IEEE pervasive COMPUTING
MOBILE AND UBIQUITOUS SYSTEMS
www.computer.org/pervasive

An Agent-Based Service Network for Personal Mobile Devices

Alessandro Genco, Salvatore Sorce, Giuseppe Reina, and Giuseppe Santoro

Vol. 5, No. 2
April–June 2006

An Agent-Based Service Network for Personal Mobile Devices

The Agent Network for Bluetooth Devices lets users access ad hoc information and high-level services using personal mobile devices enhanced with mobile agents.

Alessandro Genco, Salvatore Sorce, Giuseppe Reina, and Giuseppe Santoro
University of Palermo

Pervasive computing aims to disperse smart devices throughout the real world to supply various services, but such devices should remain invisible to the user—that is, pervasive computing mustn't become *invasive computing*. Yet, we can't hide devices designed to help humans interact with an *augmented environment*—that is, the natural environment enriched with smart devices. Furthermore, such interaction shouldn't be the same for all users. Designing universal interfaces that ignore the differences in people's needs and abilities could lead to system specifications that don't satisfy anyone. We can avoid such problems by exploiting personal mobile devices (such as PDAs and smart phones) for human-environment interaction. Such devices could be suitable as remote controllers or personal I/O interfaces for remote applications.

We propose the Agent Network for Bluetooth Devices, a system that uses personal mobile devices as adaptive human-environment interfaces to supply people with ad hoc information and high-level services. The ANBD system operates with a hierarchical framework of service-providing nodes, dynamically composed and managed by mobile agents.

Our system's novelty lies in its ability to dynamically adapt itself to environmental changes. Furthermore, accessing system services requires little overhead in terms of users' required technical skills and the necessary software. We achieved this result using Bluetooth implementations that are available in most mobile devices. In addition, the services and access modes we propose are straightforward, modifications can be made online, and the only hardware needed to access system services is a cell phone.

System features

We designed our ANBD system with a mobile-agent-based software framework to implement the information retrieval function regardless of the physical network composition. (See the "Related Work in Mobile Agents" sidebar for details on other research.) We chose mobile agents because they can move from one site to another with no initial knowledge of the sites they need to visit. We only need to tell them what to do in each site as they visit it. They can retrieve the address of the next site they'll visit while visiting their current site.

We also designed a framework to support the system's hardware. It comprises fixed, networked devices (wired or wireless) hidden in the real-world environment. The framework's architecture is hierarchical, and we divided the environment into logical areas, each accomplishing specific tasks within the network and each having an appropriate scope. Hence, ANBD is scalable, modular, and general purpose. It could be useful in various locations, including universities, hospitals, large stores, or museums, even if composed of small physical parts such as university departments and classrooms, hospital wards, or wings of large stores or museums.

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/06/$20.00

Related Work in Mobile Agents

In some projects, researchers use PDAs or cell phones with application execution environments as interfaces for devices hidden in the environment.[1] Takuya Maekawa and his colleagues implemented a Web-browsing system for cellular phones.[2] Users of such a system can scroll a page on a remote display or follow the available hyperlinks.

Personal mobile devices can also be used for indoor positioning. We implemented a positioning system by measuring link quality values between mobile devices and fixed devices used as references.[3] We obtained good results, and we're carrying out experiments on algorithms to improve accuracy and optimize the fixed devices' layout.[4]

As far as service provision is concerned, the Iceberg project[5] deals with communication technologies integration. Compared with our proposed system, Iceberg is mostly devoted to low-level service provision, such as voice over IP.

Museums, meeting rooms, and classrooms are typical environments where we can use personal mobile devices' I/O interfaces to access pervasive services. Several projects support users in such pervasive environments by providing them with ad hoc information or media on mobile devices. In this field, Paolo Busetta and his colleagues proposed a system that uses agents for dynamic information generation within active museums,[6] and En-Yi Chen and his colleagues implemented an agent-based system for teaching support in a smart classroom.[7] All these projects require the use of ad hoc software on mobile devices.

Our proposed system differs because the goal is to provide users with services in the easiest and cheapest way. We use mobile agents for system management (service definition and listing) and for service management (information retrieval).

