A Generic Approach to Satisfy Adaptability Needs in Mobile Environments

F. André and M.T. Segarra
IRISA Research Institute
Campus de Beaulieu
35042 Rennes Cedex
France
{fandre, segarra}@irisa.fr

Abstract

The need for redesigning existing “data-intensive” applications to execute in mobile environments has led to an enormous proliferation of systems that propose a solution to different issues. In this paper, we classify these approaches and give evidence of the lack for code reuse efforts. We introduce the Molène system as a set of generic services that aims at providing a systematic approach to build mobile applications. Depending on the generic nature of the provided services, Molène architecture is divided into a toolkit and a framework layer. The former provides device management related services. Services of the framework are application-dependent and their customization generates an instance of Molène that is well suited to the targeted application. Further, services design allows to easily modify their behavior depending on the application needs.

1. Introduction

The existing telecommunications infrastructure was initially utilized to wiredly access centralized mainframe processors, then, applications and data distributed among desktop computers. However, wired accesses fail to provide the ability to compute from anywhere and anytime as users should have physical access to the computers wired to the network. Recent technology advances have enabled, on the one hand, the development of portable computers from laptops to personal digital assistants (PDAs) and, on the other hand, the emergence of wireless networks ([19], [11]). These advances lead to the wireless computing paradigm that offers the necessary flexibility to accomplish universal access.

As wireless computing allows users to perform computations while on the move, many new technical challenges should be addressed to perform an efficient wireless computing. This challenges are related to the characteristics of the wireless medium, the portable devices resources limitations and the mobility itself. The impact of all this on the system goes beyond the networking level and also affects the application software itself. Although some protocols at the network level have been introduced that alleviate or even hide the effect of mobility, changing the way applications execute in a wireless environment is equally important. For example, mobile devices depend on a limited energy supply, a battery. As energy consumption depends on the activity of devices such as the display or the network interface, applications may adapt their behavior to reduce energy consumption. Therefore, the redesign and reimplementation at least partially of the applications should be envisaged.

Research on developing new ways of building wireless applications lack for providing a generic approach that is independent of (yet customizable to) different applications and their semantics. Such approach would help developers in their task and would facilitate code reuse. The Molène project aims at building such system. We claim that its required generic nature can only be achieved by abstracting concepts frequently used by previous approaches dealing with data access issues on wireless environments and encapsulating them in generic and customizable services. Some of the services execute on the portable device, others on the fixed network, the cooperation between them being precisely defined in our system. The Molène architecture is derived from the different generic nature of the available services: some of them are general enough to be application-independent while others need to be customized for each applications. Further, the need for adaptation to changing environmental conditions has been recognized as an important feature of wireless clients. Therefore, Molène provides means for environmental adaptation by: 1) allowing applications to express their resources needs for each of the utilized services, 2) tracking the environment to detect changing conditions, and 3) designing services to gracefully adapt to changes.

The reminder of the paper is organized as follows. Sec-
tion 2 points out adaptation as a need at the application level. By showing the characteristics of wireless environments, their highly variable nature is evidenced and implications in the application design are presented. Section 3 describes advantages and weaknesses of different models used by current approaches when extending existing distributed applications to make them aware of their wirelessness. Section 4 introduces the main properties that guided the design of Molène. Section 5 presents the architecture of Molène by discussing its key services and their customization and adaptation capabilities. Finally, the paper concludes in section 6.

2. The Need for Adaptation

2.1. Portable Computers

Making computers portable implies striving for properties such as small, light, and requiring minimal power usage for long battery life. Managing battery life to sustain computations is presented in [11] as an important feature. Minimizing power consumption can improve portability by reducing battery weight and lengthening the life of a charge. Power can be conserved at the hardware level by properly designing components and, at the software level, by providing energy-efficient systems that switch off individual components when they are idle [6]. As battery life depends on the utilization of resources on the mobile host, applications can also contribute to the conservation of energy by adapting their requirements on communications, computations or memory [7] depending on battery life.

The small size and weight of a portable computer mean restricted memory size, small storage capacity, restricted computing power, and small user interface. Different types of portable devices exist from PDAs to laptop computers, each having made different concessions in resources availability and thus, in the degree of portability. Applications should take into account resources limitations of portable units by being more flexible when a required resource is unavailable. Generating an error message to the user immediately after detecting the unavailability of the resource, is not a satisfying solution in this kind of restrictive environments. Moreover, current technologies allow to insert/remove peripheral devices (PCMCIA) in portable computers while executing applications, thus dynamically changing the configuration of the portable unit. Therefore, applications should be designed to be able to adapt to such changes by modifying their resources requirements depending on availability.

Finally, making computers portable increases the risk of accidental damage, theft or loss, arguing for more dependency on “tethered” hosts.

2.2. Wireless Networks

Wireless networks face more obstacles than wired networks because the surrounding environment interacts with the signal, blocking signal paths and introducing noise. As a result, wireless links are characterized by low bandwidths, high bandwidth variability, and frequent spurious disconnections.

