A Generic Approach to Build Mobile Applications

Françoise André, Maria-Teresa Segarra

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Generic Approach to Build Mobile Applications

Françoise André et Maria-Teresa Segarra

N ° 3723

Juin 1999

THÈME 1

ISSN 0249-6399

A Generic Approach to Build Mobile Applications

Françoise André et Maria-Teresa Segarra

Thème 1 — Réseaux et systèmes
Projet Solidor

Rapport de recherche n3723 — Juin 1999 — 22 pages

Abstract: Mobile environments are characterized by their variable resources (such as network bandwidth), limited amount of resources (memory, processor), and their battery-dependent functioning. These characteristics necessitate the redesign of applications in order to maintain a good quality of service for execution in such environments. Existing approaches provide application-specific solutions which greatly reduces code reuse. Further, mobility issues are embedded in the application functionality making them more difficult to develop and to maintain. In this paper, we propose an object-oriented framework, MolèNE, that provides a set of generic base services as the abstraction of concepts usually utilized to deal with mobility issues. The generic nature of our approach allows services to be customized to different applications. On the other hand, resources management is an important issue in a mobile context. Therefore, MolèNE services have been designed to adapt their behavior depending on resources availability. Moreover, applications needs are taken into account when the adaptation implies the modification of application semantics. All these features make MolèNE a highly adaptable system unique in the mobile computing domain. To illustrate MolèNE use, we present the caching service and its customization for two different applications, a transactional internet application and a file system.

Key-words: wireless environments, adaptive systems, generic approach.

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Une approche générique pour la construction d'applications mobiles

Résumé : Les environnements mobiles se caractérisent par la variabilité de ressources comme la bande passante du réseau, leur capacité limitée notamment en mémoire et processeur, ainsi que par leur dépendance vis à vis d'une batterie pour leur fonctionnement. Ces caractéristiques n'ayant pas été prises en compte lors de la conception des applications existantes, celles-ci ne sont pas bien adaptées à une exécution dans un contexte mobile. Les approches qui tentent d'aborder ce problème fournissent des solutions dédiées au contexte particulier de l'application, ce qui limite la réutilisation de code. De plus, leurs mises en oeuvre sont très imbriquées dans les fonctionnalités des applications ce qui les rend difficiles à développer et à maintenir. Dans cet article nous proposons l'utilisation d'un framework orienté objet, MolèNE, qui fournit des services génériques de base par abstraction de concepts souvent utilisés pour gérer la mobilité. Cette généricité permet la spécialisation des services pour différents types d'applications. De plus, la gestion de ressources étant un facteur important des environnements mobiles, l'adaptation du fonctionnement des services MolèNE aux conditions d'exécution (en termes de ressources disponibles) devient incontournable. Les services de MolèNE sont conçus pour permettre plusieurs types d'adaptation en fonction des besoins des applications qui les utilisent. Cette propriété rend MolèNE un système hautement adaptable permettant une gestion très fine des ressources disponibles. Pour illustrer l'utilisation de MolèNE, nous présentons le service de chargement de cache et sa spécialisation pour deux applications différentes, une application transactionnelle sur le Web et un système de fichiers.

Mots-clé : environnements sans fil, systèmes adaptatifs, approche générique.

1 Introduction

The use of portable computers connected through wireless networks to fixed ones is rapidly spreading. The characteristics of these wireless environments[1] are different from those of fixed stations connected through wired links [9]. Indeed, wireless networks suffer from low bandwidth, high bandwidth variability and periods of disconnection due to radio interferences. Also, the portable computer often has a limited amount of resources: few ram memory capacity, small and low resolution screen. This affects all the layers involved in the execution of an application on a portable computer, from the physical communication layer up to the application software itself. Even if new protocols at the network level tend to hide the effect of mobility ([3], [4], [19]), applications cannot remain unchanged. For example, locking mechanisms which constitute the basis for most of the consistency algorithms in database systems, are not convenient in the mobile context as the locking periods may be very long. Further, mobile devices depend on a limited energy supply, a battery. As energy consumption depends on the activity of devices such as the processor, disk, display or network interface, applications behavior should adapt to optimize resources utilization and, thus, reduce energy consumption. Therefore, the redesign, at least partially, of the applications should be envisaged.

Solutions to build adaptive applications for mobile environments have already been proposed. In our opinion, these solutions can be divided into two categories:

- Solutions that provide basic mechanisms to allow applications to adapt to changing execution conditions. In this type of systems the burden of the work still remains on applications developers. Rover [10] provides the Relocatable Dynamic Objects (RDOs) and the mechanisms to perform relocations, detect inconsistencies, and manage disconnections for them. Developers should encapsulate applications semantics as RDOs and should provide their own mechanisms to decide when to migrate functionality.

- Solutions that provide more advanced mechanisms to monitor the execution context and to adapt to changes. For example, Odyssey [17] provides applications with a set of ready-to-use data access policies and a context detection/notification subsystem. Applications are responsible for informing Odyssey about: 1) when the execution conditions have significantly changed for it, and 2) the policy to be used upon current execution context. When a significant change occurs, Odyssey informs the concerned application which decides the new policy to be utilized.

Although these efforts have allowed to make significant progress in the mobile computing domain, we think that they present some inconvenients when utilized in the real world context. The first type of approach needs a great effort of applications rewriting, a task that most developers are not ready to perform due to the huge code sizes. Drawbacks of the latter type of solutions correspond to those of a monolithic architecture. As the detection

[1] We consider a wireless and a mobile environment as the same type of context.

subsystem is embedded in the adaptation one, the global functioning becomes rigid. It is impossible to define new resources and variations making these approaches not extensible.

The approach followed by our system, MolèNE (MObiLE Networking Environment), is different from both previously presented approaches as the system design has been guided by the following desirable properties:

- Generic nature of the approach: MolèNE design allows programmers to customize MolèNE to their particular applications.
- Adaptability to environment conditions: the components that make up the MolèNE system are able to dynamically modify their behavior depending on the execution context and according to applications requirements
- Flexibility and extensibility: MolèNE has not been designed as a "closed system" but as a flexible one. It allows applications to inject new components in the system thus providing means to extend MolèNE.

