users. It asks an agent to roam the environment to find a node where its new resource requirements can be satisfied (which might be even the same node). The agent finds a new node, but it notices that some of the components on which the service depends are missing on that node. It, then, requests the system to install the components, while it contacts the service to notify it of the results of its search. The service then migrates to the new node where it can run with the new QoS guarantees.

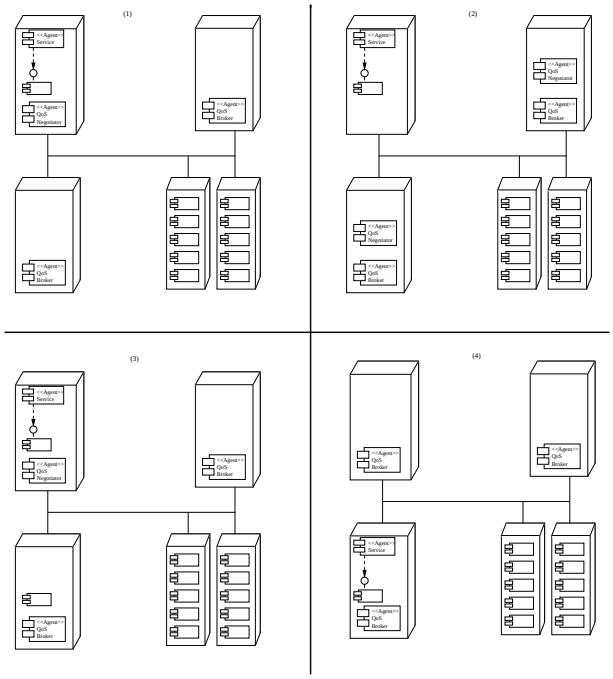

Figure 1: UML deployment diagram of the described scenario. A service notices a loss of performance and requests a new level of resource reservation to the QoS negotiator (1). The negotiator traverses the system negotiating a new contract with each node (2). The negotiator notifies the service of the chosen node, while the node updates any components it might need to run the service (3). Finally, the service migrates to the node and continues executing, with a better QoS contract (4).

In the next sections, we describe each of the subsystems in detail.

3.1 Service Adaptation

Pervasive computing environments are very dynamic. Many devices with completely different architectures enter and leave the system all the time. These devices are integrated to the users' activities and aim at augmenting their perception of the world. Thus, the pervasive services must be very responsive to their users, having small response times, or else this integration will not succeed. Services must fulfill client requests under bursts or peaks with the largest possible efficiency.

On the other hand, there is usually a lot of computational power distributed throughout a pervasive computing system. Services must be able to use all this power and must adapt themselves to changes in the environment, reducing the impact of unusual fluctuations in the system.

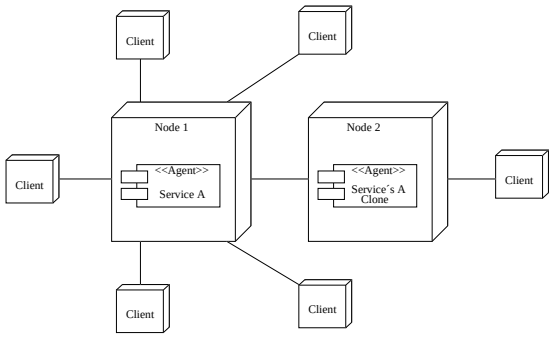

Figure 2: UML deployment diagram for an adaptive service. It shows *Service A* on *Node 1* under heavy utilization. The service notices the dynamic state and spans a copy of itself to *Node 2*. New requests are now served by *Service's A Clone*

Mobile agents can easily migrate from one host to another. By implementing pervasive services as mobile agents and by detecting changes in the environment, such as high CPU utilization, the services can search for a node with lighter load and, using its mobile agent functionality, migrate or span a copy of itself to a new node in the pervasive environment and improve its performance. This also increases the flexibility of the systen, since it makes easier for the service to adapt itself to changes on the environment.