REFERENCES

1. T. Uemukai et al., "A Remote Display Environment: An Integration of Mobile and Ubiquitous Computing Environments," *Proc. IEEE Wireless Communications and Networking Conf.* (WCNC 02), vol. 2, IEEE Press, 2002, pp. 618–624.
2. T. Maekawa et al., "A Java-Based Information Browsing System in a Remote Display Environment E-Commerce Technology," *Proc. IEEE Int'l Conf. E-Commerce Technology* (CEC 04), IEEE CS Press, 2004, pp. 342–346
3. F. Agostaro, A. Genco, and S. Sorce, "A Fuzzy Approach to Bluetooth Positioning," *WSEAS Trans. Information Science and Applications*, vol. 1, no. 1, 2004, pp. 393–398.
4. A. Genco, S. Sorce, and G. Scelfo, "Bluetooth Base Station Minimal Deployment for High Definition Positioning," *Proc. ACM/IEEE Int'l Conf. Mobile and Ubiquitous Systems* (MobiQuitous 05), IEEE CS Press, 2005, pp. 454–460.
5. H.J. Wang et al., "Iceberg: An Internet Core Network Architecture for Integrated Communications," *IEEE Personal Communications*, vol. 7, no. 4, 2000, pp. 10–19.
6. P. Busetta et al., "Service Delivery in Smart Environments by Implicit Organizations," *Proc. IEEE Int'l Conf. Mobile and Ubiquitous Systems* (MobiQuitous 04), IEEE CS Press, 2004, pp. 356–363.
7. E.-Y. Chen et al., "Seamless Provisioning of Service in the Ubiquitous Computing Environment," *Proc. Int'l Conf. Machine Learning and Cybernetics*, vol. 3, IEEE Press, 2003, pp. 1904–1909.

The physical network's definition depends on the physical environment's configuration. In any case, the network will consist of an appropriate set of devices belonging to two groups:

- mobile devices, such as PDAs, smart phones, notebooks, or laptop computers, that are Bluetooth-enabled and equipped with the Java 2 Micro Edition (J2ME) execution environment, and
- fixed devices, such as PCs exploited as database or agent servers, and Bluetooth access points that connect mobile devices with the fixed part of the system.

Another ANBD system goal is for system administrators to be able to make services available to a large heterogeneous audience, so the service network's users shouldn't need programming skills. Furthermore, the system shouldn't require additional software or network access fees. To this end, we restrict agents to operate and move along fixed devices, and mobile devices are only devoted to the human-environment interface.

ANBD layout

We divide the environment in which ANBD will be used into logical areas, each belonging to a given hierarchical level. The number of levels depends on the physical environment's size (see figure 1). We can create two or more logical areas to obtain a simple subnet, which we can then connect to other subnets to form a bigger network. Each area or subnet can be connected to (or disconnected from) the network at any time, thus leading to dynamic distribution levels. Each logical area acts as an autonomous entity when it's disconnected from any other subnet, and users within the area can only access local services.

There are two main area types. *Service areas* (first-level areas) are the smallest entity within the ANBD framework. An SA consists of networked Bluetooth base stations (BT-BSs), and its physical size corresponds to the signal coverage of its BT-BSs. Users can access information stored in the SA central server through BT-BSs used as anchor nodes for mobile devices. We can also connect each central server to a remote server to supply users with more services and to lighten the SA's central server load.

