Improving Mobile Computing Performance by Using an Adaptive Distribution Framework

F. Le Mouël, M.T. Segarra, and F. André

IRISA Research Institute
Campus de Beaulieu
35042 Rennes Cedex, FRANCE
{flemouel,segarra,fandre}@irisa.fr

Abstract Portable devices using wireless links have recently gained an increasing interest. These environments are extremely variable and resource-poor. Executing existing applications in such environments leads to a performance degradation that makes them unsuitable for this kind of environments. In this paper, we propose the utilization of AeDEn, a system that performs a global distribution of applications code and data in order to take benefit from the underutilized resources of the mobile environment. This is achieved by designing AeDEn as part of the MolèNE system, a more general framework that we have built to encapsulate a set of generic and adaptive services that manage wireless issues.

1 Introduction

Portable devices (laptops, PDAs, mobile phones, etc) using wireless links (GSM, satellites) have recently gained an increasing interest. Simultaneously, many applications are designed requiring more and more resources (memory, processor, etc) and assuming an environment that ensures the availability of all of them. Executing such applications on a wireless environment leads to a performance degradation for three main reasons: 1) portable devices are generally battery-limited, resource-poor entities compared to their counterparts in the fixed network, 2) wireless networks are subject to important variations and suffer from frequent disconnections due to the interaction with the environment, and 3) when moving, stations (mobile or not), devices (scanners, printers) and functionalities (proxy cache) that can be accessed by a portable computer change. Existing applications are not designed to tolerate such variations and their presence generally leads to applications finishing abruptly.

Approaches that deal with the performance degradation problem propose the transfer of resource-consuming tasks to a fixed station [4, 5]. However, all these solutions lack from providing a systematic approach to exploit unused or under-utilized external resources. In this paper, we propose the AeDEn system (Adaptive Distribution Environment) [1] as providing the necessary means to effectively distribute applications code and data in a rapidly varying environment. Changes in the environment are taken into account at two very different levels: the application and the distribution system itself. The first one concerns the dynamic layout of the application onto a set of stations and the second one allows to dynamically change the distribution algorithm used to decide about the layout.

In order to ensure the generic nature and the adaptability of AeDEn, we have designed it as part of a more general system called MolèNE (Mobile Networking Environment) [10]. We have designed MolèNE as a system providing a set of generic functionalities that abstract concepts usually utilized by mobile applications. Moreover, MolèNE provides the necessary mechanisms to track environmental changes and to modify the behavior depending on these changes and applications needs. All these mechanisms are used by AeDEn making it a framework into which applications can insert their own distribution preferences.

The paper is organized as follows. Available mechanisms for the construction of adaptive services in MolèNE are presented in Section 2. Adaptive and dynamic distribution schemes of AeDEn are introduced in Section 3. Section 4 compares our systems to existing approaches. Finally, Section 5 concludes and gives future research directions.

2 Adaptation in MolèNE

Existing approaches to adapt applications behavior to a mobile environment are based on a set of common concepts which implementation is usually application-dependent. We have identified these concepts and their abstraction constitutes the foundation of MolèNE. Application-independent concepts, called *tools* are implemented as concrete object-oriented classes. Other functionalities are application-dependent and have to be customized to each application. Such functionalities, that we call *services*, are provided as an object-oriented framework. Some of its classes remain abstract allowing developers to customize services to different applications [10]. This is usually a simple task and allows the framework to perform other tasks much more complex.

Static and dynamic characteristics of mobile environments vary significantly over time making it necessary to adapt services behavior to the execution conditions. On one hand, many portable computers exist from laptops to palmtops that make different tradeoffs between resources availability and portability. On the other hand, different wireless networks can be used by a portable computer that provide a varying transmission quality.