3.2 Component Updates

Pervasive systems are composed of many computational devices, running many different software components. Like in any other environment, software needs to be updated, e.g. due to programming errors or new feature implementations. The point is that an active space has far more computer devices than any other traditional distributed system and that the set of devices that compose the system varies frequently. It is not feasible to update or install components in all of the devices manually. There must be an automatic and spontaneous mechanism for software updates. We call this a *push* approach for component update.

Moreover, software components on a pervasive computing environment may depend on other components to work. Whenever a service migrate to a new host or a new software is installed, the dependencies needed for it to operate properly must be satisfied. In this case, the system pulls the required components from one of the repositories in the Internet. We call this a *pull* approach for component update.

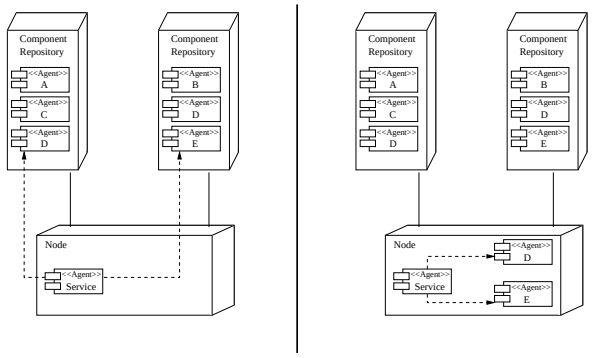

Figure 3: UML deployment diagram for a component update. It shows a service dependent of components *D* e *E*. The node hosting the service does not have the components installed. The system then *pulls* the required components from a remote repository. It receives not only the components, but also all the commands required for its installation and initialization.

Mobile agents can travel across the system carrying component code as its data. Besides, these traveling agents can carry customized code that is able to install and initialize service components in the pervasive computing nodes. All the commands needed to deploy the component are encapsulated in the agent, and may execute in parallel in many nodes. The system can also program a path of nodes to be traversed by a single mobile agent, that would then stop on each node to install the new component.

This approach increases the scalability of the system, since it provides an automatic mechanism to manage components. It also has the benefits of reducing network load, as described in [KGA+00], and increasing performance by executing the deployment commands in parallel throughout the pervasive system.

3.3 Quality of Service Negotiation

A pervasive computing environment is highly complex. Thousands of devices interact with the users and with each other to augment the users' perception of the world and to help them to work, study, or have fun more easily and effectively. The interface between the users and the environment must be simple, yet powerful, enabling them to use the space at its maximum extents.

Multimedia will be a central part of the user interface for pervasive applications. Multimedia interfaces can receive many different kinds of user input and can generate as many kinds of outputs, too. This feature enables

the creation of interfaces with greater expressive power. But, multimedia applications are highly time-sensitive. QoS guarantees are fundamental for an efficient pervasive computing experience. Therefore, services and applications must specify its software and hardware resource requirements to work properly with a good QoS level and to avoid affecting other applications already running in the system.

In Section 3.1, we argued that service migration is an important mechanism to adapt the system and improve its performance. However, careless migration may lead to performance reduction. The migration process is computationally expensive, and must not be carried out without the guarantees that its benefits will not be lost in the near future. Hence, the application must have guarantees that the agreed QoS contract will be respected so that the migration will be cost-effective.

Application QoS negotiation may require a reasonable amount of connected network communication, especially in the case where the application is prepared to work with multiple levels of quality of service. In this case, the application must exchange several messages with the service responsible for QoS negotiation and admission control in the node (sometimes called QoS broker) to agree on a contract.