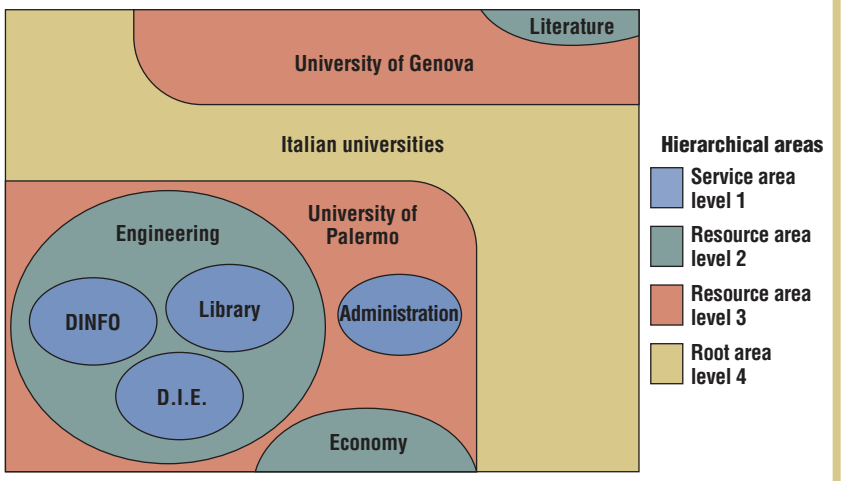


Figure 1. A resource area template for using the Agent Network for Bluetooth Devices system in a university environment. The hierarchical areas indicate the hierarchical level within the ANBD system. D.I.E stands for Dipartimento di Ingegneria Elettrica, or Department of Electrical Engineering, and DINFO stands for Dipartimento di Ingegneria Informatica, or Department of Computer Engineering.

Java Agent Development Framework

TILAB-Italy (Telecom Italia LAB, the research department of Telecom Italia) developed JADE, which is middleware for distributed-agent-based application development fully implemented in Java that complies with the Foundation for Intelligent Physical Agents (FIPA) specifications (www.fipa.org). The framework supplies agents with life-cycle services such as white pages, yellow pages, message transport, and message encoding. The communication architecture offers flexible and efficient messaging based on the FIPA Agent Communication Language.

Several third-party add-ons extend the default platform's capabilities or improve some features. In particular, the Lightweight Extensible Agent Platform (LEAP) add-on replaces part of the JADE kernel (http://jade.cselt.it), forming a modified runtime environment with a reduced memory footprint. JADE-LEAP (http://jade.tilab.com) is intended for a range of devices varying from servers to Java-enabled cell phones. The JADE-S add-on protects a JADE-based multiagent system against security attacks.

Second, *resource areas* (or *n*th-level areas, $n > 1$) consist of a federation of one or more lower-level areas that a control server manages. The RA topology depends on its hierarchical level. A second-level RA contains one or more SAs, and higher-level RAs contain lower-level RAs and SAs. The highest-level RA is the *root area*, which contains all the lower-level areas and doesn't have any higher-level RAs.

Server-side implementation

All area servers and control servers run the Java Agent Development Framework mobile-agent platform with one or more local and remote containers, a database to store user information, and a Web server to access information. (See the "Java Agent Development Framework" sidebar for more details.) Each JADE platform performs specific tasks, selecting the agent most appropriate for each task on the basis of the area in which the agent runs.

Common elements of our system also in a JADE platform, depending on the area type, include

- a *main container*, which stores default JADE agents;
- one or more *containers*, which execute system agents;
- a *logger agent*, which manages user logins and routing services;
- a *setup agent*, which registers each area within the network during the start-up phase;
- *user agents*, which represent each user logged in within the area;
- *connection agents*, one for each BT-BS, which manage connections between mobile devices and servers; and
- *service agents*, one for each available service.

Communication among these agents takes place with FIPA ACL (Foundation for Intelligent Physical Agents Agent Communication Language) messages, which are composed according to an appropriate protocol.

Each BT-BS defines a cell with a limited coverage range, so we must optimize the station's layout to ensure that the entire area is covered. This could lead to overlapping cells, so we defined a protocol to let users with a device within two or more cells connect to the BT-BS with the minimum number of active connections. Connection agents manage this feature to lower the amount of data a single BT-BS must manage and transfer.