Dynamically adapting services behavior to these characteristics allows to offer the user a better QoS (e.g. response time). Therefore, we have considered the adaptation as a necessary property and we have built the framework as a set of adaptive services managing issues related to the mobile environment on which applications execute. Three mechanisms are available in MolèNE to perform the dynamic adaptation of the services:

- a Detection & Notification Service that detects hardware and software *changes* and notify the interested applications;
- an adaptation interface allowing applications to describe their adaptation strategy: which are the significant changes and which are the *reactions* to be

performed when the changes occur. Therefore, application-dependent adaptations can be performed by MolèNE;
- a *reactive system* that takes into account the adaptation strategy provided by the application and uses the Detection & Notification Service to ensure the execution of the strategy.

2.1 Services Design

Services have been designed so that: 1) they can be customized to different applications, 2) they are extensible in the sense that they are able to consider application-provided strategies, and 3) they ensure the flexibility of the framework since different strategies can be considered for a service.

Functionalities of a service are implemented in MolèNE by *components*. A component is made up of a set of objects and constitutes the smallest software unit in MolèNE. For example, a cache management service provides two functionalities: the *data replacement* that is implemented by a MolèNE component deciding which data to delete from the cache and the *cache loading* that is implemented by a component that decides the data to be loaded into the mobile computer cache.

Figure 1 shows the architecture of a MolèNE component. The *Interaction* object receives functional requests from other components by means of events (synchronous or asynchronous) containing the necessary information to manage the request. The *Implementation* object provides the functionality of the component. Finally, applications can build an adaptive component by adding a *Controller* object to the component architecture. This object is the main element of the reactive system: it ensures the execution of the adaptation strategy for the component by using the Detection & Notification Service to be informed of changes on the environment and the *Adaptation Strategy* to know the reactions to be carried out.

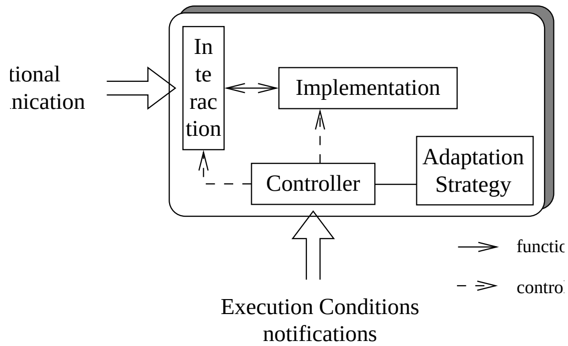

Figure 1. MolèNE component architecture

Designing services in this way 1) facilitates their construction since a well-defined set of issues is identified that ease the choices of developers and limit redundant solutions, and 2) allows a fine-grained customization and dynamic adaptation of the services.

2.2 Describing the adaptation strategy

The key element of the adaptation mechanisms of MolèNE is a set of automata that describes the reactions of the MolèNE components to changes on the execution conditions. These automata constitute the distributed adaptation service of MolèNE. Each automaton is described by application designers as a set of states that represent execution conditions and transitions which determine the reaction to be performed when changing from one state to another. The reactive system is then responsible for interpreting the automata and for executing the corresponding reactions.

The automaton for a component is described as an initial state and a set of reactions. The former specifies the initial behavior for the component and the execution conditions (the state of the automaton) on which it can be used. The latter describes the reaction to be performed for each transition in the automaton. Two types of reactions may be associated to a transition. The first one modifies some of the parameters of the component behavior. The second one changes the behavior itself.

Modifying the parameters of the component behavior. When applied by the *Controller*, this reaction changes the value of some parameters of the *Implementation* object of the component. Applications must specify on the reaction, the parameters to be changed and the new values.

Modifying the component behavior itself. This reaction asks the *Controller* to replace the *Implementation* object of the component. One important aspect of this replacement is the transfer of information between the old and the new object. As an example, consider the component responsible to decide which data to load into the mobile computer cache. This component requires a connection to the data management system in order to store the data when they arrive. Changing the behavior of this component consists on replacing the algorithm used to choose the data to be load. However, the connection to the data management system can be used by the new algorithm and, thus, must be transfered to it.