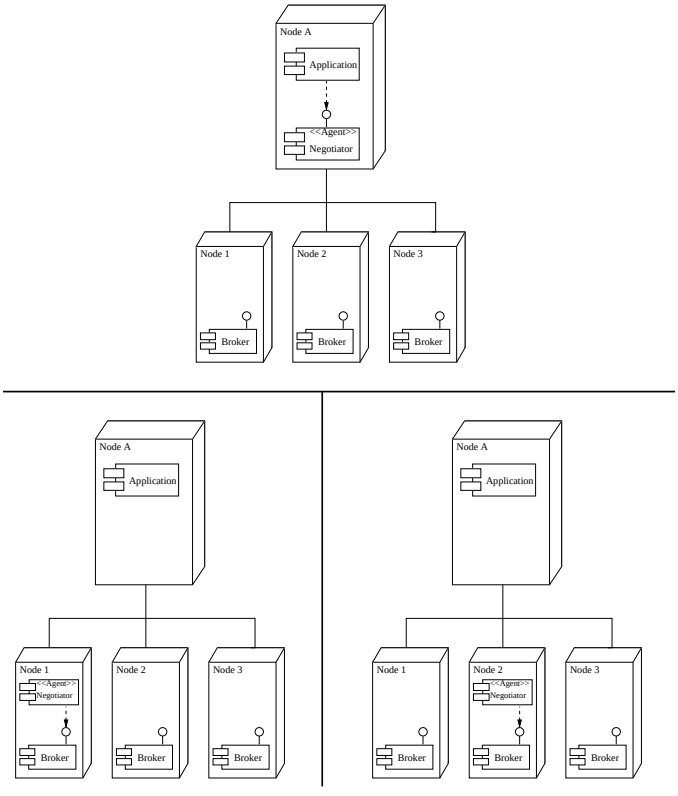

Figure 4: UML deployment diagram of a QoS negotiation. When an application needs to negotiate its hardware and software requirements, it creates an agent with the code that encapsulates the application-side of the negotiation based on the application's multiple levels of QoS. The agent then visits a group of candidate nodes and performs the actual negotiation locally in each node.

Mobile agents can avoid the need of connected network activity. By creating an agent with information about the acceptable QoS levels and the logic to negotiate them, an application can inject it into the distributed system and wait for it to come back. The agent is then responsible for negotiating the resources locally on each host, and to find the host willing to make the most profitable contract in the application's point of view. This approach brings benefits on both performance, by decoupling the QoS negotiation process from the application and assuring performance improvements when a migration occurs, and scalability, since the agents can negotiate QoS guarantees with as many nodes as it might need.

4 Current status

Currently, we are developing a service capable of adapting itself to the circumstances provided by its surrounding environment, such as number of requests or CPU load in its hosting device. For monitoring the load and activity in the pervasive environment, we are using a framework developed in our research group [dSeSEK02].

We are using Java Aglets [LO98] to develop the mobile agents, but we are aware that some pervasive devices do not have enough resources to run a JVM. This problem can be solved by using lightweight agents such as the ones used in the TACOMA project [JvRS95].

A prototype of the complete system will be ready by the end of the first semester of 2003 and the source code will be published on the web.

5 Related work

University of Washington's one.world [GDL+01] is a framework for developing pervasive applications. It is component-oriented and implemented mostly in Java. Typically, each node of the distributed system runs an instance of one.world. Each instance may have many *environments*, which are containers for the *components* and the *tuples*. The *components*, thus, are the functional part of the system, and implement the code for one.world's applications. They store and communicate data using *tuples*, a type of record structure.

Components in one.world can migrate from one node to another as a means for service adaptation. However, one.world does not use mobile agents for this task. Instead, they developed a mechanism for environment migration that has some functionalities of process migration and some of mobile agents. Moreover, one.world neither provides mechanisms for component updates nor means for application QoS negotiation.

IBM's Pervasive Computing Software[1] is a commercial effort to develop real-world pervasive applications and infrastructure for the corporate and service provider markets. At the present time, they are concentrated on four main areas, namely, enabling of voice interfaces (through the ViaVoice family of products), application platforms,

[1] www.ibm.com/pvc

mobile solutions, and development tools. The topics most closely related to our work are application platforms and mobile solutions. Under the first topic, IBM's research focuses on providing the infrastructure to let a customer to migrate its existing web and e-business applications to the new paradigm of mobile computing. Their work addresses problems such as content adaptation and distribution, and access control. The second topic comprises all applications already developed for pervasive computing environments. Even though this infrastructure provides services for distributing software components by using the Tivoli Personalized Services Manager, it supports only the pull approach. Besides, this infrastructure does not yet provide mechanisms for resource reservation or service migration for adaptation.

