A Proactive Middleware Platform for Mobile Computing*

A. Popovici, A. Frei, G. Alonso

Department of Computer Science
Swiss Federal Institute of Technology (ETHZ)
ETH Zentrum, CH-8092 Zürich, Switzerland
{popovici, frei,alonso}@inf.ethz.ch

Abstract. An obvious prerequisite for mobile computing devices is the ability to adapt to different computing environments. Otherwise the devices are forced to carry with them everything they may eventually need during their operational life time. This is neither desirable nor feasible, thereby hinting at the need for dynamic adaptation. The idea would be to let the environment be proactive and adapt the application rather than forcing the application to adapt itself to every possible environment. In this paper we present a platform for doing exactly this. Applications running on our modified JVM can be extended at run time with new functionality. Through this platform, mobile devices can acquire on-the-fly any functionality extension they may need to work properly in a given environment. The functionality extensions are local in time and space: they are active only on a specific site and just for the time they are needed. The platform can be used in both centralized settings (with a base station providing the extensions) or in self configuring mode (extensions are provided by peers). In this paper we describe the platform, how to use it and report on one of the several prototypes that have been constructed.

1 Introduction

Device proliferation challenges existing software architectures and creates new types of yet unsolved problems. For instance, a large number of mobile nodes, potentially heterogeneous in nature, is hard to configure and administrate [SGGB99]. Similarly, devices that are continually moving from one location to another need to be able to adapt themselves to the new locations. Otherwise, the devices need to be overprovisioned in terms of functionality so that they can operate in as wide a range of settings as possible. Such an approach bloats the applications, making them more complex and resource hungry. Moreover, there will always be situations that were not foreseen in the design or settings that have changed since application deployment. In those cases, the application will simply no longer function.

In almost all forms of mobile computing, whether it is *nomadic* (a mobile device that changes location and needs to work with different fixed infrastructures) or *ad-hoc* (mobile devices that want to spontaneously interact with each other), the key to deal with such problems is adaptability. The basic idea is that for a mobile device to work properly at a given location, it must adapt itself to that location in both time and space. Spatial adaptation implies adopting the policy and requirements of the *current location*. Time adaptation implies adopting the *current policies and requirements* of the actual location. Both can change at any time and in completely unexpected ways.

To avoid limiting the adaptation capabilities of mobile devices, we suggest not to rely entirely on the abilities of the application. Instead, we argue that there is a need for proactive environments capable of adjusting and extending the functionality of mobile devices. Note that this allows to naturally address both spatial and time adaptation. When a mobile device enters a new computing

* The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322

environment, the environment provides the necessary extensions so that the device can operate in that location at that moment. Of course, the device should be able to discard the extensions once it leaves that particular location. The environment can be anything: a base station, a community of devices interacting spontaneously, or just another device.

As an example of such proactive adaptation, consider a mobile robot used in different production halls. Every time the robot enters a particular hall, it is the hall (e.g., a base station supervising the hall) that adapts the robot to the task at hand. For instance, in one hall it might be necessary to keep track and log every single movement performed by the robot. In another hall, it might be necessary to make sure the robot does not perform certain actions. In yet another hall, every movement of the robot must be sent to another robot that mirrors exactly the movements of the first robot. As soon as the robot fulfills its task and leaves a give production hall, the behavior extensions and additional functionality explicitly added by that hall are discarded.

An advantage of this proactive adaptation is that the program controlling the robot does not need to be aware of any of the extensions appended at run-time. Thus, the program can be kept small and focused on controlling the robots, leaving any adaptation to extensions acquired on the fly. Another advantage is that proactive adaptation allows to extend the functionality of the robot in multiple ways, ways that may not have been foreseen at the time the robot was constructed. Finally, it is possible to change the policies and requirements of a production at any time. Newly arrived robots will simply acquire new extensions that reflect the new policies. Robots already in the hall will be adapted by removing the old extensions and replacing them with the new ones. Moreover, bugs, fixes, or evolution of the software running in the robot can be done through extensions so that the robot is kept functional until there is an opportunity to replace the code in the robot.

Similar scenarios for adaptability exists in a multitude of mobile computing applications. An example are PDAs entering a building being adapted with an encryption layer, a persistence module, and a filter that prevents using certain resources. Another example are accounting modules being added to mobile devices (e.g., lap-tops) to bill them for the use of services in a given location. In all these examples, the key aspect is that applications do not need to establish beforehand what they can and cannot do. Adaptation takes place through a proactive environment capable of delivering the necessary extensions to the mobile devices, devices that must carry a platform to dynamically acquire, apply, and discard extensions as needed.

In this paper we describe a complete system that can be used to implement such scenarios. The system comprises two layers. The first layer resides in each mobile device and provides the support for adaptation, i.e., it provides the ability to apply *run-time extensions* to applications. For this first layer we use the PROSE dynamic AOP system [PGA02,PAG03]. The second layer is in charge of distributing and managing the extensions. This second layer typically resides in a base station but can also be embedded in mobile devices for exchanging extensions in a peer-to-peer manner. This layer is implemented using MIDAS (MIddleware for ADaptive Services), a Jini [AWO+99] based system that can deliver extensions to mobile devices using a wireless network. Together, the two layers can be used to extend the functionality of either single devices or entire communities of mobile devices.

The rest of the paper is structured as follows. Section 2 motivates the proposed architecture by looking at the infrastructure and requirements. Section 3 presents the core functionality of the system to enable adaptive nodes, and illustrates the concepts of MIDAS. We then show in Section 4 how to use MIDAS for adapting robots in an industrial setting. Section 5 concludes the paper.

2 Motivation

In this section we state the requirements for proactive adaptation and discuss an infrastructure that could address these requirements. We also comment on related work.