Figure 2 shows a sample JADE platform GUI for an SA with four BT-BSs. The figure shows a main container, a Bluetooth_Station container for each BT-BS, and a Local_Services container where service agents run. Each Bluetooth_Sta-

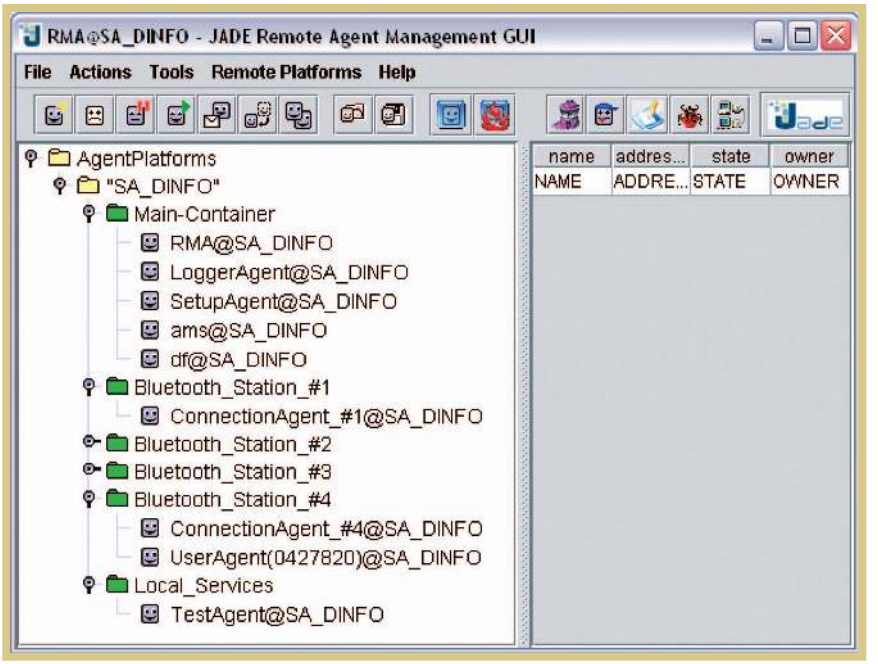


Figure 2. A sample Java Agent Development Framework platform GUI for a service area with four Bluetooth base stations (BT-BSs). In this case, only one user is connected to the service area via the BT-BS number 4.

tion container runs a connection agent and a user agent for each user connected via the corresponding BT-BS.

The main steps to setting up an area platform are *initialization* and *association*. The initialization phase lets system administrators set parameters that will be used during the association phase and service access. During the association phase, the root area or an SA links with an RA to extend the network. Once the association phase successfully ends, users within an SA can access services supplied by another area.

System administrators don't need to set up the entire network association at the same time. In fact, the system can be started with only one subnet, and other subnets can be added later. This is possible because we implemented the initialization and association protocols using agents.

The setup agent's main task is to manage interplatform communication during the initialization and association phases. Setup agents store the area type, the services list, IDs of lower-level setup agents, and the ID of the higher-level setup agent. We store such information in XML files, which setup agents from other areas use to discover local available services. We also use XML files to store the way users can access local services. We made this choice for several reasons:

- The client-side MIDlet can't process big files because of mobile devices' limited computing and memory resources. Furthermore, the Bluetooth data transfer rate (no more than 1 Mbit/sec.) should lead to long response delays. It's worth noting that the average size of exchanged ACL messages is small (150 to 250 bytes).
- Each area contains files for local services only because system administrators can more easily manage decentralized information.
- Each area is autonomous, and even if an association attempt fails, users within that area can continue to access local services. This makes the system modular, scalable, and free from bottlenecks.

(A MIDlet is a program written and compiled with Java 2 Micro Edition that runs on small devices.)

Figure 3. The Area.xml section where the setup agent of the Engineering resource area finds information about other service-providing sites.

```
<ui>
  <list title = "Engineering">
  <item title = "D.I.E.">
    <link url = "http://
      die.unipa.it/xml/Area.xml"/>
  </item>
  <item title = "Library">
    <link url = "http://biblioteca.
      unipa.it/xml/Area.xml"/>
  </item>
  <item title = "DINFO">
    <link url = "http://
      dinfo.unipa.it/xml/Area.xml"/>
  </item>
  </list>
</ui>
```

Regardless of the area type, each area platform stores information needed to access the available services in a local XML file named Area.xml. The system administrator creates an SA's Area.xml file. In this case, the file stores the list of locally available services and information on how to access these services.