The Gaia project, at the University of Illinois at Urbana-Champaign [RC00], aims at developing a middleware-level operating system for pervasive computing environments. They want to translate the concept of a traditional operating system, which controls a computer's resources such as memory, processors and disk, to an active space composed of hundreds of pervasive devices. Their architecture is very comprehensive and embraces many different services such as Security, QoS, Automatic Configuration and Component Repository. Even though Gaia faces some of the problems for pervasive computing presented here, our approach is somewhat complementary. Their research in QoS is focused on specification, representation, and requisites for QoS while we build on that structure to provide a new mechanism for QoS negotiation. Finally, while their research on Software Adaptation focuses mostly on adapt applications to the resources available when a user moves, we concentrated on the performance benefits of adaptive software.

6 Conclusion

We believe that mobile agents fit perfectly into the pervasive computing world and that they are a powerful tool to solve many of the problems that arise in such environments.

We described in this paper three common problems in pervasive computing and explained how mobile agents can be used to solve them with performance, flexibility, and scalability advantages over conventional approaches.

Based on the research presented here, we argue that mobile agents are a key factor for developing effective pervasive computing infrastructures and applications. We believe that an extensive use of mobile agents on pervasive computing can increase substantially the flexibility, scalability, and performance of such systems. We also believe that these aspects are being neglected by the ongoing research on pervasive computing and that this fact can lead to the development of intricate and rigid systems that may fail when deployed in real pervasive computing environments with millions of devices.

References

[dSeSEK02] Francisco José da Silva e Silva, Markus Endler, and Fabio Kon. Dynamic adaptation of distributed systems. *16th European Conference on Object-Oriented Programming*, June 2002.

[GDL+01] Robert Grimm, Janet Davis, Eric Lemar, Adam MacBeth, Steven Swanson, Steven Gribble, Tom Anderson, Brian Bershad, Gaetano Borriello, and David Wetherall. Programming for pervasive computing environments. Technical report, University of Washington, June 2001.

[GWS01] Krzysztof Gajos, Luke Weisman, and Howard Shrobe. Design Principles for Resource Management Systems for Intelligent Spaces. *Second International Workshop on Self-Adaptive Software (IWSAS'01)*, 2001.

[HRC02] Christopher K. Hess, Manuel Roman, and Roy H. Campbell. Building Applications for Ubiquitous Computing Environments. *International Conference on Pervasive Computing (Pervasive 2002)*, August 2002.

[IBM01] IBM, www.ibm.com/pvc/tech/whitepapers. *IBM Websphere Portal Server Product Architecture*, May 2001.

[JvRS95] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. An introduction to the TACOMA distributed system. Technical report, University of Tromsø and Cornell University, June 1995.

[KCCB02] Fabio Kon, Fábio Costa, Roy Campbell, and Gordon Blair. The Case for Reflective Middleware. *Communications of the ACM*, 45(6):33–38, June 2002.

[KGA+00] Fabio Kon, Binny Gill, Manish Anand, Roy H. Campbell, and M. Dennis Mickunas. Secure dynamic reconfiguration of scalable CORBA systems with mobile agents. In *Proceedings of the IEEE Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA/MA'2000)*, pages 86–98, September 2000.

[KHR+00] Fabio Kon, Christopher Hess, Manuel Román, Roy H. Campbell, and M. Dennis Mickunas. A flexible, interoperable framework for Active Spaces. *OOPSLA'2000 Workshop on Pervasive Computing*, October 2000.

[LA99] Danny B. Lange and Mitsuru Ashima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–89, March 1999.

[LO98] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, August 1998.

[RC00] Manuel Román and Roy H. Campbell. Gaia: Enabling active spaces. *9th ACM SIGOPS European Workshop*, September 2000.

[SH00] Gerard J. M. Smit and Paul J. M. Havinga. Lessons learned from the design of a mobile multimedia system in the MOBY DICK project. In *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing*, pages 85–99, November 2000.