A Pro-Active Middleware Platform for Mobile Environments

Denis Conan, Chantal Taconet, Dhouha Ayed, Lydialle Chateigner, Nabil Kouici, and Guy Bernard
GET / INT, CNRS Samovar
9 rue Charles Fourier, 91011 Évry, France
{Denis.Conan,Chantal.Taconet,Dhouha.Ayed,Lydialle.Chateigner,Nabil.Kouici,Guy.Bernard}@int-evry.fr

ABSTRACT
With wireless communications and mobile hand-held devices becoming a reality, new applications where users can have access to information anytime, anywhere are made possible. To design a middleware-based platform for applications such as emergency aid, crisis managements, our approach is threefold. We design and implement a middleware platform that hides as much as possible the details of the hardware, the operating system, and the telecommunication protocols from application developers and users. Then, we define a context manager associated with a middleware manager that cope with the collaboration between the users and the other middleware services. Next, users and the other middleware services rely on context information: the former for expressing needs and behaviour, the latter for being pro-active. These middleware services include disconnection management, fault management, deployment.

KEY WORDS
Proactiveness, mobile environment, context-awareness, disconnection management, fault tolerance, deployment.

1 Introduction

Since the early 90's, the field of mobile computing has witnessed tremendous research and technological advances. With wireless communications and mobile handheld or wearable devices becoming a reality, new applications where users can have access to information anytime, anywhere are made possible. In the future IT society, mobility will be the rule and no longer the exception. The emergence of the new field of pervasive computing as a successor to both distributed systems and mobile computing enforces this vision: environments will be "*saturated with computing and communication capability, yet gracefully integrated with human users*" [1]. Right now, mobile handheld devices such as Personal Digital Assistants (PDA) are attracting great attention. However, software capacities of mobile handheld devices are rather modest, the connectivity to services via the Internet is limited and the number of available services is very low compared to the possibilities of home and personal computers connected to the Internet via wired links.

The AMPROS project [2] is motivated by the enabling and enhancement of wireless communication. The objectives of the project include research, development, and prototyping of a middleware-based platform in wireless networks with the generalisations that can be used for services such as, emergency aid, crisis managements. The platform must address the challenges of interoperability, scalability, dynamics, and mobility in communicating environments.

The adaptation to the characteristics of mobile computing can be performed by the application (*laissez-faire* strategy), by the system (*transparent* strategy), or by both the application and the system (*collaboration* strategy)[3]. As surveyed in [4], there is much work dealing with mobile information access that demonstrates that the *laissez-faire* and the transparent approaches are not adequate. Our collaboration approach is then threefold. We design and implement a middleware platform that hides as much as possible the details of the hardware, the operating system, and the telecommunication protocols from application developers and users. Then, we define a context manager associated with a middleware manager that cope with the collaboration between the users and the other entities of the middleware. Next, users and middleware services rely on context information: the former for expressing needs and behaviour, the latter for being proactive. These middleware services include disconnection management, fault management, deployment.

The structure of the paper is the following. Section 2 gives the rationale for the different services making up the platform. Then, Section 3 presents the big picture of the architecture of the platform. The following sections succintly sketch the detailed architecture of the main services: Section 4, 5, 6, and 7 for context management, disconnection management, fault management, and deployment, respectively. Finally, Section 8 compares our approach with related research work, and finally, conclusions and future work are drawn in Section 9.

2 Motivations and objectives

The AMPROS platform may be dedicated to emergency applications with a great number of participants (rescue workers, patients...) which need to coordinate themselves. AMPROS applications are supported by multi-network technologies and especially by tetra network. It must allows proactiveness of both the platform and the

applications. The extra-functional requirements elicited in the AMPROS project include context awareness for allowing the platform to be aware of its environment in order to achieve a better proactivity ; disconnection management for allowing to continue the execution of distributed applications even in case of disconnection ; fault tolerance and group management for enabling distributed applications to behave in a well defined manner once fault occurs while collaboration between dynamic groups of persons occurs ; deployment of multi-component applications for minimizing the software necessary on each terminal and allowing adaptation of the software to the current context.

In proactive computing, the system adapts itself without any user interaction. This is enabled because the system collects information about its execution context. Context awareness is enabled by different mechanisms which allow to collect information and to react to context changes. Context awareness includes several level of awareness: Network awareness (*e.g.*, bandwidth availability), resource awareness (*e.g.*, terminal capabilities), environment awareness (*e.g.*, geographical location), and finally user-related awareness (*e.g.*, current users activity). All these different kinds of contexts may be relevant to the system, to the middleware, or to the application. Context awareness allows to modify their respective behaviour.