An RA's Area.xml file is created by the local setup agent during the association phase with another area. In this case, the file stores the links to the Area.xml files of the associated areas. In other words, both SAs and RAs store an Area.xml file but use it in different ways. An SA uses it to store information needed to access local services, while an RA uses it to list the services provided by lower-level SAs. Setup agents use the Area.xml file to find information about sites they will visit to fulfill a service request. For instance, figure 3 presents the Area.xml section where the setup agent of the Engineering RA finds information about other service-providing sites.

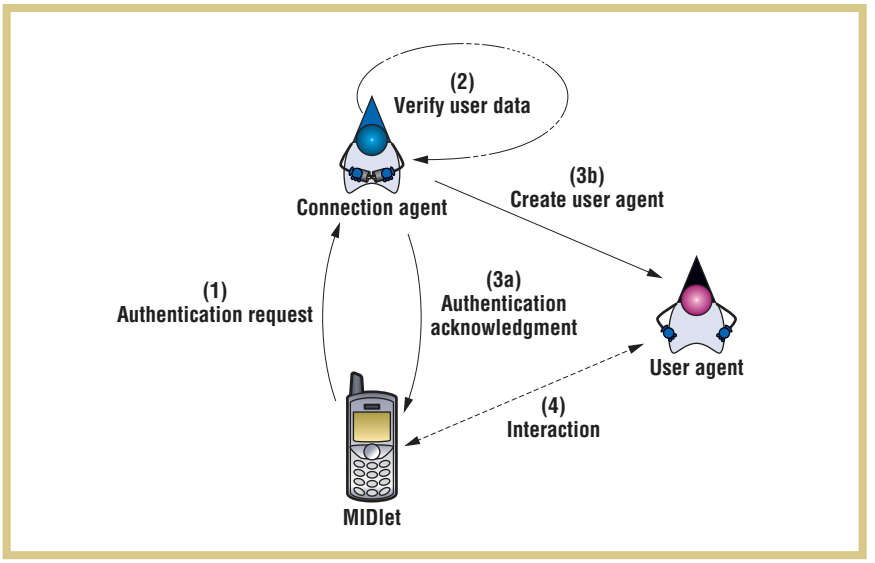


Figure 4. MIDlet connection steps. A step-by-step description of the connection between the MIDlet running on the mobile device and a connection agent running on an SA.

An SA stores another XML file named Index.xml. Setup agents create and manage this file, and the client-side application uses it to list the available options for that area. The Index.xml file is dynamically updated according to the current SA's linking status.

ACL message exchange between corresponding setup agents allows the creation of the link between an SA and RA, so we had to define an appropriate communication protocol. A sample ACL message between setup agents to associate an SA (DINFO—the Department of Computer Engineering) and an RA (Engineering) is

```
Sender: SetupAgent@DINFO
Receiver: SetupAgent@Engineering
Performative: INFORM
Protocol: ADD_PLATFORM
Content: DINFO@http://dinfo.unipa.it/xml/
        Area.xml
```

Login agents accomplish user login and authentication tasks by finding the user record containing personal login data. Each user has a unique ID within the network that's used as a primary key within the user's distributed database.

Each area server stores a local database (the Logger database) that contains the routing and login tables. The routing table stores remote Logger database addresses, and the login table stores local user login data. The login table is initially empty, and records are written each time a user logs into the area. When a user attempts to connect to the area server, its login agent searches for the user's login data within the login table. If the agent doesn't find user login data, it searches remote Logger databases using the routing table. Once it finds the user login data, the login agent writes it in the login table. The next time the same user requires services within the area, the agent will find the login data locally. This principle of operation is similar to a Domain Name System.

Client-side implementation

We implemented the interface between users and system agents using a MIDlet running on every mobile device supporting Java 2 Micro Edition (J2ME) with Mobile Information Device Profile (MIDP) 1.0 or greater APIs and Bluetooth JSR-82 APIs. The MIDlet is the only software that must be executed on mobile devices to access system services, and it consists of a static part executed apart from the connection status and a dynamic part that depends on the available services. The MIDlet negotiates a mobile-device connection with the network using a connection agent (see figure 4).