2.1 Requirements

As pointed out in the introduction, adaptation is a key approach to deal with the variety of computing environments and changing settings that a mobile application will encounter during its operational life time. Such adaptation can be achieved in many different ways. However, feasible solutions must take into account a number of important requirements.

A first requirement is for the extension mechanism to be generic rather than application specific. In the same way mobile devices cannot foresee all possible situations they will encounter, it is also not possible to predict which applications will require adaptation. Furthermore, since proactive adaptation requires a certain infrastructure, it is also not reasonable to provide such infrastructure on an application basis. Whatever the infrastructure is, it must work with a wide range of applications.

A second requirement is for the extension mechanism and supporting infrastructure to be entirely symmetric. In other words, if a mobile device is capable of receiving extensions, it should also be able to provide extensions to other nodes. Such ability does not need to be used in all cases but should not be excluded by design. For reasons of space, we concentrate in this paper mostly on solutions involving a base station although the ideas presented and the system being described can be used without modification in a peer-to-peer setting.

Finally, and for obvious practical reasons, the mechanism used for adaptation through functionality extensions must be secure to avoid that it is misused and tampered with. Secure adaptation involves two aspects: making sure that the extension comes from a trusted party and making sure that the extension does not access system resources if it is not supposed to do so.

2.2 An infrastructure for proactive adaptation

Interactions between clients and service providers have been traditionally supported by middleware. Traditional middleware mainly helps to provide a uniform view of a system, in spite of the possible heterogeneous nature of the underlying components. The middleware also provides functionality that facilitates the development of applications over such heterogeneous components. For instance, the Corba Component Model [CCM97]) adapts services with transparent middleware functionality for implicit context, authentication and authorization, etc.

In conventional settings, such service adaptation is based on a fixed server architecture. What we propose is to make any mobile computing environment act as a middleware server capable of adapting at run time any application entering that environment. Since doing so for any possible mobile computing environment would be next to impossible, we have concentrated our first efforts in Java based applications. Hence, the idea is to provide a nomadic infrastructure [BCKP95] where applications running on a JVM can be provided with extensions (also written in Java) that modify their behavior. The problem turns then into how to adapt Java programs at run time and how to manage and distribute Java extensions under the constraints imposed by the requirements listed above. In what follows we describe step by step how these two problems can be solved.

2.3 Related work

The type of adaptation we advocate is somewhat different from the conventional notions of adaptation and context awareness [SAW94]. What we propose is to dynamically extend or modify the functionality of an application. This is different from adaptation based in sensing the environment and choosing between different pre-programmed options. It is also different from adaptation based in obtaining data about the environment (either by the application itself or the environment provides the data) and changing behavior according to an established program. In what we propose, adaptation means adding functionality that was not there before. This is difficult to achieve with current technologies. The reason is that there is no generic way to augment at run-time the functionality of an application unless this was foreseen at development time.

The explicit participation of the environment in the adaptation of underlying applications has been explored in the Odyssey system [NSN+97]. This form of adaptation is known as application-aware adaptation. Application-aware adaptation has been used, for instance, to hide the effects of

mobility using replication and cache consistency techniques. Conceptually, this work is related to our approach since it also advocates an active implication of the infrastructure in the adaptation of applications.

The same need to shift a part of the adaptation logic away from the application has lead to approaches that propose new software architectures to support adaptive systems [ECDF01,KF01]. Thus, ICrafter [PLF+01] advocates the move of intelligence from the end-points to the resource-rich infrastructure. ICrafter uses pattern matching techniques to overcome the need for standardized interfaces, and relies on user interface generators to create functionality for new services. The user interface generators in the ICrafter design roughly correspond to the application-aware infrastructure we want to associate to each location. However, it is application specific and some of the ideas might be difficult to generalize. In our case, we use aspect-orientation techniques [KLM+97,OL01,BH02] to ensure an application and adaptation neutral platform.

Finally, it is worth mentioning the ongoing work in dynamic and adaptive middleware. An example is an adaptive service layer in CORBA [ZBS97] that provides horizontal support for the simultaneous adaptation of several applications. A step further is represented by reflective middleware [APW01,CBCP01,Bea01], which opens the definition of the infrastructure and allows to dynamically reprogram the service layers. Implementations of reflective middleware can be found at the CORBAng [EGK+99] project which uses meta-models to structure a meta-space. Some approaches [YK01] even propose dynamic hardware-reconfiguration to support adaptability. The Cactus project [HSH+99,CHS01] is another good example of how adaptive software systems can be used in distributed environments. While these approaches represent considerable progress, we believe our approach complements them by addressing a more generic form of adaptation.

3 System architecture

In a first step, we describe how to extend the functionality of an application at run-time. In a second step, we describe the management layer for extensions.

3.1 Step 1: Generic support for run-time extensions with PROSE

There are many similarities between the problems addressed by conventional middleware architectures for fixed computing and the type of adaptations we envision for mobile settings. For example, when the problem of passing implicit context information along a remote call translates into *adding* functionality at a *large number* of points in the execution of an application, such as all incoming and outgoing method calls.

In such cases, it is not sufficient to instantiate new components into an existing service. One must actively modify existing components. With this in mind, we turned to *Aspect Oriented Programming* (AOP) [KLM+97] as the most suitable approach to address this problem.