Mobile applications must keep working even while being weakly connected or disconnected. Weak connectivity means intermittent communication, low-bandwidth or high-latency [5]. We distinguish between two kinds of disconnections: Voluntary disconnections when the user decides to work on their own to save battery life or communication costs, or when radio transmissions are prohibited (as aboard a plane); and involuntary disconnections due to physical wireless communication breakdowns such as in an uncovered area or when the user has moved out of the reach of a base station. However, the need to continue to work in a mobile environment raises the problem of data availability in the presence of disconnections. Thus, to offer this continuity, the application and the system should be reactive to mobile environment changes. Service continuity while being disconnected implies the creation of disconnected components before loosing connectivity.

Distributed applications must be able to behave in a well-defined manner once faults occur. A complete fault tolerance solution is not drawn in the following solution, but only some of the fundamental paradigms such as connection and failure detection. The distributed application relies on two middleware primitives to reach a common decision, which depends on the initial inputs proposed by the distributed entities, despite crash failures.

Another part of proactiveness is taken into account by the discovery of available resources. We call resources all the entities that may be discovered. This includes users (they may be seen by the system as resources), computers, software components at different stages (software component packages that may be discovered in order to be downloaded, or instantiated software components that are resources which offer services). An application is made up of several distributed components and there are several kinds of terminals used. The applications software is installed just in time on the terminals (when it is necessary). This feature is important to achieve proactivity since it allows to install software suited to the execution context.

3 Architecture

Figure 1 presents a layered architecture of our platform. The higher level components represent application-specific components. The left lower-level components are the entities already provided by the operating system and the available middleware. In the area surrounded by a dotted line, are depicted the middleware entities specific to our platform. In the rest of the paper, Section 4 details resource monitoring, context management, monitoring/simultation, middleware management and multi-network managment. Then, Section 6 presents disconnection management, and logging and reconciliation. Section 6 details failure detection and group management. Finally, Section 7 develops proactive trading and deployment management.

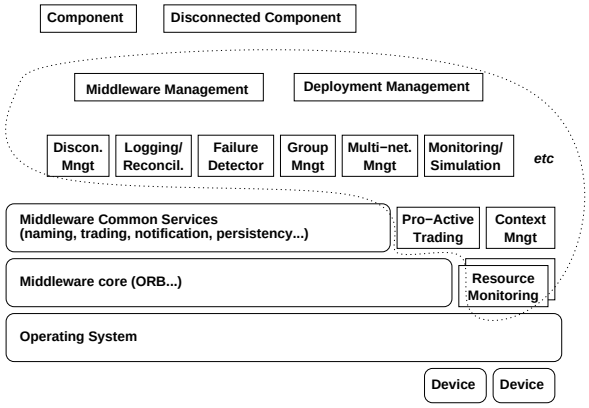


Figure 1. Acrhitecture of the plaform.

4 Context management

The context management service is a rather generic service, which is required to support other middleware services. The behaviour of applications or middleware services is influenced by a number of *resources* (*e.g.*, battery state, network bandwidth, geographical location). We name *context* a valuation of this set of resources. The context management service is structured in a layered

manner, as in [6]. Now, we describe the role of each layer from bottom to top (see Figure 2).

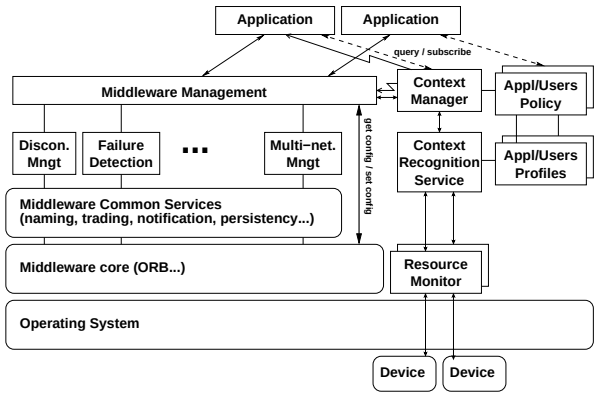


Figure 2. Context management architecture.