MolèNE provides the mechanisms allowing the transfer of information between *Implementation* objects. These mechanisms are performed by the objects to be replaced as well as the adaptation strategy. When an object is replaced by another, it must construct its *internal state* that is composed of the set of informations that it uses when it is functional. This state is modified by the *state adapter* associated to the reaction by applications in order to build a new internal state that can be interpreted by the new behavior.

2.3 The Reactive System

The *Controller* of a component (see Section 2.1) ensures the execution of the adaptation strategy defined by applications. The most difficult reaction to implement is the replacement of behavior since relations of the component with other elements must be considered. In order to perform this type of reactions, *Implementation* object collaborates with the *Controller* by providing its internal state that will be modified by the state adapter associated to the reaction.

Modifying the behavior of a component implies the collaboration of three elements in MolèNE: 1) the *Controller* that decides when to execute a reaction, 2) the *Implementation* object that asks for information to other components and builds its internal state, and 3) the state adapter associated to the reaction that implements the passage from one state to another. All of them are used by the Distribution Service available in MolèNE and presented in the next section.

3 The Distribution Service

Designing a distribution algorithm that satisfies all types of applications needs and takes all possible environment changes into account is not a realistic approach. Therefore, we aim at providing not a particular distribution algorithm but a framework, AeDEn, into which designers can insert their own distribution algorithm according to their applications needs and changes of the execution environment.

Traditionally, a distribution algorithm is composed of three policies [12]: *the information policy* which specifies the nature of informations needed for the election (CPU load, etc) and how they are collected, *the election policy* which elects processes which must be suspended, placed or migrated according to the collected informations, and *the placement policy* which chooses the station for each elected process.

Distribution functionalities are provided by AeDEn services which implement the different policies as illustrated in Fig. 2. The election and placement policies are part of the Distribution Service while the information policy is shared between the Data Manager Service and the Detection & Notification Service. The former stores the resources and services states. The latter detects environment changes and notifies the Data Manager Service accordingly.

AeDEn services benefit from two dimensions of adaptation. The first one allows applications to customize AeDEn to their needs. The second one refers to the capabilities of AeDEn to adapt to fluctuations in the environment.

3.1 Application-adaptive Distribution

Applications interaction with AeDEn takes place at launching time. Figure 2 illustrates the steps performed by AeDEn at this time. Applications should provide the Distribution Service, and more concretely the election policy, with the set of components that can be distributed (1,2). The election policy interacts