The application domain covered by MolèNE concerns distributed data-intensive applications. This software accesses data available in a (possibly remote) information system and performs their computations on them. A transactional e-commerce application on the Internet, a video conference application or a weather forecast application are examples of software that can benefit from our system. MolèNE executes between the information system and the applications that use it, adapting data management to the needs of each particular application and execution context. In order to achieve this, it relies on a set of low-level services that may be composed together to offer the quality of service required by the applications. Such services include a data access service, a caching service, a consistency service, and a networking service. For a particular application and wireless environment, the application developer chooses the required services. Some of them will be implemented on the portable computer, others on the fixed network, the cooperation between them being precisely defined in our system.

All the properties of MolèNE are provided by extending the benefits of the object-oriented programming to the information systems in wireless contexts. Therefore, MolèNE is designed as an object-oriented framework which has to be 1) customized by the middleware programmer so that an instance of the framework is able to interact with a particular information system, and 2) informed by applications developers about applications preferences so that the particular middleware is able to adapt its behavior to the execution environment. In order to achieve this, each service in our system is encapsulated in an object of the framework and its interface is decoupled from its implementation. This well-known technique of object-oriented programming is utilized at two levels on the framework making MolèNE a highly flexible system well-suited to wireless environments. Moreover, a large amount of code can be reused and, as mobility issues are managed in the middleware, the impact of mobility is confined in specific code.

In this paper we will describe MolèNE in detail and show how its caching service is customized for two different information systems: a distributed file system (MFS) and a

transactional Web-based information system (METIS). Section 2 introduces the concept of object-oriented frameworks and the most frequently used object-oriented mechanisms that are used by them. Section 3 presents the main characteristics of the MolèNE system: its generic nature and flexibility. The architecture of MolèNE is discussed in section 4 and its main services are presented. Section 5 concentrates on a particular service of MolèNE, the caching of data, and its customization to MFS and METIS. Conclusion and future work are given in section 7.

2 Frameworks

A framework is a set of cooperating classes which constitutes a reusable design for a specific class of software. A framework is customized to a particular application by creating application specific subclasses of abstract classes from the framework. The framework dictates the architecture of the application. It defines the overall structure, its partitioning into classes and objects, the key responsibilities of each component, how the classes and objects collaborate and the thread of control.

Using a framework provides two main advantages. First, most of the body of the application is given by the framework, so that only a small amount of specific code should be implemented by the developer. Second, frameworks are easily extensible and adaptable. Extensibility is ensured by adding new classes and inheriting from existing ones. Adaptability is ensured by instantiating only the framework components needed for a particular application and by subclassing existing ones.

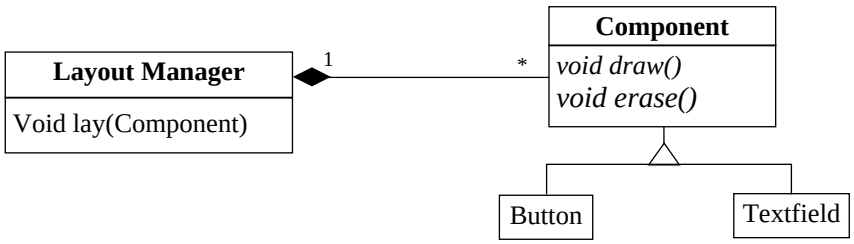

Figure 1: Example of polymorphism

Building a framework requires a complete knowledge of the application domain covered by the framework. When this knowledge is acquired, it is necessary to abstract the identified functionalities in order to make a framework usable for a large amount of future applications of the covered domain. The danger of making a framework based on past applications is to make it tailored to these applications and not easily adaptable for future ones. Therefore, there has to be a trade-off between making abstractions that will translate to abstract classes and making implementations that will translate to concrete ones. Having too many abstract classes reduces code reuse of the framework but adapting it to a particular application is an

easy task. Having too many concrete classes brings about the opposite effect: it facilitates code reuse but the framework becomes difficult to adapt to a particular application.

Frameworks make a great use of object-oriented concepts like composition and polymorphism. Composition is the mean for one object to use the functionality of another one at run time. In order to achieve this, an object must have the reference of another one. Composition is largely used for code reuse. Polymorphism extends the concept of composition by allowing a concrete subclass to be used wherever the abstract super class was expected. It allows an object `O` to use at run time the functionalities of another object `O'` without knowing its type. `O` needs only to know the interface of `O'`. Polymorphism is a way to make a framework extensible and adaptable to a particular application. Consider the example of a framework for building applications that put graphical objects such as buttons and textfields, in a window. Such applications need a layout manager. If one makes a layout manager that only knows how to lay buttons and textfields out, this solution is not extensible because adding images for example implies the modification of the code of the layout manager. Another solution is to use polymorphism. The architecture derived from polymorphism is shown in figure 1[2].

The *Component* class provides two methods, `draw()` and `erase()`, that put and delete a *Component*, respectively. The `lay(Component)` method of the *LayoutManager* class lays out a *Component* in the window. The *Button* and *Textfield* classes implement the *Component* class each in a different way. At run time a specific *Component* will be laid out and the correct implementation of the methods will be called. Therefore, the *LayoutManager* implementation becomes independent from specific components. It simply knows the *Component* class and the specification of each method of the class. In this way, a developer can easily add new components such as images.

3 Characteristics of MolèNE

Existing applications are designed for environments in which resources availability is sufficient and does not vary much over time. With the advent of mobile devices these assumptions are no more valid: resources are scarce compared to those in desktop computers and may significantly change over time. Therefore, applications design should be revised in order for applications to execute in a mobile environment with an acceptable quality of service for their users. Issues like data caching, consistency management and remote information access should be addressed in a more appropriate way. Moreover, as resources availability may change greatly over time, means for adaptation should be provided. This may be a difficult task for an application programmer not used to manage these issues and if the mechanisms to remedy all these problems are not well isolated, they may obscure the functionalities of the application making it more difficult to develop and maintain.

MolèNE aims at alleviating the task of programmers faced with the problem of making an application scalable i.e. being able to execute in environments of different characteristics.

[2] Abstract methods in the figures of the paper are represented in italics as mentioned in [21].

In order to achieve this, MolèNE offers a set of generic services dealing with mobility issues. The ideal generic nature would be such that all services perform their task regardless of the application that uses them. Unfortunately, the wide disparity of applications and data types requires some of the services to be dependent on the application. For example, the consistency requirements of data from a NFS server differ substantially from those of a relational database server. Therefore, some services need a feedback from the applications in order to perform their work. MolèNE provides programmers with a significant amount of ready-to-use mechanisms that adapt to the utilizing application.