4.1 Resource monitoring

There is one *Resource Monitor* (RM) for each type of resource. A RM is responsible for getting the current value of the associated resource. The value is obtained from a "driver" (in the broad sense: *e.g.*, a network driver for monitoring the current bandwidth, or a pseudo-driver for getting the available memory or the CPU utilisation). RMs can include some system- or device-dependent code. They get raw data and process them, if needed, in order to provide system-independent low-level context data (*e.g.*, the average bandwidth value during the last minute). Periodically, RMs push low-level context data to the *Context Recognition Service*, for instance `Device:Network:WiFi:Bandwidth = 30`, or `Location:Coordinates =` {`80.0, 140.3`}. Low-level context data are hierchically organised, as in [6].

4.2 Context recognition service

The *Context Recognition Service* (CRS) gathers low-level context data from RMs and provides the appropriate high-level context data, as considered by applications/users profiles, to the context manager. For instance, a profile containing the declarations `Device:Network:*:Bandwidth` and `Location:Coordinates` denotes the interest of the application (or the user) in monitoring bandwidth and location.

4.3 Context manager

Policies describe the interest of individual applications, or users, or else the middleware itself, to be notified whenever significant changes in the context state occur, for instance `if (Device:Network:*:Bandwidth < 10 && Device:Processor > 25) then ''enable compression''`. It is the responsibility of the *Context Manager* (CM) to notify the appropriate entity (application, user, or middleware) when significant changes occur: *e.g.* `notify ''middleware'', ''enable compression''`, or `notify ''user'', ''battery low''`.

5 Disconnection management

In this section, we describe the role of the architecture of the disconnection management, and the logging and reconciliation (see Figure 3). Disconnection management is based on the Domint prototype [7, 8], and logging and reconciliation uses transformation functions [9].

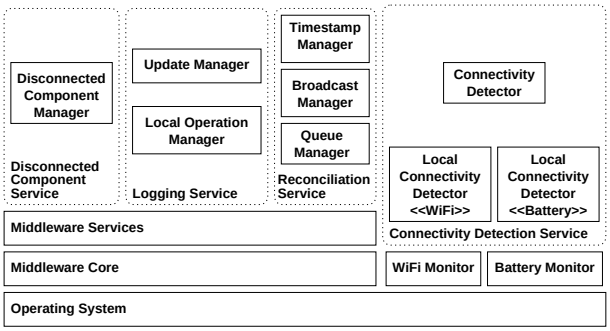


Figure 3. Disconnection management service.

5.1 Connectivity management

The *Connectivity Detector* (CD) monitors the availability level of local resources: *e.g.*, bandwidth, memory, battery, by creating a *Local Connectivity Detector* (LCD) per local resource. LCDs use data provided by the context recognition service for connectivity management. Each LCD uses an hysteresis mechanism for smoothing variations in resource availability to determine the connectivity mode: Connected, partially connected, and disconnected. The role of CD is to gather LCDs data and to provide a "global" conectivity view.

The *Disconnected Components Manager* (DCM) centralises the control of all the *Disconnected Components* (DC) in the cache of the mobile terminal. The cache is shared between all the applications running on the mobile terminal so as to avoid having several copies of the same DC. DCs have the same structure as the remote components: Data and code similar to the remote components ones. When DCs are instantiated in the mobile terminal, their states equal the remote ones. In the connected mode, requests are sent directly to remote components. In the partially connected mode, requests are sent

to disconnected components that forward them to remote components. In the disconnected mode, requests are sent solely to disconnected components.

5.2 Logging and reconciliation

The logging and reconciliation is based on SAMS [10] and transformation functions [11] and adapted for the component-oriented systems.

The logging management service is responsible for managing operations executed when disconnected. This service comprises the *Local Operations Manager* (LOM) and the *Update Manager* (UM). While being disconnected or partially connected, operations called on remote components are redirected to local DCs and logged in LOM. While becoming connected, UM is responsible for flushing the log, and for updating the state of DCs with calls performed by the corresponding remote component while being disconnected.

Each terminal owns a reconciliation service in order to maintain the coherence between DCs and their corresponding component. This service comprises three entities. The *Broadcast Manager* (BM) manages all the communications between servers and clients. The *Timestamp Manager* (TM) is a sequencer giving tickets in order to serialise calls. The *Queue Manager* (QM) stores all the operations of all users in different queues.

6 Fault Management

The distributed system is asynchronous: There is no bound on message delay, clock drift, or the time to execute a step. Hosts can fail by crashing —*i.e.*, by prematurely halting. The fault tolerance is masking [12], stating that users do not experiment the failure, except the ones using the physical display of the failed hosts. Figure 4 depicts the architecture of the fault management.