AOP allows adding extensions to an existing application. AOP is originally intended for extensions that cannot be easily expressed using traditional object-oriented techniques like inheritance. The description of such extensions is based on the concept of *aspects*, the part of a software system that affects the behavior of a component. An aspect is defined by a *crosscut* and a *crosscut action*. A simple aspect example may be:

```
before methods-with-signature 'void *.send*(byte[] x,..)'
do encrypt(x)
```

This aspect specifies that in all methods whose name starts with "send", and which receive a byte array as a parameter, the byte array must be first encrypted. The crosscut of this aspect is the collection of method entries in a given application that matches the specified signature patterns. In AspectJ [XC02,LK98], e.g., crosscuts contain patterns for matching the invocations of method(s) of a set of classes, access and modification of objects fields, and exception handling. The crosscut action (here, the encryption of the byte array) is the code to be inserted at (before or after) the points defined by the crosscut. The act of inserting the new code, thereby changing

the behavior of the application, is performed by a so called *weaver* tool. Weavers are typically based on a preprocessor or a specialized compiler as AOP was originally designed as a compile time technique.

Such platforms for aspect-oriented programming [XC02] are not appropriate for expressing run-time adaptations, because they bind aspects (extensions) and application classes at compile-time. The alternative is to modify the application code at run-time. For this purpose, we have developed PROSE [1] (PROgrammable extenSions of sErvices), a system in which aspects are first-class Java entities, and all related constructs are expressed using the base language, Java [PGA02,PAG03]. PROSE allows programmers to:

- adapt the functionality of a *running* application by dynamically injecting an extension
- make adaptation secure by providing the appropriate protection from malicious extensions that may use the proactive adaptation as a trapdoor.

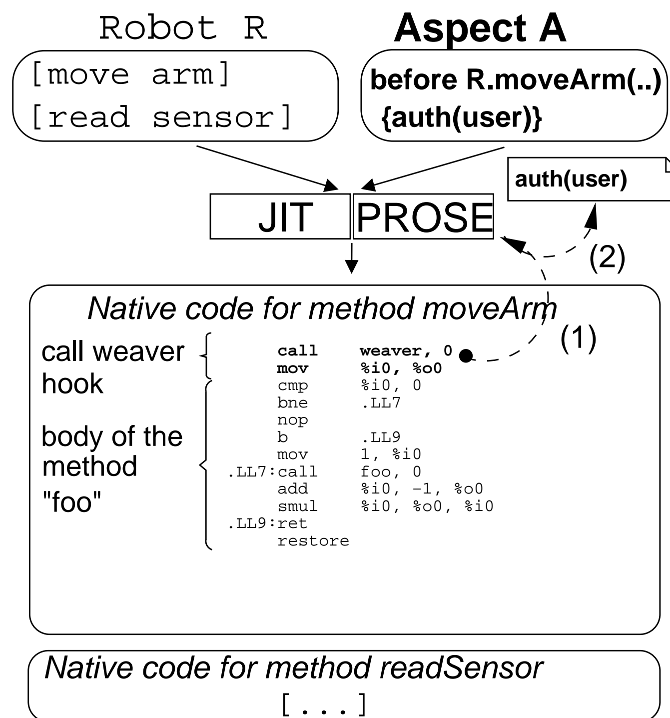

Fig. 1. The run-time adaptation process in PROSE

Addressing run-time adaptation in PROSE To add functionality at run-time, PROSE leverages the fact that most modern JVMs [Sun02,SOT$^+$00] uses a just-in-time (JIT) compiler. A JIT compiler continuously translates at run-time code being interpreted by the JVM into native code. The native code is equivalent to the Java byte-code, but can be executed more efficiently. PROSE adds extension functionality by instructing the JIT-compiler to insert additional actions (advice code) when transforming the bytecode into native code.

Consider as an example the problem of extending the behavior of a robot R. For simplicity, assume that the movements of the robot are controlled by only two methods, *moveArm* and

[1] PROSE is available for download from our web site `http://prose.ethz.ch`

readSensor. An aspect A defines the policy for the robot actions in a given production hall. A will be woven through the robot application whenever R enters the production hall. For simplicity, we consider that A adds the middleware functionality for authorization. With this, each execution of $R.moveArm$ is dynamically extended such that it is preceded by a call to *auth(user)*. By this transparent authorization, the production hall has the ability to prevent the robot from executing actions on behalf of clients that are not authorized. Figure 1 shows how PROSE modifies the translation of Java bytecode into native code. R's functionality is translated into native code, and PROSE adds *minimal hooks* (or *stubs*) before the actual code of the methods. Stubs must be woven at *all* potential join-points in R's code (such as field changes, method boundaries, exception throws and handlers, etc.). In our case two native instructions are added before the native code corresponding to *moveArm*. Every time *moveArm* is called, PROSE is notified (step 1). When this happens, PROSE checks whether any additional action must be executed, and eventually executes all actions corresponding to that join-point (step 2).

This layer of indirection – the stub code – leads to an increase of the resulting code size (since code is added at locations where no advice is needed). However, given the small size of the minimal hooks, the impact on performance is small [PAG03].

Addressing secure execution in PROSE Functionality extensions received from foreign hosts, could contain malicious code. To prevent this, PROSE was designed in such a way that the extension code is entirely isolated from the original code of the application. This allows practically any Java application to use the standard Java security model [SM] to run in a sandbox the extensions received from remote hosts . Through this, PROSE defines an *aspect sandbox* in which interceptions, although spread through various components, are treated as if they belong to the same component.

With PROSE on every mobile node we gain the capabilities of AOP together with the ability to perform the weaving at run-time without disrupting the application. With this, we achieve the necessary generality as well as the support for dynamic adaptation we are aiming at.