Once the user is authenticated on a platform, the connection agent sets up a user agent uniquely associated with that user. From now on, the connection agent can continue to manage incoming connection requests from other users, and the MIDlet contacts the user agent for service requests.

The user agent sends the appropriate GUI using an XML protocol message, thus letting system administrator dynamically define service access modes. Hence, administrators only need to create XML files for appropriate GUI generation, thus avoiding MIDlet source rewriting for each needed GUI.

We decided to do this, with no agents running on mobile devices, for two reasons. First, we want people to use their cell phones in the same way they usually do, with the same graphical menu-style interface. We just change the choices available in the menus according to the services available in the user's area. This way, we avoid installing ad hoc software, which could be a difficult task for unskilled users as well as produce different results on different devices.

Our second reason deals with the JADE Lightweight Extensible Agent Platform implementation (see the related sidebar for details). As we discussed earlier, containers are the agent execution environments within a JADE platform. In a JADE-LEAP platform, containers are split into a FrontEnd (actually running on a handheld device) and a BackEnd (running on a Java 2 Standard Edition host), linked together through a permanent connection. This execution mode (split mode) is particularly suited for resource-constrained and wireless devices because the FrontEnd is more lightweight than a complete container, the bootstrap phase is faster, and fewer bytes are transmitted over the wireless link. However, the split mode doesn't support agent mobility. This means that if we want to run agents on mobile phones, we have to deploy JADE-

LEAP in the split mode and only exploit agents' communication features.

Owing to these implementation choices, our system can ignore device coverage problems because user agents perform even if the user is offline (out of BT-BS communication range).

A service request involves a user agent and a service agent. The former acts on behalf of users who need services, and the latter supplies users with the required service. In more detail, when a user requires a service via the MIDlet on a mobile device, the user agent on the platform reads the Index.xml file to obtain data related to the service required and the name of the agent that can provide the service. Next, the user agent asks the appropriate service agent for a service template, which it will use to correctly compose the service request. In fact, the service request must make all the relevant services available within the user's SA and adhere to the proper service provision mode. For example, in the Administration SA, students should be able to check their curriculum and their fee payment status. In this case, the service must be provided in a secure mode to protect personal data transmissions, and the request must contain all the required fields.

Of course, we set up suitable communication protocols (dealing with handshake, authentication, and communication closing), a data packet format, and a template layout to easily code them in XML files.

Case study

We designed the ANBD system to be used in several physical environments; in particular, we carried out some experiments within our university to test our design choices. The university environment consists of departments, each with a specific scope and services (research, teaching, and secretariat) and a typical kind of user (professors, students, and employees).

A possible list of services for users include

- document requests;
- exam enrollment;
- access to teaching information, such as course time and exam dates;
- requests for teaching aids, either directly downloaded onto the mobile device or sent to an email address; and
- access to location-related information, such as department directories or professor meeting times.

When users come within a BT-BS's coverage area, the local Bluetooth service discovers their devices and pushes the MIDlet onto them (only if the users allow the MIDlet installation). We accomplish this with a simple Java application that loops a Bluetooth device discovery sequence. Once a new device is discovered, the application establishes a Bluetooth connection with it and sends the MIDlet .jar file. Most mobile devices' operating systems receive the file as a message attachment, and installation automatically begins upon user conformation. We used Atinav's AveLink API (www.avelink.com/Bluetooth) to implement all low-level interactions between fixed and mobile Bluetooth-enabled devices.

We experienced problems in our tests while pushing the MIDlet to some phone models, most likely due to the different Bluetooth stack implementations. In particular, the Sony-Ericsson Z600 and Siemens S55 don't support the push mode and reported an "unknown-file-format" when receiving the message. However, most Java-enabled Nokia models allow this interaction mode.

We implemented a Web-based MIDlet download mode for devices that don't support this push procedure. In these cases, users can connect via General Packet Radio Service (GPRS) to a URL that corresponds to the .jar file. The device then automatically downloads the MIDlet, but users must start installation manually. We accepted this compromise solution because users must be aware of the MIDlet download in this interaction mode.