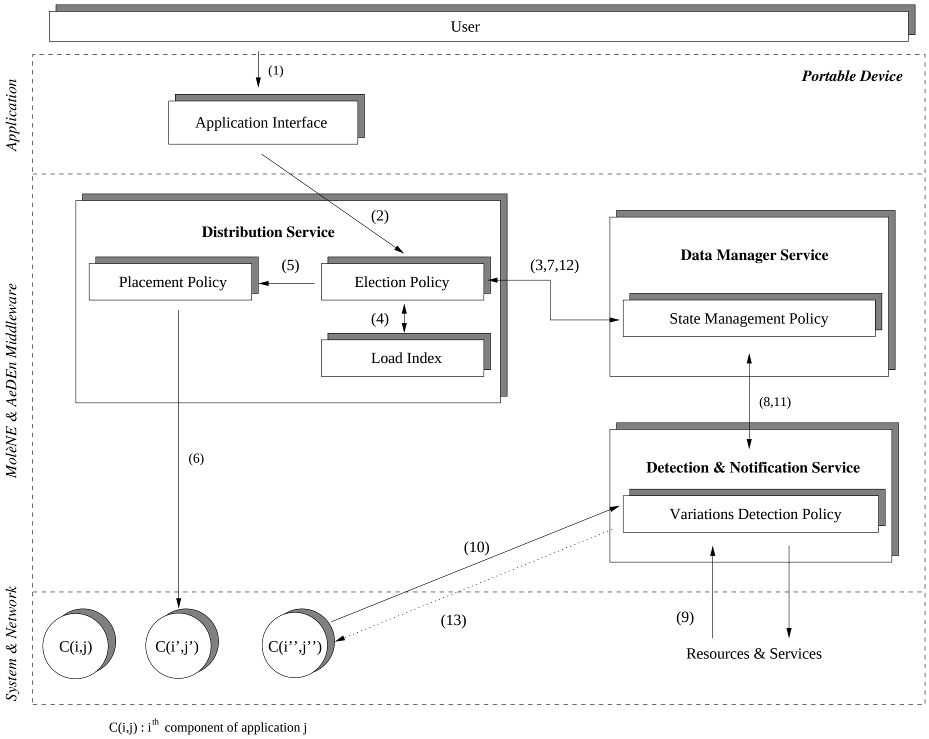

Figure 2. AeDEn architecture

with the Data Manager Service in order to ask for resources and services states (3) so that load indices can be calculated (4). The components are then submitted to the placement policy (5) which determines the most suitable stations for them and performs their transfer (6). It registers the components in the Data Manager Service (7) and activates the monitoring of these components (8).

3.2 Environment-aware Distribution

Changes in the environment are taken into account by AeDEn services at two different levels: the application and the distribution system itself. The first one concerns the mapping of the application onto a set of stations. According to environment changes and to policies decisions, AeDEn services dynamically change the placement of applications components. The second level allows to dynamically change each policy of the distribution algorithm used to decide about the mapping.

Dynamic Placement. When the Detection & Notification Service detects a change in resources or services of the environment, step (9) in Fig. 2, or in components (for example a removal) (10), it notifies the Data Manager Service which updates its information database (11) and notifies the Distribution Service if some of the distributed components are involved[1] (12). Then, the election policy determines if these changes affect the placement. In this case, a new placement is performed (3,4,5,6) which may imply the placement of new components, the suspension, and/or the migration of existing ones.

Dynamic Policies. As we said in the beginning of the section, a particular policy can be more or less relevant depending on the application needs and constraints. In order to ensure the suitability of AeDEn in all cases, we introduce *dynamic policies* which change their implementation as changes occur in the environment. Each dynamic policy has only one implementation at time t but can dynamically change and have a different implementation at time $t+1$.

This opportunity to dynamically change the implementation of a policy requires well-adapted and efficient programming tools. We use the MolèNE object-oriented framework which provides these adaptation facilities. Each AeDEn policy is implemented as a MolèNE component. Interactions among themselves and with the application and the environment are therefore performed by sending/receiving events and the different possible implementations for a policy are provided as *Implementation* objects.

We illustrate the interest of dynamic policies by taking the example of the election policy (see Fig. 3). The events received by the *Interaction* object of this policy originate from the application level or the Detection & Notification Service while the placement policy receives those that are sent. These events correspond to those described in Sections 3.1 and 3.2. The notification of changes in the environment are also implemented as events that are sent by the Detection & Notification Service to the *Controller* of the election policy. They indicate that the current election *Implementation* object may be no longer adequate. The current *Implementation* is then changed by the *Controller* according to the *Adaptation Strategy* object provided by the application.

For example, let's consider a user working with a laptop. As long as the laptop is connected via a dock-station to a local network with a central server, it is reasonable to have a centralized and total election policy. When the laptop moves into the building, keeping the connection via a wireless link, a better policy would be centralized and partial in order to reduce the network traffic. If the laptop moves out-door and is connected to an ad-hoc network, the policy should be switched to a distributed and partial one.

This mechanism of dynamic policy is also used for the other policies of the distribution algorithm and for the load index. The load index can dynamically change to take into account mobility criteria such as the low bandwidth, the probability of disconnection, and the battery index.