3.2 Step 2: Extension management with MIDAS

When every mobile node runs on a PROSE-enabled JVM, it can be extended at run-time – provided that an extension is woven into the system at the right time and place. Adding and removing extensions, and guaranteeing that the right extensions are inserted into the appropriate nodes is an important task that guarantees the locality of adaptations. This task – the *extension management* – is provided in our architecture by MIDAS. MIDAS builds on top of PROSE and provides the following services:

extension distribution: discover new nodes joining a local environment, distribute extensions to them and then activate these extensions using PROSE,

locality of adaptations: keep extensions alive for the time a mobile device reaches that location, revoke extensions for those nodes that leave the location, and allow the replacement of obsolete extensions with new ones in case the local policy evolves or it is changed, and

security: enhance the sandbox security model provided by PROSE with a trust model in which extensions are accepted by mobile nodes only if they come from a trusted party.

Addressing extension distribution To achieve this goal, MIDAS separates nodes into two roles. *Extension base* nodes contain a list of extensions. They discover new nodes joining the network and send extensions to the newcomers. *Extension receivers* can get extensions from extension bases. We assume that each extension receiver has PROSE activated on its JVM. When it obtains an extension from an extension base, it immediately inserts the extension using the PROSE API. Extension receivers also discard extensions when they leave a network or lose contact with the extension base.

By appropriately assigning extension base and extension receiver roles, one can achieve various forms of adaptations. At one extreme, each node can contain an extension base. When it joins a new community, it distributes its extensions and receives others from the existing nodes. This type of organization is appropriate for creating an information system infrastructure in an entirely ad-hoc manner. At the other extreme, each physical location may have a base station as extension base. All other nodes (e.g., the mobile nodes) are extension receivers. This organization is appropriate for adaptations that correspond to infrastructure and organizational requirements. Between the two extremes, many other configurations are possible.

Addressing revocation of extensions The proactive platform must be designed for device mobility. This implies that extensions must transiently adapt a service (for as long as the service is working in a given space). To model this behavior, the extensions are *leased* to each node (i.e., to the adaptation service of a node). It is the responsibility of each extension base to keep alive the functionality it has distributed among nodes. When a node leaves a given space, the leases on the extensions acquired in that space fail to be renewed and they will be discarded. Each extension is notified before leaving a proactive space so that it can execute a shut-down procedure ensuring that all current operations are completed and a consistent state is achieved. The revocation service is achieved as follows:

1. each MIDAS extension base keeps track of its extension activity (what nodes where adapted, at what point in time) and optionally implements a simple roaming algorithm to deal with nodes migrating between areas.
2. each MIDAS extension receiver keeps track of what extensions have been obtained from what base. If a MIDAS base fails to keep a given extension alive, the extension is immediately withdrawn from the system. By autonomously withdrawing extensions, extension receivers address the space and time dimension of adaptations.

Addressing security The layer of security provided by PROSE (in which extensions are run in a sandbox) is enhanced by MIDAS with an additional layer of verification. In MIDAS each extension instance has to be signed. This ensures that the received extension has been instantiated and configured by a trusted entity. The verification of the originator of an extension is done before insertion of the extension in PROSE. Each extension receiver node (and thus each mobile device) may define its preferences and trusted entities.

3.3 Example of MIDAS

The best way to describe how MIDAS works is through an example. Consider a service m_R exported by a robot (R). Figure 2.a illustrates this situation. What we would like to do is to adapt the functionality m_R of robot R as the robot enters a production hall. This adaptation occurs through the adaptation service that the robot carries with it (Figure 2.b).

The first step in the adaptation process is to detect the adaptation service of the node. For service detection and brokerage, one can use existing platforms for spontaneous networking. In our case, we have chosen Jini [AWO+99]. The adaptation service advertises itself as a Jini service, thereby announcing its presence to the environment (assume for simplicity that the environment is a base station). The environment recognizes the adaptation service and, therefore, knows that the node can be adapted. Let's assume the production hall has a set of predefined adaptations. Furthermore, assume that these adaptations implement an access control policy and a quality assurance mechanism that logs persistently all changes to the state of a robot (represented as $*$ in Figure 2) in a database associated to the production hall.

The two adaptations are sent to the adaptation service of the new node as aspects specifying how and where the application has to be changed (step 1 in Figure 2.b). The activation of the aspects comprises two steps. The first is to include in m_R the code necessary to trap the execution at the appropriate points (step 2' in Figure 2.b). The second is to instantiate the extensions that

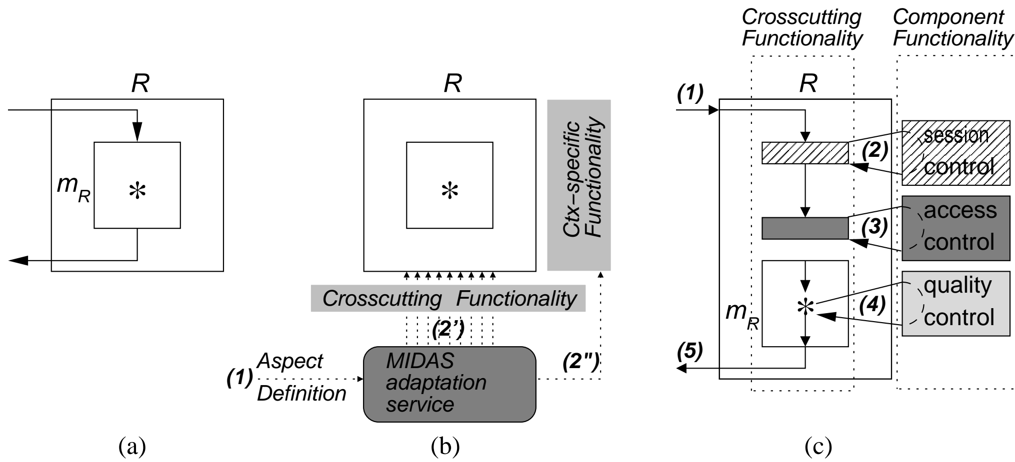

Fig. 2. (a) Remote method call of m_R on a node (b) Node containing the adaptation service and (c) Remote method call of m_R after the node is adapted.

will carry out the adaptation (step 2" in Figure 2.b). Once the adaptation service has activated the incoming aspects, node R reaches the state shown in Figure 2.c.