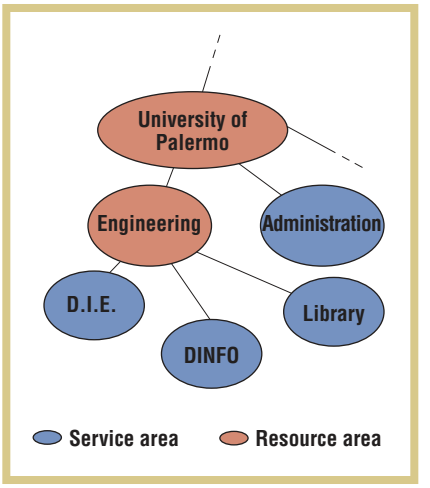


Figure 5. A hierarchical representation of the trial environment. SAs always link to an RA.

A .jad file is generated and attached to the main .jar MIDlet file both in Bluetooth push mode and in GPRS OTA provisioning. The .jad file describes the MIDlet content, its memory requirements, and the certification data, when available. The description file is smaller than the actual MIDlet .jar file (1 to 2 Kbytes versus 24 Kbytes). The .jad file is sent before the .jar file so that the mobile phone's operating system can alert the device's user, who can accept or refuse the MIDlet according to the information received and the execution environment. This reduces the amount of data to transmit and, therefore, the fee users must pay when using the GPRS OTA provisioning mode.

Once installed and started, the MIDlet shows a GUI that is dynamically created depending on the available services.

Figure 5 shows the trial environment with four SAs and two RAs successfully connected. The logical-areas distribution fits the real environment hierarchy. In this scenario, a user might enter the library SA coverage area carrying a smart phone. The MIDlet's static part will show a GUI with the connection option list and dialog boxes for login data introduction. Once authenticated, the client-side MIDlet downloads the appropriate GUI with a

list of possible choices and shows it on the device display. Because related SAs (D.I.E.[Department of Electrical Engineering], DINFO, Library) are connected to the same parent RA (Engineering), the first menu lets the user choose services supplied by one of the available SAs, even if he or she is only covered by one of them. Users can also access other areas connected to the network via a higher-level RA.

In a different scenario, another user might enter the DINFO SA coverage area and connect a device to the network. In this stage of the experiment, we shut off the Engineering RA platform, simulating a fault to test the residual system operation capability. The user can still access services supplied by the D.I.E., DINFO, Library, and University of Palermo areas because the user agent already has all information needed to connect to other SAs. The only users who might experience problems because of the platform fault are those who need to access services supplied by one of the D.I.E., DINFO, or Library areas from an external SA. For instance, users within the Administration SA can't access services supplied by the DINFO SA during an Engineering RA fault.

User experiences

We implemented an exam enrollment service to evaluate the system from the user's point of view. Because the service is useful for students and professors, we were able to test reactions from both sets of users.

Because Italian students must fill in a form with name, matriculation roll, and email address to enroll for exams, we asked 200 students to enroll for their exams using their mobile devices. They filled in the form fields with the appropriate data and sent it back to their user agents, which compiles the XML node.

We received positive feedback from students who found this kind of service provision more familiar than a common Web interface, probably because cell phones are readily accessible and seem to be more familiar than computers among young people.

System agents can act autonomously, and we designed them to always be in a consistent state.

Confirming this informal result, we noted an average of 48 accesses per day from the system log files. The test period lasted 25 days, and we saw a rapid growth from nine accesses the first day up to a peak of 112 accesses in 90 minutes on the fourth day. From the third day on, we had a minimum of 23 accesses per day. These results are impressive when compared with the number of accesses logged for our Web site for the same task in the same period (39 per day average, with a maximum of 91). The activity log file shows that students repeatedly checked their subscription status and the exam room and time. We think this is due to the system's novelty and because the service is free of charge.

Our mobile application is as fast as Web-based access, so we think that students appreciated the ability to access services at any time without having to search for Internet points and, above all, pay for Internet access.

From the professor's point of view, it's straightforward to add, modify, or delete exam information because we defined a service template layout that involves only a few lines of XML code. The seven professors who evaluated our system used it less than the students, but they found it useful when they needed to rapidly change some service-related data. In particular, some professors changed the exam start time just minutes before the scheduled exam time.