[1] Application components can also be designed to adapt themselves according to environment changes. In this case, they are directly notified by the Detection & Notification Service (13).

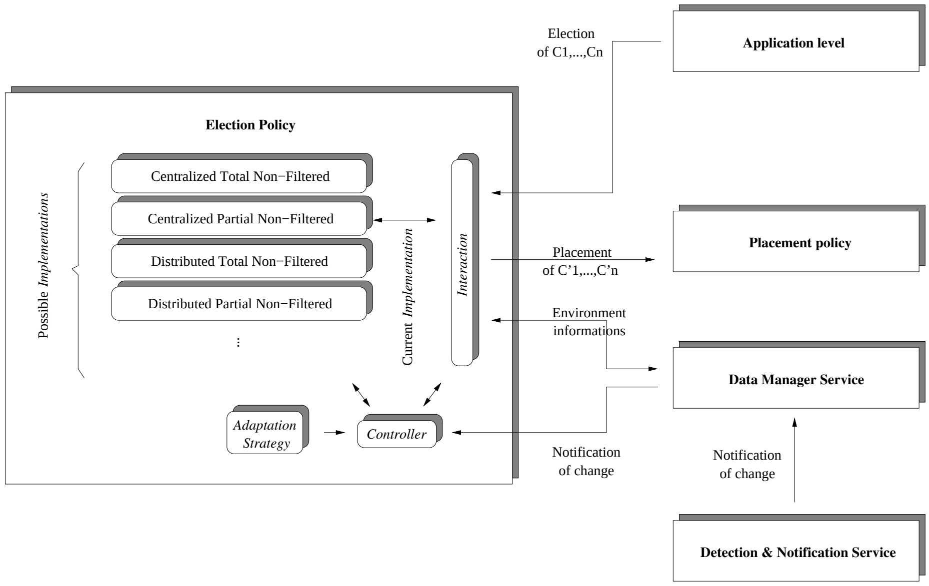

Figure 3. Strategies for the election policy

4 Related Work

Adaptation. Dynamic adaptation to execution conditions has become a hot topic in the last years and several systems propose their dynamic adaptation to changing conditions. Odyssey [8] allows applications to specify a data access strategy depending on the type of the accessed data and current resources availability. Mobiware [2] proposes a programmable mobile network on which mobile computers, access points and mobile-capable routers are represented by objects that can be programmed by applications to support adaptive quality of service assurances. Ensemble [9] is a network protocol architecture constructed from simple *micro-protocol* modules which can be stacked in a variety of ways to meet the communication demands of its applications. The adaptation mechanisms proposed by these approaches are specific to the problem they address: the bandwidth variability. MolèNE mechanisms such as the adaptation interface or the *Controller* element of a component are general and can provide solution to different types of problems.

A more generic mechanism is proposed by the 2K system [6]. It performs the reconfiguration of components based on the reification of components dependences. This mechanism is complementary of the one provided by the *Controller* of a MolèNE component. In our system we have emphasized the adaptation of the components themselves which is not defined in the 2K system. Moreover, we are currently formalizing a way to manage relations among adaptive components.

Distribution. Few approaches propose the utilization of remote resources to overcome their lack on portable devices. The mobile agent model is one of them [5]. This model allows to delegate actions such as information gathering and filtering to the mobile agent on the fixed network so that only the resulting data are transmitted to the mobile client. In the Client/Proxy/Server model, a proxy runs on a fixed station and uses its resources to provide facilities for the mobile user such as efficient new network protocols, storage of results while disconnections or filtering such as degradation or display media conversion [11].

Proxies can also be distributed to increase the number of available resources. For example, distribution techniques are used in Daedalus [4] to deploy adaptation-based proxy services on a cluster of PCs. The applied strategy consists in increasing the number of proxies according to the number of requests in waiting queues. This approach is nevertheless specific: distributed processes are exclusively proxies defined by the adaptation system designer for a particular task; the distribution strategy is also fixed and cannot take new criteria into account. To distribute an application in a more general way, distribution techniques proposed in the distributed computing area should rather be considered.