3.1 The Generic Nature of MolèNE

Object-oriented frameworks aim at helping programmers to develop applications by offering them a set of cooperating classes that determines the architecture of the application. The generic nature of frameworks does not come from the sharing of an entity by a set of applications but from the abstraction of typical concepts for a particular type of application. Programmers need only to implement those parts of the framework that depend on the application rather than designing the whole. This *"fill in the gaps"* corresponds to the implementation of the abstract methods of the framework; relations among classes are established by the framework itself. MolèNE uses this concept and stresses it in order to maximize the reuse of components and to ease the adaptation which is an important feature in mobile environments. MolèNE architecture is divided into two levels as depicted in figure 2.

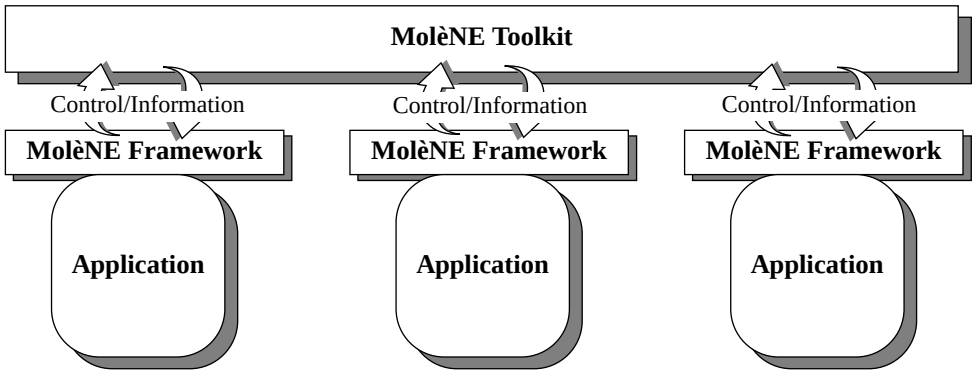

Figure 2: General architecture of MolèNE

The MolèNE *toolkit* contains services that are completely independent of a particular application and that we will refer as *tools*. They are executed at the computer level thus being shared by applications and correspond to concrete classes that do not depend on any abstract one. Services such as the management of traffic coming/going from/to the network

interface or the measurement of resources availability are good candidates to be *tools* of MolèNE.

The second layer that we call MolèNE *framework*, contains functions that depend on applications. These are called *services* and correspond to abstract classes and concrete ones depending on them. The MolèNE *framework* utilizes the *tools* to perform its tasks and is executed at the application level i.e. each application utilizes it to manage issues related to mobility. In contrast to the *tools*, the *services* should be tailored to a particular application. The customization comes from the exploitation of the inheritance and polymorphism mechanisms. As an example, different applications may use different units of information (e.g. rows for a database management system, blocks of files for a filesystem) which may be stored in different repositories (e.g. databases, file repositories). To deal with this heterogeneity, the class *Data* in the *framework* is an abstraction of the units of information managed by applications. Its methods allow the data items to be stored or removed from the data repository and thus are abstract methods. The programmer of an application that works on rows of a database, for example, creates a class *Row* that inherits from the *Data* one and implements its methods. Therefore, each row of the database is managed in the *framework* as a *Data* object (utilization of polymorphism) and it can be stored or removed from the database by calling its methods. These methods are easy to implement (an SQL insert or delete operation for the database application) and allow MolèNE to carry out other tasks much more complicated such as data caching or consistency.

3.2 Adaptation in MolèNE

Another goal of MolèNE is to provide means for adaptation to changing conditions. Applications which behavior changes according to the perceived constraints in the environment are called adaptive applications. For example, the consistency guaranties of an application may be relaxed when the network bandwidth becomes scarce to reduce network traffic. Therefore, means for monitoring the environment and adapting to its changes are provided in MolèNE as *tools*. As a global view of the system is available at the *toolkit* level, adaptation decisions can be taken according to the requirements of all applications being executed rather than those of a particular one allowing a better utilization of the computing substrate.

In order to ease the adaptation, *services* are implemented by three types of objects. Each *service* is represented by a *Service* object which role is divided into subtasks, referred in the paper as *microServices*, encapsulated in *MicroService* objects. Further, the implementation of a particular *microService* is performed by an *Implementation* object. Relations among these objects are determined by the bridge design pattern (see figure 3). This pattern *"decouples an abstraction from its implementation so that the two can vary independently"* [7]. By encapsulating in an object implementations for a particular abstraction and establishing a composition between the abstraction and its implementation, the modification of the latter becomes simple. This pattern is applied in MolèNE at two levels. In the services-level bridge *Services* play the role of the abstraction while *MicroService* objects play that of their implementation. On the other hand, the microServices-level bridge establishes a composi-

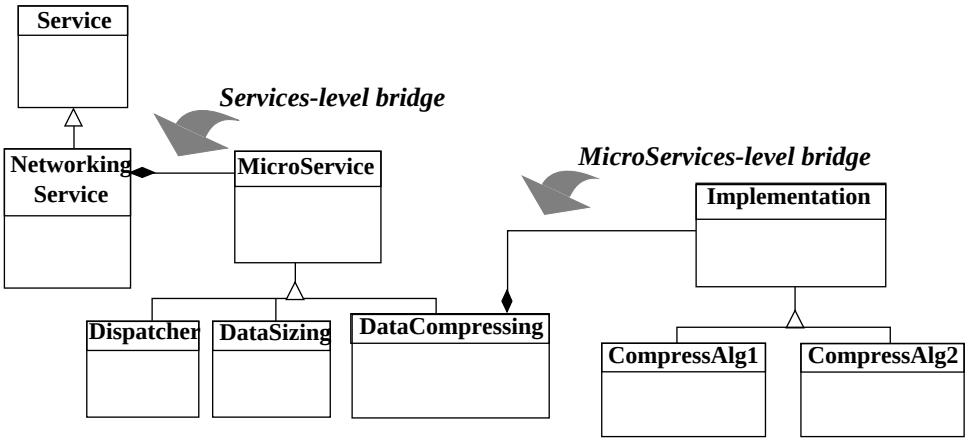

Figure 3: A two-level bridge architecture of MolèNE for the Network ing Service

tion between *MicroService* and *Implementation* objects so that it is possible to dynamically change the implementation of a *MicroService*.