It's worth noting that our test subjects didn't report any errors or failures on their mobile devices while executing the MIDlet.

Security issues

The trial implementation of our system didn't deal explicitly with security problems because we were mainly testing the system's operation and usefulness. Furthermore, the databases we're currently using don't store information to be secured. However, ANBD users and programmers are intrinsically provided with suitable security in the three main categories of security problems—execution-environment security, problems raised by agent protection, and communication channels—thanks to the middleware capabilities. Specifically, the limits of agent execution and resource access are well defined within JADE containers. Furthermore, from the agent's point of view, the JADE-S add-on lets users set up enhanced agent-authentication features using, for instance, user fingerprints.[1] Furthermore, communications in JADE platforms use the FIPA-ACL, so it's easy to encrypt messages exchanged between agents.

Concerning robustness, we designed ANBD to adapt itself to environmental changes and faults, providing users with a reduced set of services, if necessary.

System agents can act autonomously, and we designed them to always be in a consistent state. The worst case occurs when a mobile device disconnects during agent setup or data exchange, for instance, because of a battery fault. An agent is only fully set up once the instantiation procedure completes. Therefore,

if the fault occurs before the procedure concludes, the agent simply doesn't exist and must be reinstantiated. This is a less critical situation than an agent left in an inconsistent state. If the fault occurs after the procedure is completed, the agent can start its task and come back to its container until the client can connect again.

Although our system exploits the basic JADE and Bluetooth security features, we plan to enhance system security in the future.

Our future work will seek to improve security features during login, data access, data transfer, and database storage. To let users communicate with each other, we hope to implement a messenger feature that, for example, professors might use to contact students with last-minute messages or students would use to contact colleagues.

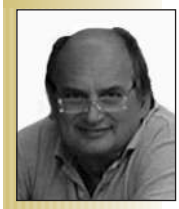
Furthermore, we're evaluating the possibility of letting mobile devices be more involved in the agent platform. We think this could be useful when some local execution is needed, given that next-generation microprocessors for cell phones and PDAs are capable of high-performance computing. Moreover, we're experimenting with using smart phones to store user profiles, in hopes of composing intelligent queries according to users' skills and behaviors.

Owing to space limitations, this article doesn't provide full implementation details on the ANBD system. For further information, please contact the authors directly.

REFERENCE

1. V. Conti et al., "An Enhanced Authentication System for the JADE-S Platform," *WSEAS Trans. Information Science and Applications*, vol. 1, no. 1, 2004, pp. 178–183.

the AUTHORS



Alessandro Genco is an associate professor of computer science in the Department of Information Engineering and a coordinator of the PhD School of Computer Engineering at the University of Palermo. His research interests include mobile networks and pervasive-system middleware design for augmented reality. He received his degree in mathematics from the University of Palermo. Contact him at Dipartimento di Ingegneria Informatica, Viale delle Scienze, edificio 6, 90128 Palermo, Italy; genco@unipa.it.

Salvatore Sorce is an instructor and research collaborator at the University of Palermo. His research interests are in mobile agents in collaborative augmented-reality environments and in parallel and distributed applications, pervasive computing, wireless networks and communications, wearable computers, positioning systems, and personal-mobile-device programming for pervasive systems. He received his PhD in computer engineering from the University of Palermo. Contact him at Dipartimento di Ingegneria Informatica, Viale delle Scienze, edificio 6, 90128 Palermo, Italy; sorce@unipa.it.

Giuseppe Reina is a final-year student at the University of Palermo. His current research is in mobile-device programming for remote-service access. He will receive his degree in computer engineering from the University of Palermo this year. Contact him at Dipartimento di Ingegneria Informatica, Viale delle Scienze, edificio 6, 90128 Palermo, Italy; reina@studing.unipa.it.

Giuseppe Santoro is a final-year student at the University of Palermo. His current research is in mobile-device programming for remote-service access. He will receive his degree in computer engineering from the University of Palermo this year. Contact him at Dipartimento di Ingegneria Informatica, Viale delle Scienze, edificio 6, 90128 Palermo, Italy; santoro@studing.unipa.it.