At this stage, the node is adapted and its functionality has been modified (Figure 2.c). When m_R is invoked (step 1, Figure 2.c) and before the method m_R is executed, a first interception occurs (step 2, Figure 2.c). This first interception is used to call a module that extracts session information like the callers identity. After the execution of this first extension is completed, another interception occurs (step 3, Figure 2.c) that will invoke the access control extension. The access control extension uses the session information to determine whether the call should be completed or not depending on the policy defined as part of the extension. If the call can be completed, the execution of m_R begins. Assume that as part of this execution, the robot changes its internal state (*). These changes are intercepted and propagated by the quality control extension (step 4, Figure 2.c) to a database at the base station. Once the changes are safely stored, execution of m_R resumes and, upon completion, the results are returned to the caller (step 5, Figure 2.c).

The important issue to understand in this procedure is that R needs to carry neither the interception points nor the extensions. All R needs is a PROSE enabled JVM and have the adaptation service. The rest is provided by the context and dynamically added to the application.

For simplicity we have omitted many details of the execution of the adaptations. In addition to the ones described, there are other adaptations that are transparently added to R. These adaptations take care of marshaling and unmarshaling arguments, adding and removing MIDAS specific information to each call, etc. Of the extensions used as examples, the session management extension is an implicit extension needed to implement other extensions (like the access control). When an extension that requires session information is added to a node, the session management extension is automatically also added to that node. The access control extension is an example of an adaptation that does not require to know the source code. It is enough to know the published interface of m_R. There are also many useful extensions which don't know anything of the application, not even the interface. For instance, it is very easy to design an extension that will encrypt every outgoing call from an application and decrypt every incoming call. Another example is a variant of the logging extensions that records every call to an application.

4 Application development for proactive environments

We have already used MIDAS to implement several prototypes and various forms of extensions (e.g., [PA02,PAG03]). These prototypes have been used for testing and benchmarking. They pro-

vided us with feedback on the overall functionality, and we could determine how easy was for a programmer to start working with an extensions.

4.1 Basic design

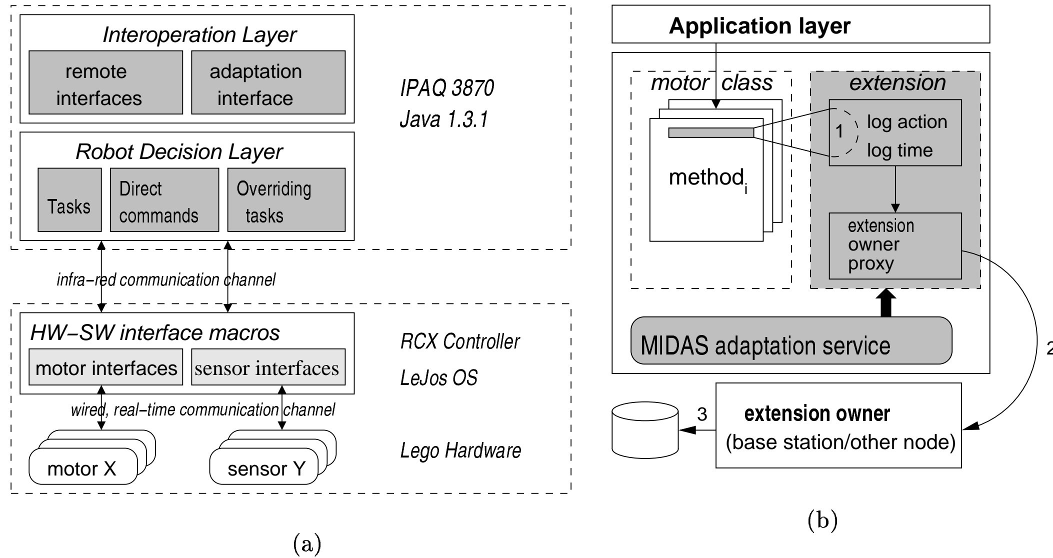

Fig. 3. (a) Software architecture in each robot and (b) the extension application for hardware monitoring.

Once the adaptive middleware infrastructure is in place, developers can start to design concrete environments that impose their own policy and services to all applications within their boundaries. This involves identifying the necessary adaptations, identifying where they will take place, and building all the interfaces required by the environment. In the case of the robots, we employ the RCX controller available in Lego's Robotics Invention System [Leg] as the platform for developing the robots. For communication purposes we use Jini, although any other protocol for spontaneous interaction (e.g., [LCX+01]) could be considered.

Figure 3.a shows the basic architecture of the software attached to each robot. The upper layer defines the functionality for inter-operation with other nodes. Depicted from left to right, this layer defines (i) the services the node makes available to other nodes, like event processing, interface publishing, or lease management (provided by Jini in this case) and (ii) the adaptation service of MIDAS.

The second layer defines the application logic of the robot. This layer defines small programs (tasks) that define an objective for the robot (e.g., searching for a particular object). A task is a basic program that decides what the robot is going to do. A task is broken into activity requests (hardware macros) that are sent to the lower layers, modeling the hardware. A good example of a hardware macro would be, e.g., "turn left 30 degrees". A task is also notified whenever an event of interest is detected by the sensors. When this happens, the hardware completely freezes its activity and notifies the robot application layer of the occurred event (e.g., a touch sensor identified an obstacle). A task may decide to continue the interrupted command, or abort it and continue with a new sequence task.