In order to show the high degree of flexibility of MolèNE, let us consider the example of the management of network traffic for an application. This is accomplished by the *Networking Service* class. Relations between the objects involved in this *service* and the parent classes namely *Service*, *MicroService*, and *Implementation* classes are shown in figure 3. The main goal, which is performed by the *Dispatcher microService*, is to dispatch traffic coming from the network to the corresponding service depending on the type of the received information. For example, the response to a caching request should be forwarded to the *service* that manages cache content. Moreover, as all network traffic generated by an application is centralized by this *service*, a *DataCompression microService* may be used that formats data to adapt them to the characteristics of the mobile environment. It is only utilized when adaptation is necessary, thus changing the structure of the *Networking Service*. Different compression algorithms exist each consuming resources in a different way. Therefore, the utilized compression method can be substituted for another one consuming less resources when battery life becomes scarce. Finally, due to the small size that usually characterizes the display of mobile computers, video or images have to be scaled to be correctly displayed. This operation is carried out by the *DataSizing microService*. Again, depending on the battery life of the mobile computer it may be executed either on the server of the data or on the mobile computer thus changing its location.

Adaptation in MolèNE can thus be performed by changing the structure of a *service* or changing the implementation and/or location of a *microService*. The next section presents how MolèNE is able to carry out these types of adaptation.

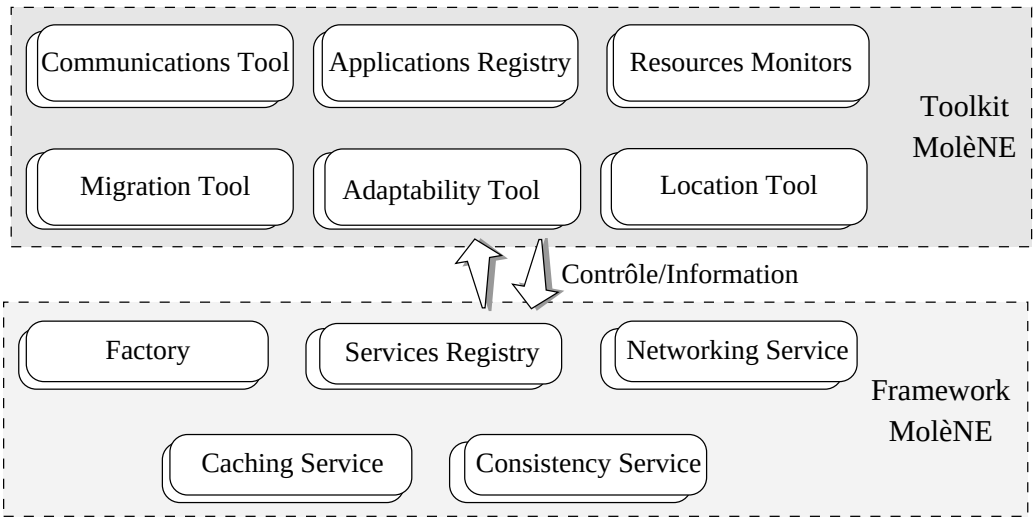

Figure 4: MolèNE *tools* and *services*

4 The MolèNE System

We describe below the different functionalities provided by the MolèNE system. MolèNE is divided in two parts, the MolèNE *toolkit* and the MolèNE *framework*. Figure 4 shows some of the *tools* and *services* available in MolèNE.

4.1 The MolèNE Toolkit

System administrators determine which of the *tools* should be executed for a particular environment depending on the functionalities they want to give. The *Communications Tool* is responsible for sending and receiving information from the network interface and for managing connections of the applications. The *Migration Tool* is required if computations coming from remote sites are authorized to execute. It needs the *Communications Tool* to distribute computations.

The *Resources Monitors* measure resources availability such as processor, memory and battery life and let other components know about it. The *Applications Registry* maintains information about the identity of applications that are currently being executed. The *Adaptability Tool* is the core of the dynamic adaptation in MolèNE. It triggers adaptation according to the resources availability and applications needs.

The *Service*, *MicroService*, and *Implementation* classes that constitute the *services* of the *framework* encapsulate useful information to perform the adaptation (see figure 5). Therefore, the *Adaptability Tool* utilizes their attributes and methods to adapt the behavior of

the *services* i.e. to modify the current implementation and/or location of their composing *microServices*.

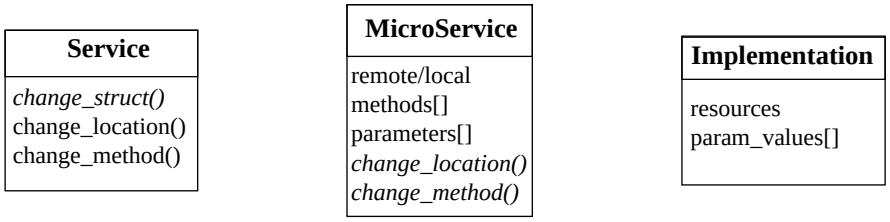

Figure 5: Available information for adaptation

The *Service* class provides three methods corresponding to the three types of adaptation that are possible in MolèNE. The `change_struct()` method provides means for changing the *MicroService* objects that implement a *Service*. The `change_location()` and `change_method()` methods change the location and implementation of a *MicroService* object, respectively. The *MicroService* class encapsulates information related to the adaptation at the *microServices* level. This information includes if the location of the *microService* can be changed, the parameters that characterize its adaptation, and the `change_location()` and `change_method()` methods that carry out the adaptation of the *microService*. Finally, the *Implementation* class contains information about the resources utilized by the concrete implementation of a *microService* and the value of the parameters for this implementation.

Consider the example of caching of data items. Caching of frequently-accessed data is critical for reducing contention on the narrow bandwidth channels of wireless networks. This task manages data items mentioned in section 3.1 and thus, it should be provided by the *framework*. One of its implementing *microServices* is the *ReplaceμService* which is responsible for deciding when to replace which *Data*. It is characterized by two parameters: `which` and `when` to replace some *Data* from the cache. A number of strategies are possible in order to decide which data should be replaced. The most frequently used strategy considers the time of the last read of the data. However, other strategies may be considered in a mobile environment such as considering the loading time of the data. Applying the chosen strategy may be performed `"on demand"` or `"periodically"`. The most frequently used strategy is the first one, in which the *ReplaceμService* chooses the *Data* to be evicted when new *Data* should be cached. In the second case, *Data* are evicted when the amount of used memory exceeds a threshold.

During the registration process of an application, the *Adaptability Tool* is informed about *services* that are required and the preferences (if any) of the application concerning their implementation. If the application expresses a preference, the *Adaptability Tool* will take it into account when some adaptation should be performed. If it cannot be satisfied, the application is informed. For each *service*, the *Adaptability Tool* keeps track of the implementation strategy for the applications that use the *service*, i.e. what are the composing

microServices, their location and concrete implementation. In this way, it is able to know about current configuration and resource consumption and to take decisions based on them.