Many distribution algorithms exist which ensure good performance, load-balancing and stability [3, 7]. However, these algorithms perform load sharing among stations based only on CPU criterion. Multi-criteria approaches also exist [13] but they do not take into account the new criteria introduced by mobile computing such as a low and variable bandwidth, the probability of disconnection (previous approaches assume to have a high and continuous connection) or a battery index.

5 Conclusion

In this paper, we have presented AeDEn, a system which provides adaptive and dynamic mechanisms to distribute applications in an extremely varying environment. This distribution allows to overcome the poorness of available resources on portable devices by using resources and services of the environment. It also allows to reduce variability effects by executing applications components on stations less sensitive to variations. Our mechanisms are application-adaptive in the sense that they take into account application needs. Moreover, environmental changes are considered not only by allowing the dynamic placement of the application but also by dynamically changing the policies used to decide about the placement.

AeDEn has been constructed as a part of a more general system, MolèNE, that has been designed in order to facilitate the construction of adaptive solutions to manage wireless issues. MolèNE is structured as a set of self-adaptive components thanks to the *Controller* object encapsulated into their architecture. MolèNE extensibility is ensured by the distinction between the interface and the implementation of a component and the utilization of introspection and reflexion techniques at the interface level.

We have implemented a preliminary AeDEn prototype using the MolèNE components developed in Java. We are currently integrating it in the MolèNE system and we plan to test the different distribution mechanisms with a real electronic press application. This application is a good candidate for our distribution system since it is interactive, manipulates an important volume of data and requires quite a lot of computation power.

References

[1] F. André, A.-M. Kermarrec, and F. Le Mouël. Improving the QoS via an Adaptive and Dynamic Distribution of Applications in a Mobile Environment. In *Proc. of the 19th IEEE Symp. on Reliable Distributed Systems*, Nürnberg, Germany, October 2000.

[2] O. Angin, A.T. Campbell, M.E. Kounavis, and R.R.-F. Liao. The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking. *IEEE Personal Communications Magazine*, August 1998.

[3] B. Folliot and P. Sens. GATOSTAR: A Fault Tolerant Load Sharing Facility for Parallel Applications. In *Proc. of the 1st European Dependable Computing Conf.*, volume 852 of *Lecture Notes in Computer Science*. Springer-Verlag, October 1994.

[4] A. Fox, S.D. Gribble, Y. Chawathe, and E.A. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives. *IEEE Personal Communications*, 5(4), August 1998.

[5] R. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko. Mobile agents for mobile computing. Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, May 1996.

[6] F. Kon and R.H. Campbell. Supporting Automatic Configuration of Component-Based Distributed Systems. In *Proc. of the 5th USENIX Conf. on Object-Oriented Technologies and Systems*, California, USA, May 1999.

[7] M.J. Litzkow and M. Livny. Experience With The Condor Distributed Batch System. In *Proc. of the IEEE Work. on Experimental Distributed Systems*, Alabama, USA, October 1990.

[8] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proc. of the 16th Symp. on Operating Systems Principles*, St. Malo, France, October 1997.

[9] R. Van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building Adaptive Systems Using Ensemble. *Software-Practice and Experience. Special Issue on Multiprocessor Operating Systems*, 28(9), July 1998.

[10] M.T. Segarra and F. André. A Framework for Dynamic Adaptation in Wireless Environments. In *Proc. of the Technology of Object-Oriented Languages and Systems*, Saint Malo, France, June 2000.

[11] Wireless Internet Today. *Wireless Application Protocol - White Paper*, October 1999.

[12] S. Zhou. A Trace-driven Simulation Study of Dynamic Load Balancing. *IEEE Transactions on Software Engineering*, 14(9), September 1988.

[13] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Systems. *Software - Practice and Experience*, 23(12), December 1993.