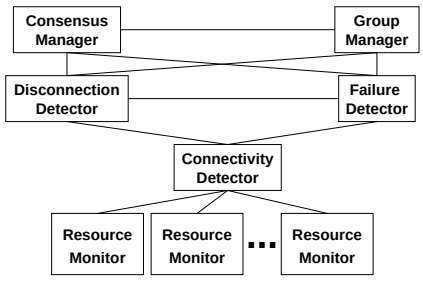


Figure 4. Fault management service

In case of crash failures of some nodes, the *Failure Detector* (FD) transparently detects theses failures in order to activate a (rollback or roll-forward) recovery of the application. FD implements an unreliable failure detector of the class $\diamond\mathcal{S}$ with strong completness and weal accuracy [13]. FD is built on top of a connectivity detector in order to leverage the failure detection by not sending heartbeat messages to disconnected hosts.

On top of the connectivity detector, the *Disconnection Manager* DM provides a global view of disconnections by implementing a distributed algorithm fullfiling the disconnection properties [14] of strong completness —*i.e.*, eventually every process that disconnects is permanently seen disconnected by every connected process— and strong accuracy —*i.e.*, no process is seen disconnected before being really disconnected.

The *Group Manager* (GM) is able to provide the list of co-operating members of the same application, and the hosts where the co-operating members reside. Then, the members of the group can work collaboratively, for instance via consensus. The GM subscribes and unsubscribes group members to FD. GM is responsible for determining coherent views of the set of members (including faulty and disconnected hosts) that participate to the distributed application.

The *Consensus Manager* ()CM tolerates remote hosts failures and disconnections by excluding faulty hosts and disconnected hosts from the current consensus. Disconnected entities can reintegrate only after a reconciliation.

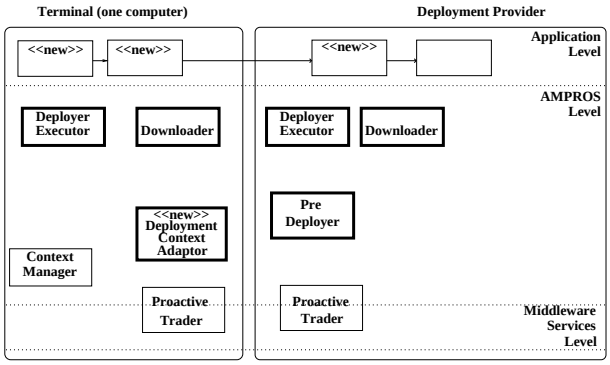
7 Deployment Management

Software deployment is the complex process that covers all the activities performed after the development of a software system. These activities include configuring, releasing, installing, updating, reconfiguring and even deinstalling a software system [15].

Context awareness plays a significant role in the deployment. It permits to automatically install and reconfigure a software system on the consumer site depending on users needs and preferences, and environmental constraints.

Let us take the case of two users, one using a laptop and another one using a PDA who want to deploy the same application. The context-aware deployment solution installs for the former user a version comprising a normal graphical interface and for the latter one a version with a small graphical interface appropriate to the screen size of the PDA. In addition, if the resources of the PDA are too scarce to install the whole software, only some parts of the applications are installed on the PDA, and the others are installed on other servers belonging to the producer site. The whole procedure is transparent to the user and performed automatically.

In this Section, we propose a solution for making context-aware deployment of multi-component applications whose components may be distributed on different machines [16]. The architecture is depicted in Figure 5.


Terminal (one computer)
Deployment Provider
Application Level
AMPROS Level
Middleware Services Level
<<new>>
<<new>>
<<new>>
Deployer Executor
Downloader
Deployer Executor
Downloader
<<new>> Deployment Context Adaptor
Pre Deployer
Context Manager
Proactive Trader
Proactive Trader

This is an important feature since disconnections are no longer the exception but the rule, and then must be dealt with.

Deployment management is included in our architecture. Indeed, deployment of an application is the first step in which context awareness may be taken into account [23], especially if the deployment is made just in time —*i.e.* when a user starts the application. We think that it is important to disminish the work of application developers for deploying their applications. We argue that some context reconfigurations may be driven by the middleware itself without many application-specific description. For instance, this is mostly the case for disconnection management and multi-network management.