Extensibility of MolèNE is thus ensured. Applications can create their own implementations for a particular *microService* by inheritance of the *Implementation* class. They only have to specify the resource consumption (such as the amount of memory utilized by the class) and the value for the characterizing parameters of the *microService*. As all the information useful for the adaptation is encapsulated in the class, new implementations may be easily taken into account by MolèNE. Adding new *services* is more complicated and requires skilled programmers. However, once available, other applications are able to use them if required.

4.2 The MolèNE Framework

It determines the interface between the *toolkit* and the applications. It is composed of a set of cooperating classes that implement only those *services* that are required by a specific application. For example, an application that does not want to cache information (for security reasons for example) does not need such a *service* for it. Some of the available *services* in MolèNE are shown in figure 4.

All *services* in MolèNE are asymmetric, i.e. they have a client and a server side that work together to carry out their task. Consider again the example of caching of data items. Loading data items into the cache is performed by the *FetchμService* class. Its client side is responsible for sending loading requests of *Data* items to be cached. The server side gets the required *Data* items and sends them to the client. This asymmetry is modeled by providing a client/server architecture for each *service*. Both, the client and the server side inherit from the *Service* class to provide means for adaptation. The management of this architecture is a MolèNE responsibility and completely transparent to applications.

Two *services* are involved in the initialization of an application that uses the *framework*. The first one is the *Factory*. This *service* is the only one that an application must create at the beginning of its execution and is responsible for creating objects that are necessary to the proper functioning of the *framework*. The second one is the *Services Registry*. It is responsible for the creation of the *Service* objects depending on the requirements of applications. Their associated *MicroService* and *Implementation* objects (determined by the *Adaptability Tool* as discussed in the previous section) are also created and composition among them is performed.

The *framework* provides the *Data* class as an abstraction of the units of information managed by applications (see section 3.1). Two other classes are related to it as depicted in figure 6. The *Metadata* class extends the information stored in *Data* to provide MolèNE with additional information about the *Data* items such as the loading time for the item or the last read time. This information may be used by the *framework* to define policies for cache replacement for example. The class *Cache* is an abstraction of the data repository(ies) used by applications. Its methods allow the storage or removal of *Data* objects when the corresponding unit of information has been stored or removed from the data repository.

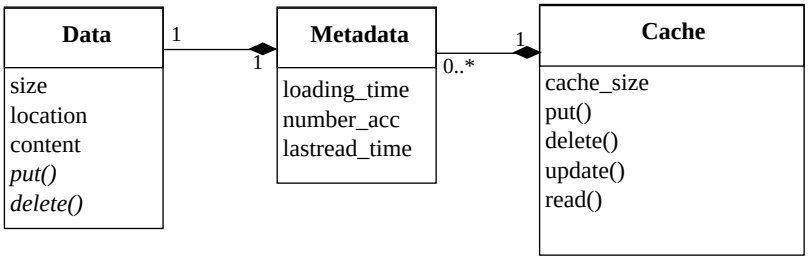

Figure 6: Data management in MolèNE

Two *services* utilize the *Cache* to perform their task: the *Caching* and the *Consistency services*. The first one manages the content of the *Cache* of an application and, as previously mentioned, is composed of the *FetchμService* that fills the *Cache* with *Data* items and the *ReplaceμService* that decides about which *Data* items to replace and when. It is discussed in detail in the next section.

Data caching introduces the problem of ensuring consistency of the data in the cache. Ensuring consistency is also a *service* that is usually performed by mobile applications. MolèNE offers a *Consistency Service* that ensures the consistency of the *Data* items in the *Cache*. Two *microServices* implement it. The *UpdateμService* is responsible for updating the *Cache* with modifications performed by other clients on the copy of the data on the server. The *ReintegrateμService* reintegrate operations carried out on *Data* items of the *Cache* during disconnection. Working disconnected is a common operation in mobile devices to reduce communications costs. While disconnected, operations are carried out on data available in the cache, letting the user continue his work. Upon reconnection, these operations should be applied to the data on the server in order for other users to see them.

5 Customization of a MolèNE Service

In order to show the generic nature and the flexibility of the MolèNE *framework*, this section describes in detail one of the *services* available in MolèNE, the caching of data. Then we show how it has been used to manage local data in two different applications.

5.1 The Caching Service

The classes which compose the *Caching Service* are depicted in figure 7. Five methods are implemented by the *Caching Service* all involved in the caching mechanism. The `read()` method uses the `load()` one when the required *Data* are not available in the *Cache*. This method creates a *DeferredReply* object so that the application is not unnecessarily blocked if it does not need to access immediately the required information and forwards the caching request to the *Networking Service*. The `get()` method is called at the server side when a