Although the task model allows robot autonomy, there are situations in which a robot must be controlled by a human. Imagine, for example, that a robot reaches a dead end and is not capable of autonomously leaving that space. The *direct mode* layer is basically an interface that allows

direct connection to the robot hardware. The *overriding layer* is a way to override an existing task without using the direct mode.

Both the inter-operation layer and the robot application layer are implemented as Java programs running on a H3870 iPAQ PDA.

The third layer contains software models and macros for operating all operative parts of the robot, e.g., motors and sensors. This layer offers a homogeneous view of the underlying hardware, and it is implemented using the LeJOS [Jos02] operating system running on LEGO's RCX device controller. LeJOS is a tiny Java VM operating system, with a footprint of less than 20 kBytes. The hardware entities have been encapsulated in a Device class with Sensor and Motor as sub-classes. For each particular device (e.g., light sensor, motion sensor) further sub-classes are added to the system.

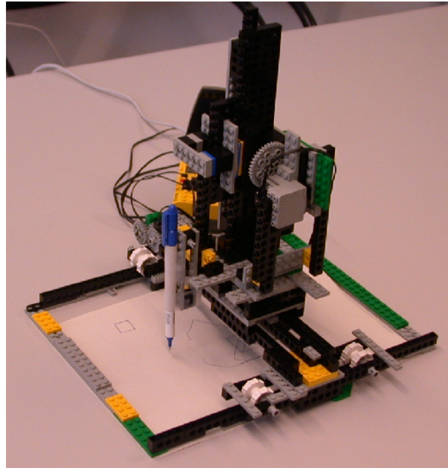

Fig. 4. A plotter prototype integrated in the proactive platform.

4.2 Application

4.3 Target scenario

Of the several prototypes developed using this architecture, we will describe here the plotter of Figure 4. This robot acts as the head of a printer as it moves a marking pen across three dimensions. The same robot can be used to control any other device in a similar manner (a saw, a scalpel, a drill, an electric contact, etc). Movement across each dimension is controlled by a motor. The overall movement is determined by a drawing program that exports a drawing interface as a Jini service. The program and the robot do not contain any code beyond that related to drawing.

4.4 Adaptation example

An first example of adaptation is a hardware monitoring and logging extension. The idea is to record every movement of the robot and to store these movements persistently. Figure 3.b shows the situation after the robot has received an extension that monitors hardware activities. For each one of the motors, any calls to their proxy objects will be intercepted by the added extension (the gray box). For each method invocation of the motor proxy, the extension logs the time when the command was issued, its duration, as well as the identity of the robot (1). This data is first locally stored and then asynchronously sent to a base station (2). At the base station, the data is stored in a database (3).

The extension for hardware monitoring and logging is very concise. A simplified version of it is depicted in Figure 5. It is a 100% Java class compiled and instantiated on the base station. Line 6 specifies that entries and exits of *any* methods belonging to a Motor class must be intercepted. The REST parameter indicates that the signature of the method (specific arguments) are not important. Once intercepted, PROSE will call ANYMETHOD (lines 6-9). This method does the actual logging by calling the ownerProxy object.

```
1 class HwMonitoring extends Aspect {
2   // the remote owner
3   RemoteOwner ownerProxy;
4
5   // the interception specification
6   public void ANYMETHOD(Motor thisMotor, REST params)
7       {
8           ownerProxy.post(thisMotor.getId(),System.currentTimeMillis(),...)
9       }
10 }
```

Fig. 5. An extension for remote monitoring.

It is important to notice that neither the robot nor the program controlling the robot is aware of the extension. The extension can be added or removed as needed. If the robot is moved to a different location, that location can add a new extension that indicates where the data must be sent for persistent storage. Or, within the same location, the extension can be exchanged for a new one that indicates that the data must be sent to a program that shows the movements in a graphic display. Similar extensions could be used, for instance, to disable certain movements of the robot, or certain combination of movements, to replay sequences of movements, etc. A clear advantage of this form of adaptability is that devices only need to carry their basic functionality. Anything else are location specific adaptations inserted or extracted as needed.

4.5 Applications of the adaptation example

The simple monitoring and logging extension described can be used in many forms. We have developed several such applications by making the base station itself available as a Jini service. One can, thus, connect to the base station and query the database that stores all movements performed by robots being monitored by the base station. Figure 6 is a screen-shot of a client application connected to the base station. On the left side, it displays a list of all the motor actions ever executed by the robot named robot:1:1. Out of the action list, a selection was transferred to the right panel. The right panel allows manipulations of these movement sequences. Some examples of useful manipulations are:

Remote replication If the robot is being controlled by a human, it is possible to use the extension to monitor all the moves and feed them to an identical robot in a remote location (or to a collection of identical robots in other locations). That way one can either duplicate the work or follow up what is being done. It is also possible that the replication of the work takes place at a scale different from what is being done by the original robot. The only thing needed is to amplify or reduce the extracted sequence of movements to adjust it to the new scale.

Simulation In difficult or important situations, one may want to record all movements performed. That way, if an accident or failure occurs, one can replay a part of the sequence of movements to see if the failure can be reproduced or better understood. This feature is particularly interesting if the failure is due to the interaction between different robots: the system can be instructed to replay the sequence of movements of all robots at the right relative time, thereby reproducing the interaction between them.