Finally, the discovery service has an important place in our architecture. Since the discovery servic allows to discover available services as soon as they are available, it is one of the means to react to mobility changes. Furthermore, it permits to find the most suitable resources according to a given context.

9 Conclusion

In this paper, we presented a proactive middleware platform dedicated to emergency applications running in the context of mobile environments. We introduced the basic blocks of the architecture. Since most of the extra-functional services present in the platform cannot be transparent, the first and more innovative entity is the context manager which is central for the collaboration between the applications/users and the platform. Examples of contexts may include but are not limited to network information (*e.g.*, bandwidth availability), resource information (*e.g.*, terminal capabilities), environment information (*e.g.*, geographical location), and user-related information (*e.g.*, current users activity). Contexts are provided for the managemenent of the connectivity and the deployment. The configuration and the reconfiguration are controlled by a middleware manager. The user is able to keep working while being disconnected and can launch applications not already installed on the terminal.

We currently have first prototypes of the disconnection management and the deployment management. Specific algorithms for failure detection and consensus tolerating disconnections have been designed. Future work is then to continue the integration of these blocks through the context and middleware managers.

References

[1] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, August 2001.

[2] AMPROS Home Page. `http://www-inf.int-evry/AMPROS`, 2003.

[3] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proc 15th Symposium on Principles of Distributed Computing*, pages 1–7, 1996.

[4] J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), 1999.

[5] L.B. Mummert. *Exploiting Weak Connectivity in a Distributed File System*. PhD thesis, Carnegie Mellon, USA, September 1996.

[6] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E.-J. Malm. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, july-september 2003.

[7] D. Conan, S. Chabridon, O. Villin, G. Bernard, A. Kotchanov, and T. Saridakis. Handling Network Roaming and Long Disconnections at Middleware Level. In *Proc. Workshop on Software Infrastructures for Component-Based applications on Consumer Devices*, Lausanne, Switzerland, September 2002.

[8] N. Kouici, D. Conan, and G. Bernard. Disconnected Metadata for Distributed Applications In Mobile Environments. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, USA, June 2003.

[9] L. Chateigner, S. Chabridon, and G. Bernard. Intergiciel pour l'informatique nomade: rplication optimiste et rconciliation. In *MAnifestation des JEunes Chercheurs en STIC, MAJECSTIC*, Marseille (France), October 2003.

[10] P. Molli, H. Skaf-Molli, G. Oster, and S. Jourdain. SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments. In *17th International Conference on Computer Supported Cooperative Work in Design*, Rio de Janeiro (Brazil), September 2002.

[11] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Development of Transformation Functions Assisted by a Theorem Prover. In *CSCW*, New Orlns (USA), November 2002.

[12] F.C. Gärtner. Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments. 31(1):1–26, March 1999.

[13] Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. 43(2), March 1996.

[14] L. Temal. Disconnection Management and Failure Detection. Master's thesis, University of Versailles Saint-Quentin-En-Yveline, Institut National des Tlcommunications, vry (France), September 2003.

[15] R.S. Hall, D.M. Heimbeigner, A. van der Hoek, and A.L. Wolf. An architecture for Post-Development Configuration Management in a Wide-Area Network. *The International Conference on Distributed Computing Systems*, May 1997.

[16] D. Ayed, C. Taconet, and G. Bernard. Context-Aware Deployment of multi-component applications. In *Pro. 5th Generative Programming and Component Engineering (GPCE03) Young Researchers Workshop*, September.

[17] S.S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gu pta. Reconfigurable Context-Sensitive Middleware for Pervas ive Computing. *IEEE Pervasive Computing, joint special issue with IEEE Personal Communications*, 1(3), July-September 2002.

[18] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, November 2003.

[19] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), 1996.

[20] B. D. Noble and M. Satyanarayanan. Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications*, 4(4):245–254, 1999.

[21] A.D. Joseph, J.A. Tauber, and M.F. Kaashoek. Mobile computing with the Rover toolkit. *ACM Transactions on Computers*, 46(3), 1997.

[22] D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, and C.H. Hauser. Managing Update Conflicts in Bayou: A Weakly connected Replicated Storage System. *Proc 15th Symposium on Operating Systems Principles*, 1995.

[23] C. Taconet, E. Putrycz, and G. Bernard. Context Aware Deployment for Mobile Users. In *Proceedings of the 27th International Conference on Computer Software and Applications Conference (COMPSAC 2003)*, Dallas, Texas, November 2003.