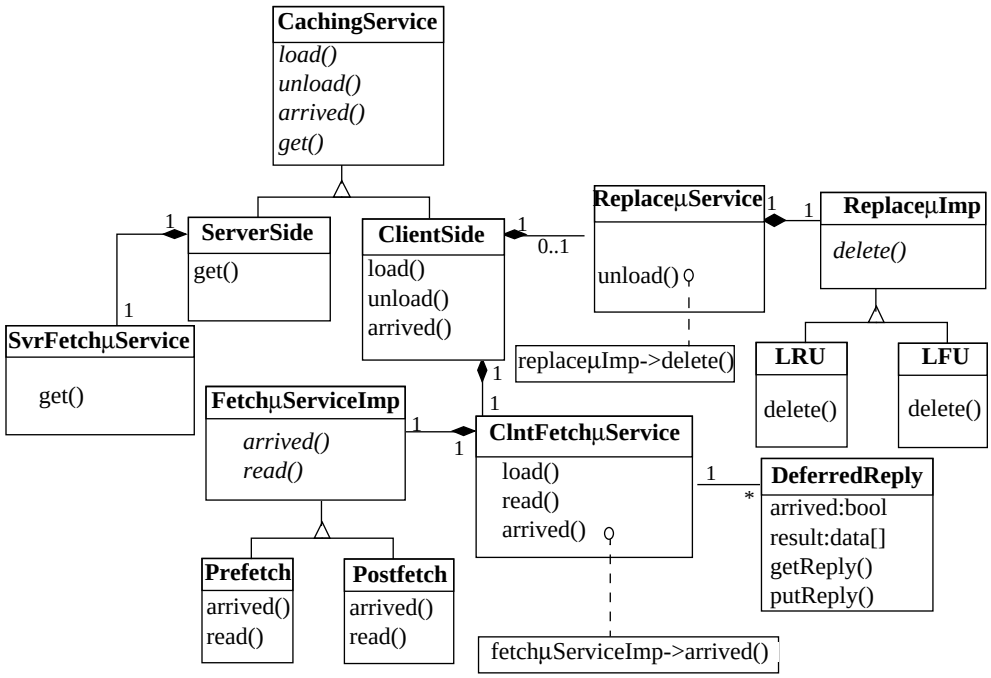

Figure 7: Structure of the *Caching Service* in MolèNE

caching request is received by the *Caching Service*. The reply is built with the required *Data* and sent to the client side. When the required *Data* arrive, the `arrived()` method is called that fills the corresponding *DeferredReply* object with the incoming *Data*. These four methods concern the *FetchμService* as they allow the loading of information into the *Cache* and are distributed between the client and the server side of the service. The `read()` and `arrived()` methods are implemented by the *FetchμServiceImp* object which allows to use different strategies.

The `unload()` method is utilized when some of the *Data* should be evicted from the *Cache*. This method implements the cache replacement algorithm and is provided by the *ReplaceμService*. Different algorithms may be considered to optimize the utilization of the cache depending on the environment conditions. Algorithms to explicitly take into account the loading time of *Data* when the network bandwidth becomes scarce may be used for example. Therefore, the `unload()` method is implemented by the *ReplaceμServiceImp* object which allows once more to easily select between different algorithms.

Generic data types are also provided in the *framework* to easily allow the management of different kinds of information. These generic classes and their relation with the specific ones for the *Caching Service* are shown in figure 8. The *Command* class is an abstraction of the operations that can be performed by a particular *service*. For the *Caching Service*,

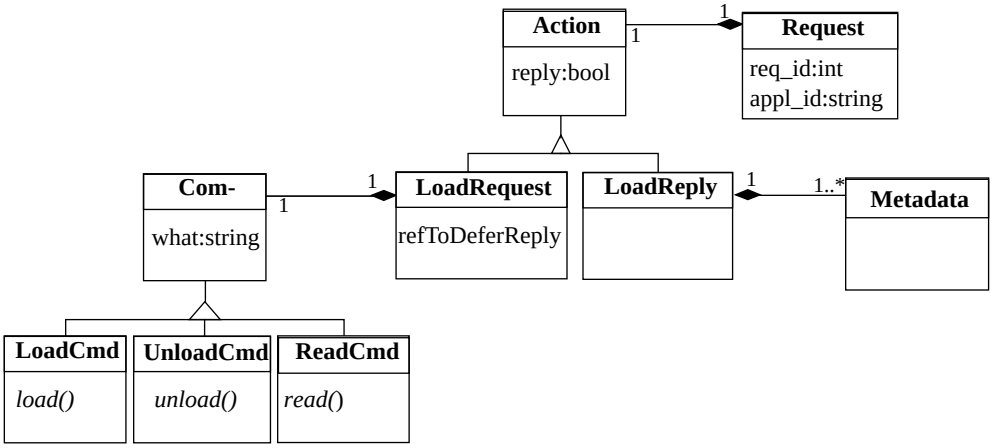

Figure 8: Data types involved when loading some information

Command refers to a load, unload and read operation. For the *LoadCmd* class for example, the `what` attribute indicates what has to be loaded from the server while the `load()` method determines how to submit the what attribute to it.

Different *services* in MolèNE manage different kinds of requests. A caching request for example is quite different from a reintegration one. To deal with this heterogeneity, the *Action* class is provided which is an abstraction of the types of requests available in MolèNE. Its only attribute, `reply`, indicates if the *Action* needs a reply from the destination. The *Caching Service* manages two types of *Actions*, the *LoadRequest* and the *LoadReply*. The first one is managed by the client side of the *Caching Service* and encapsulates the *Command* object while the latter, managed by the server side, contains the required *Data* items.

An *Action* intended to be sent through the network, is first managed by the *toolkit*. Therefore, means allowing it to identify the application should be added. This is achieved by encapsulating the *Action* in a *Request* object which is managed at the *toolkit* level.

Applications only deal with the *Command* class. Other classes are managed by the *framework* and *toolkit*. Methods in the *Command* classes for the *Caching Service* are abstract and their implementation is a simple task for application developers as they deal with objects and procedures that are strictly application dependent such as fetching a row in a database. Mobility dependent decisions are confined in other MolèNE *services*. The following sections show how the *framework* is tailored to be used by two different applications.

5.2 Data Caching in a Transactional Internet System

METIS [1] (Mobile Environment for Transactional Internet Systems) is a middleware designed to allow Internet applications to execute on a mobile computer. Its main goal is to allow a user to utilize an internet transactional application while disconnected from the

wireless network. There are four main functionalities implemented by METIS: 1) loading the application's code into the mobile computer; 2) caching a subset of the database on the mobile computer; 3) processing the transactions on the mobile computer using the cached data; and 4) reintegrating the transactions on the database server. The first version of METIS has been implemented in an ad hoc manner. However, all its functionalities are now services of MolèNE and thus they can be tailored to be used by METIS this giving to it a new structure, more adaptable. We concentrate in this paragraph on the customization of the caching functionality.

Row	
int size	// size in bytes of the row
string location	// database's name + table's name + the primary key of the row
byte[] content	// value of each field of the row
void put()	// INSERT SQL query
void delete()	// DELETE SQL query

LoadRows	
string what	// SQL SELECT query
row[] load()	// 1. apply the what query to the database server; // 2. create a Row object for each row of the result

Figure 9: *Data* and *LoadCmd* subclasses in METIS

METIS uses the row of a table as the cached data granularity. Therefore, we create the *Row* class that inherits from *Data* and implement its methods. The *Load Rows* class is also implemented that inherits from *LoadCmd* class. The attributes and methods of these classes are shown in figure 9.

Let us consider the example of an application wanting to load all the sport articles from a virtual mall. The application creates a *LoadRows* object and fills its `what` attribute with the SELECT SQL query that corresponds to its requirements. Then, the `load()` method of the *Caching Service* is called and it waits until the *Data* arrive.

```
cmd = new LoadRows();
cmd.what = "SELECT * FROM shop WHERE type='sports'";
result = cachingService.load(cmd);
result.getReply();
```

All network communication aspects are transparent at this level. In the next paragraph we will show that it is even possible to build higher level operations.