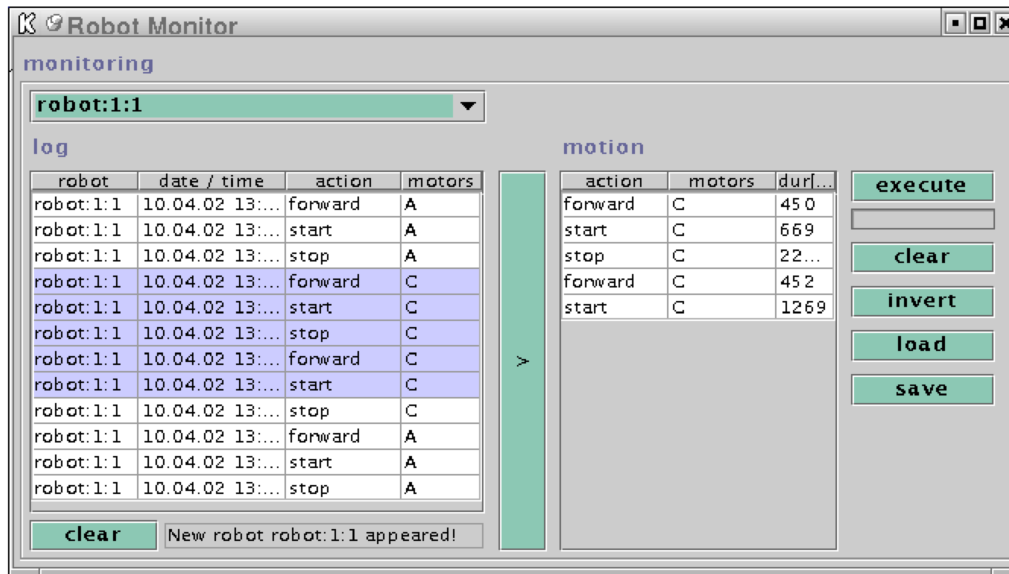

Fig. 6. A screen-shot of a simple hardware monitoring tool.

Control It is possible that when the robot is used in certain locations, one might want to limit what the robot can do. For instance, one may forbid movements beyond certain coordinates so that certain parts of the paper remain untouched. If a drill is used, one may prevent lateral displacement of the drill when it is brought down. For this, the monitoring extension only needs to incorporate the coordinates or sequence of movements that are not allowed and simply check before allowing the movement to take place.

Again, the relevant point in all these potential applications is that none of them require to have any code in the robot. Depending when and how the robot is used, adaptations are added and removed as needed. Moreover, new adaptations can be easily designed as the use of the robots changes or evolves over time. This greatly simplifies the design of the software for the robot itself but also makes the maintenance of the adaptations much more manageable (compared with the case where they are embedded in every robot). This feature makes the approach highly attractive in many industrial settings.

4.6 Discussion

Our experience with the platform for proactive adaptation has been extremely encouraging. One of our initial goals was that the programming of extensions should be easy to use by application programmers. We have had this expectation confirmed during the past year, as students involved in projects and courses had to use MIDAS for exercises and development projects. Indeed, if a student was proficient in Java, a few days sufficed for the student to be able to program extensions.

This user experience lead to new applications we did not consider in the beginning. One example is a security extension that intercepts readings of all sensors of the robots. The security aspect intercepts all service calls and decides, *before* the execution of the application logic, whether the remote caller has the right to execute the intercepted method. If the access is denied, the execution is ended with an exception. Another example are applications where the "age" of the device corresponds to the trust associated to that device. A proactive context can add an extension that records the "birth date" of a device. The very same extension may intercept all service invocations of all possible devices and decide how to proceed depending on the device's age. We are at this stage considering other alternatives also suited for ubiquitous computing environments [KZ01].

One important issue is the cost of having a platform for run-time adaptation activated in each node. When no extensions are added, an overhead of about 7% (measured using a SPECjvm benchmark [SPE]) could be observed. When adding a do-nothing extension that traps method entries, all methods not affected by interceptions are not slowed down. For those methods where interceptions are performed, an overhead of roughly 900ns can be expected. For comparison, a void non-intercepted interface call costs 700ns on a Pentium 2, 500 MHz CPU. We measured the overhead of extensions implementing security, transactions and orthogonal persistence. In all cases the cost of the interceptions was much less then the cost of executing the additional functionality, indicating that the platform overhead is negligible. The results of these measurements are described in [PAG02].

Future work MIDAS heavily relies on the Jini infrastructure. As Jini is required on all participating nodes, a resource-scarce device would need a full Java runtime environment. To reduce this resource consumption, some parts of MIDAS are being re-implemented to obtain a smaller footprint. Further we are looking at tuple spaces [Gel85,LCX+01] to get a more flexible and expressive platform for distributing extensions.

5 Conclusion

In this paper we have presented a generic platform for proactive middleware. The platform supports the adaptation at run time of applications by extending their functionality with new code that enhances, modifies or controls the functionality already present in the application. We have described the architecture in detail and shown with an example how it can be effectively used in different industrial settings. We are aware that the type of proactiveness we propose is not suitable to all form of mobile computing. Nevertheless, our experience in developing several prototypes using the proposed platform show that the technology works quite well in nomadic settings, where high-end mobile nodes like PDAs, laptops or robots interact using wireless networks. Such devices cover already a wide range of mobile computing applications. The prototypes built also demonstrate that our approach can help to significantly reduce some of the cost associated to the maintenance of widely distributed systems and simplify the process of developing software capable of working in mobile computing scenarios.

References

[APW01] D. Arregui, F. Pacull, and J. Willamowski. Rule-Based Transactional Object Migration over a Reflective Middleware. In *Middleware 2001: IFIP/ACM Intl. Conf. on Distributed Systems Platforms*, volume 2218 of *LNCS*, pages 179–196, 2001.

[AWO+99] K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Waldo. *The Jini Specification.* Addison-Wesley, Reading, MA, USA, 1999.

[BC01] G. S. Blair and G. Coulson. The Design and Implementation of Open ORB version 2. *IEEE Distributed Systems Online Journal*, 2(6), 2001.