5.3 Data Caching in a File System

MFS [2] (MolèNE File System) is a middleware built to allow the NFS file system [16] to be used by mobile users. Standard NFS clients utilize the main memory of their workstation as the cache repository and are responsible for data consistency. Therefore, while connected, a lot of data validity requests are sent to the servers to verify consistency of the cached data. In a mobile environment this is a battery consuming operation and, even worse, while disconnected, users are unable to work. MFS utilizes different services adapted to mobility. It caches frequently-accessed data on the mobile computer and performs operations on the mobile computer using them. Moreover, data consistency is ensured by reintegrating operations on the NFS server and updating cached data when they are modified on the server (a *"middleprocess"* executes on the fixed network so that no requests for data verification are sent by the mobile client). We concentrate on the caching functionality of MFS.

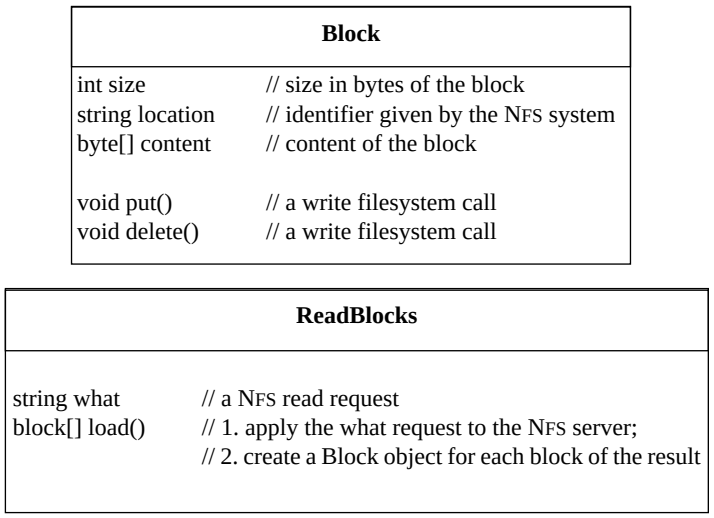

Figure 10: *Data* and *ReadCmd* subclasses in MFS

NFS uses blocks of a well-known size to manage information stored in servers. Therefore, MFS should create the class *Block* as a subclass of *Data* and implement its methods (see figure 10). A *LoadBlocks* class is provided that allows the loading of *Blocks* in the same way the *LoadRows* class allows the loading of *Rows* of a database. Finally, a *ReadBlocks* class that inherits from the *ReadCmd* one is provided. When a user asks for the access to a file, NFS generates a read request that is managed by MFS by creating a *ReadBlocks* object and

filling its `what` attribute with the NFS request. Then, the `read()` method of the *Caching Service* is called.

```
cmd = new ReadBlocks();
cmd.what = "NFS read request";
result = cachingService.read(cmd);
result.getReply();
```

This code does not tackle with network availability and is even more transparent from any caching mechanism than using the *LoadRows* object directly by MFS as it has been described for METIS. It is the `read()` method which is responsible for creating the *LoadBlocks* object to load the required *Data* if they are not available in the cache.

6 Related Work

A number of proposals provide applications with ready-to-use solutions for a particular issue. Adapting network traffic to the mobile environment characteristics is one of these issues. Most of the approaches propose an architecture in which a process, called a proxy, executes between the client and the server side of an application and manages traffic before the wireless link. The proxy can act at the network protocols as the filters in [24] or at the data level (images, text or video) as in [6]. Odyssey [17] does not utilize a proxy but extends the file system interface to allow applications to specify the data access strategy to be performed for a resources availability interval. When the availability goes out this interval, applications are notified and decide about a new strategy. MolèNE provides the *DataCompression microService* that carries out similar tasks depending on available resources and applications needs. In contrast to existing approaches, our *microService* is confined to the *framework* and thus, application functionalities are isolated from its management.

Another issue concerns hoarding of data. Systems that address this issue predict which are the data that will be accessed during disconnection so that the cache of the mobile computer is loaded with them. Mobile users are thus allowed to work while disconnected. This can be done transparently to the user as Seer [13] that analyses the current user behavior and establishes relationships between various files. The cache is filled with the groups of files in which the user worked recently when a disconnection is imminent. Other systems utilize the help of the user to decide which data to cache. In Coda [12] for example, users can determine which files they will use during disconnection. MolèNE provides the same services. Further, different strategies may be used by an application.

Data consistency is frequently tackled by mobile systems. Proposed solutions are based on optimistic replication of data that allows users to modify a locally available copy of the data while disconnected. Therefore, inconsistencies may appear and data consistency management systems aim at detecting and resolving them. In the data bases domain, Bayou [5] assumes that applications know the type of consistency provided and allows them

to define their notion of conflict and resolution methods. In Ficus [18] a file consistency management system, a subset of the conflicts are automatically detected but resolution is applications responsibility. The abstraction of the units of information provided by MolèNE makes it independent of the type of data used by applications. With a little and simple help of programmers, MolèNE is able to manage different types of systems.

How to monitor the environment conditions and notify applications about important changes is addressed by a number of systems. In [23] changes in the environment are modeled as asynchronous events that encapsulate information related to the environmental change it represents. Events are delivered to applications by means of events channels which implement the event delivery mechanism and are usually dedicated to one component of the mobile environment (e.g. a power channel may be associated with events related to changes in power supply). The environment servers proposed in [15], contains a database that manages environmental information. This information is represented as objects containing pairs of attribute name and value that define the environment. The server interface provides applications with means to register to a particular change and to get state of the environment. MolèNE provides the *Resources Monitors* that perform similar tasks. Once more, this code is confined to the *framework* and thus, application functionalities are isolated from it.

Other systems provide mechanisms that allow an application to implement its own solutions. Rover [10] provides relocatable dynamic objects (RDOs) and queued RPC. The first are cached by clients and their consistency should be ensured by applications. The latter allows applications to communicate with remote RDOs without blocking them while disconnected. These mechanisms are low-level ones thus, obliging applications to provide their own solutions to mobility problems. Code reuse is hardly possible.

Adaptation to changing conditions is tackled by Ensemble [22] by using a set of interchangeable modules (micro-protocols). Applications explicitly characterize the conditions under which adaptation is required and the system reserves resources to meet the demands. Violations are reported to the application which adjusts its requirements, leading to a reconfiguration of the micro-protocols. In [11] the open implementation concept [14] is used. Designers of components open up the implementation of their component so that it can be adjusted to different conditions. In contrast to Ensemble, components are fixed entities and it is their implementation that changes over time. None of these approaches are sufficient in high variable environments such as the mobile ones. MolèNE exploits both ideas, the first one at the *service* level and the second one in *microServices* implementation leading to a highly adaptable system well adapted to the mobile universe.

7 Conclusion

In this paper, we have presented MolèNE, a system that provides applications with a set of generic services to ease adaptation of applications to mobile environments. Its architecture maximizes the reuse of services by considering their generic nature. The MolèNE *toolkit* encapsulates *tools* such as the *Communication Tool*, which are independent of the applications that use them and are shared by applications. The *framework* provides *services* as the

abstraction of concepts usually utilized in existing mobile applications such as caching or consistency of data. *Services* in the *framework* are not shared but reused by applications and constitute the interface between applications and the *toolkit*.

MolèNE is designed to allow easy modification of existing *services* by providing the *Service*, *MicroService* and *Implementation* classes. Extensibility of MolèNE is achieved by encapsulating adaptation information in these classes so that new services and implementations of existing ones can be easily taken into account by MolèNE. Further, the bridge design pattern applied between the objects of the *framework* makes MolèNE a highly flexible system unique in the mobile computing domain.

As an example, we have presented the *Caching Service* in MolèNE and its customization to an electronic commerce application (METIS) and a file system (MFS). A transactional application in the medical domain is also being considered to be adapted to a mobile environment. This application allows medical stuff to work directly on the patients folders while doing their daily visit in the patients rooms. All these experiments allow us to validate the flexibility and extensibility of MolèNE. We are currently working on a complete implementation of MolèNE in Java. Our initial experience shows interesting results concerning already available services, and demonstrate MolèNE as a powerful alternative for building mobile applications. Further research consists of extending MolèNE with new services and defining new adaptation strategies for them.

References

[1] André, F., and Saint Pol E. A Middleware for Transactional Internet Aplications on Mobile Networks. In: *Proc. of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas Hilton, Las Vegas, July 1998.

[2] André, F., and Segarra M.T. On Building a File System for Mobile Environments Using Generic Services. In: *Proc. of the 12th Parallel and Distributed Computing Systems (PDCS)*, Fort Lauderdale, Florida, August 1999.

[3] Bakre A., and Badrinath B.R. I-TCP: Indirect TCP for Mobile Hosts. In: *Proc. of the 15th International Conference on Distributed Computing Systems (ICDCS)*, Vancouver, British Columbia, Canada, May 1995.

[4] Chandran K., Raghunathan S., Venkatesan S., and Prakash R. A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks. In: *Proc. of the 18th International Conference on Distributed Computing Systems (ICDCS)*, Amsterdam, The Netherlands, May 1998.

[5] Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and Welch, B. The Bayou Architecture: Support for Data Sharing among Mobile Users. In: *Proc. of the Workshop on Mobile Computing Systems and Applications (WMCSA)*, Sta. Cruz, CA, December 1994.

[6] Fox A., Gribble S.D., Brewer E.A., and Amir E. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In: *Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, Massachusets, 1996.

[7] Gamma E., Helm R., Johnson R., and Vlissides J. Design Patterns Elements of Reusable Object-Oriented Software. *Addison Wesley*, 1995.

[8] Howard J.H., Kazar M.L., Menees S.G., Nichols D.A., Satyanarayanan M., Sidebotham R.N., and West M.J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems* 6, 1, February 1988.

[9] Imielinski T., and Korth H.F. Mobile Computing. ed. by Tomasz Imielinski and Hank Korth, *Kluwer Academis Publishers*, 1996.

[10] Joseph, Antony D., Tauber, Joshua A., and Kaashoek, M. Frans. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers* 46, 3, 337–352, March 1997.

[11] Kiczales G. Beyond the Black Blox: Open Implementation. *IEEE Software*, January 1996.

[12] Kistler, James J., and Satyanarayanan M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems* 10, 1, 3–25, February 1992.

[13] Kuenning, Geoffrey H., and Popek, Gerald J. Automated Hoarding for Mobile Computers. In: *Proc. of the 16th Symposium on Operating Systems Principles (SOSP)*, St. Malo, France, October 1997.

[14] McIlhagga M., Light A., and Wakeman I. Towards a Design Methodology for Adaptive Applications. In: *Proc. of the 4th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Dallas, Texas, October 1998.

[15] Nakajima T., Aizu H., Kobayashi M., and Shimamoto K. Environment Server: A System Support for Adaptive Distributed Applications. In: *Proc. of the 2nd International Conference on Worldwide Computing and its Applications (WWCA)*, March 1998.

[16] Network Working Group. RFC 1094: NFS 2.0 Protocol Specification. *Sun Microsystems, Inc.*, March 1989. Available at: http://www.cis.ohio-state.edu/rfc/rfc1094.txt.

[17] Noble, Brain D., Satyanarayanan, M., Narayanan, D., Tilton, James E., Flinn, J., and Walker, Kevin R. Agile Application-Aware Adaptation for Mobility. In: *Proc. of the 16th Symposium on Operating Systems Principles (SOSP)*, St. Malo, France, October 1997.

[18] Page T.W., Guy R.G., Heidemann J.S., Ratner D.H., Reiher P.L., Goel A., Kuenning G.H., and Popek G.J. Perspectives on Optimistically Replicated, Peer-to-Peer Filing. *Software-Practice and Experience* 28, 2, 155–180, February 1998.

[19] Perkins C.E. Mobile Networking Through Mobile IP. *IEEE Internet Computing* 2, 1, 1998.

[20] Srinivasan M., Chellapa R., and Bukrlina P. Adaptive Source-Channel Subband Video Coding for Wireless Channels. *IEEE Signal Processing Society 1997.* Workshop on Multimedia Signal Processing (MMSP), Princeton, New Jersey, June 1997.

[21] UML version 1.1 specification. *http://www.rational.com/uml/resources/documentation/index.jtmpl.*

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

[23] Welling G., and Badrinath B.R. An Architecture for Exporting Environment Awareness to Mobile Computing Applications. *IEEE Transactions on Software Engineering* 24, 5, May 1998.

[24] Zenel B., and Duchamp D. A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment. In: *Proc. of the 3rd Annual International Conference on Mobile Computing and Networking (MobiCom)*, Budapest, Hungary, September 1997.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399