[BCKP95] R. Bagrodia, W. Chu, L. Kleinrock, and G. Popek. Vision, Issues, and Architecture for Nomadic Computing. *IEEE Personal Communications*, 2(6):14–27, 1995.

[BH02] J. Baker and W. Hsieh. Runtime Aspect Weaving Through Metaprogramming. In *1st Intl. Conf. on Aspect-Oriented Software Development, Enschede, The Netherlands*, pages 86–95, April 2002.

[CBCP01] M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. In *Middleware 2001: IFIP/ACM Intl. Conf. on Distributed Systems Platforms*, volume 2218 of *LNCS*, pages 160–178, 2001.

[CCM97] CORBA Component Model RFP. Available at http://www.omg.org/docs/orbos/97-05-22.pdf, 1997.

[CHS01] W.-K. Chen, M. Hiltunen, and R. Schlichting. Constructing adaptive software in distributed systems. In *Proc. of the 21st Intl. Conf. on Distributed Computing Systems (ICDCS-01)*, pages 635–643, Los Alamitos, CA, April 16–19 2001. IEEE Computer Society.

[ECDF01] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. An Architecture for the Effective Support of Adaptive Context-Aware Applications. *LNCS*, 1987, 2001.

[EGK+99] F. Eliassen, V. Goebel, T. Kristensen, T. Plagemann, A. Andersen, H. Rafaelsen, W. Yu, G. Blair, F. Costa, G. Coulson, and K. Saikoski. Next generation middleware: Requirements, architecture, and prototypes. In *The Seventh IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 60–, Tunisia, South Africa, December 1999.

[Gel85] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

[HSH+99] M. A. Hiltunen, R. D. Schlichting, X. Han, M. M. Cardozo, and R. Das. Real-time dependable channels: customizing qoS attributes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):600–612, June 1999.

[Jos02] Jose Solorzano. leJOS, Java for the RCX. www.lejos.org, 2002.

[KF01] E. Kiciman and A. Fox. Separation of Concerns in Networked Service Composition. Position Paper Workshop on Advanced Separation of Concerns in Software Engineering at ICSE 2001, Toronto, Canada, May 2001.

[KLM+97] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *In Proc. of ECOOP'97 Jyväskylä, Finland*, volume 1241 of *LNCS*, pages 220–242. Springer-Verlag, New York, NY, June 1997.

[KZ01] Tim Kindberg and Kan Zhang. Context authentication using constrained channels. Technical Report HPL-2001-84, HP Labs, 2001.

[LCX+01] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V . Vasudevan, S. Landis, P. Davis, Bruce K., and P. Bowman. Hitting the distributed computing sweet spot with TSpaces. *Computer Networks (Amsterdam, Netherlands: 1999)*, 35(4):457–472, March 2001.

[Leg] Lego. Lego mindstorms robotics invention system. At http://mindstorms.lego.com, 2002. Lego Mindstorms.

[LK98] C. V. Lopes and G. Kiczales. Recent Developments in AspectJ. In Serge Demeyer and Jan Bosch, editors, *Object-Oriented Technology: ECOOP'98 Workshop Reader*, volume 1543 of *LNCS*, pages 398–401. Springer, 1998.

[NSN+97] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-Aware Adaptation for Mobility. In *Sixteenth ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, 1997.

[OL01] D. Orleans and K. Lieberherr. DJ: Dynamic Adaptive Programming in Java. In *Reflection 2001: Meta-level Architectures and Separation of Crosscutting Concerns*, Kyoto, Japan, September 2001. Springer Verlag.

[PA02] A. Popovici and G. Alonso. Ad-Hoc Transactions for Mobile Sevices. In *Proc. of the 3rd VLDB Intl. Workshop on Transactions and Electronic Services (TES '02)*, Hong Kong, China, August 2002.

[PAG02] A. Popovici, G. Alonso, and T. Gross. Design and evaluation of spontaneous container services. Technical report no. 368, Computer Science Department, Swiss Federal Institute of Technology, 2002.

[PAG03] A. Popovici, G. Alonso, and T. Gross. Just in time aspects: Efficient dynamic weaving for java. In *2nd Intl. Conf. on Aspect-Oriented Software Development, Boston, USA*, 2003.

[PGA02] A. Popovici, T. Gross, and G. Alonso. Dynamic Weaving for Aspect Oriented Programming. In *1st Intl. Conf. on Aspect-Oriented Software Development, Enschede, The Netherlands*, April 2002.

[PLF+01] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. *LNCS*, 2201, 2001.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[SGGB99] E.G. Sirer, R. Grimm, A.J. Gregory, and B.N. Bershad. Design and Implementation of a Distributed Virtual Machine for Networked Computers. In *Symposium on Operating Systems Principles*, pages 202–216, 1999.

[SM] Sun Microsystems. The Java Security Model. http://java.sun.com/.

[SOT+00] T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, and T. Nakatani. Overview of the IBM Java Just-in-Time Compiler. *IBM Systems Journal*, 39(1):175–193, 2000.

[SPE] Spec - The Standard Performance Evaluation Corporation. SPECjvm. Web access http://www.spec.org/osg/jvm98/.

[Sun02] Sun Microsystems. *Java 2 Platform, Standard Edition, v 1.4.0: API Specification*, 2002. `java.sun.com/j2se/1.4/docs/api/`.

[XC02] Xerox Corporation. The AspectJ Programming Guide. Online Documentation, 2002. http://www.aspectj.org/.

[YK01] S. S. Yau and F. Karim. Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad-Hoc Network Environments. In *5th International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 319–326, March 2001.

[ZBS97] John A. Zinky, David E. Bakken, and Richard E. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1), 1997.