Wireless networks deliver lower bandwidth than wired networks: while Ethernet provides 10 Mbps, and fast Ethernet and FDDI 100 Mbps, wireless technologies achieve only 1 Mbps for infrared communication, 2Mbps for wireless LANs, and 19 Kbps for cellular telephony (GSM). Further, network bandwidth is divided among the terminals on a cell, and thus the data rate dedicated per user is even lower.

Also, bandwidth variability suffered by mobile devices is greater than that suffered by static elements. Bandwidth can shift one to four orders of magnitude, depending on whether the system is plugged in or using wireless access. Moreover, a portable device can communicate using different network interfaces as it moves. Moving from indoors to outdoors forces a mobile device to change from a wireless LAN providing 1Mbps to another network such as a cellular one providing 19 Kbps. Even staying outdoors may require the portable computer to change a cellular coverage in city and a satellite coverage in the country. In order to avoid the user suffering from such variations, applications should adapt their behavior by degrading or upgrading information traveling through the communication links.

Wireless connections, are significantly less reliable than wireline connections: the mobile unit may go out-of-range or, behind a barrier that blocks the signal. Therefore, disconnections in mobile environments are much more frequent than in traditional distributed ones. Disruptions may be hidden from the wireless applications by error recovery procedures on either of the communications layers. However, this often results in excessive retransmissions.

2.3. Mobility

When moving, portable devices change their point of attachment to the fixed network. This dynamic nature in location, introduces several issues related to computer network addresses and location-dependent information. Today’s networking is not designed for dynamically changing addresses: addresses of hosts are fixed and used to route traffic through the network. Therefore, network protocols are ill-suited for mobile nodes. Mobile environments require dynamic mechanisms enabling the tracking of mobile units in the communication infrastructure to allow fixed stations to communicate with mobile units.

Mobility suggests a new class of applications called context-aware computing [21] that are aware of the con-
text in which they are run, based on a limited amount of information covering user’s proximate environment. Location information enables software to adapt according to its location of use, the collection of nearby objects, as well as the changes to those objects over time. As indicated in [21], changing the location may imply going away from the servers currently in use and becoming closer of new ones. Applications should dynamically transfer service connections to closer servers. Finally, context-aware applications such as [1] take advantage of location information to improve the information presented to the user when visiting an unfamiliar place or rapidly locating users in an office environment.

3. Wireless Computing Approaches

The previous section has pointed out the need for applications to adapt their behavior to changes. Some work has already been performed to extend existing applications to make them aware about variability of the environment. In this section we classify these approaches by identifying the elements that make up the architecture of the proposed solution.

3.1. The Middle-Process Model

In this model, an intermediary or middle-process provides some functionality to the applications running on the mobile device in order to manage wireless issues. Existing approaches using this model consider different places to execute the middle-process and allow it to provide its functionality at different levels. Depending on both parameters different opportunities are provided to the applications that utilize the proposed middle-process. In all cases, servers are not concerned with the special characteristics of mobile users. All data transformation and customization for mobile clients may be handled by the middle-process thus offloading significant computational burden from the server.

Three levels are consider for the middle-process to provide its functionality. At the highest level, one middle-process provides functionality to all the applications running on the mobile computer [17]. These are coarse-grained middle-processes as the same adaptation policy is applied to all applications. A middle-process may also be attached to a specific service such as Web browsing [8] or databases [15]. In this case, using different services implies the deployment of different middle-processes but adaptation may be more suitable to applications. Finally, the finest-grained middle-processes are those attached to a specific application [24] which allows to provide an adaptation policy tailored to each application but may imply the deployment of much more infrastructure.

![Figure 1. The middle-process model.](image-url)

The second parameter in this model concerns the position for the execution of the middle-process. A number of approaches propose the fixed network to execute the middle-process [4, 24, 8]. This approach is depicted in figure 1(a). The exact position of the middle-process at the fixed network depends on its role. Placing it at the fringe of the fixed network makes it easier to gather information about wireless link characteristics. Moreover, special network protocols can be used between the mobile host and the middle-process to take into account wireless networks characteristics [24].

Placing the middle-process on the fixed network allows for light-weight devices of limited computational power and resources. The middle-process can carry out computations on behalf of the mobile host which 1) reduces the amount of code on the mobile device allowing it to host an application that would otherwise be too large, and 2) preserves battery life as resources on the mobile device are not used. However, as no support is provided at the mobile computer, approaches following this option fail to sustain computations at the mobile device during periods of disconnection.

Other approaches consider the placement of the middle-process in the mobile computer (see figure 1(b)). Most of them [14, 20] intend to allow mobile users to work while disconnected. In order to achieve this, they assume a full client being executed on the mobile device and provide a middle-process with enough functionality to act as a thin server that serves requests coming from local clients. This option is more appropriate for heavy-weight devices with enough computational power and secondary storage to support the middle-process.

In all cases, the functionality provided by the middle-process highly depends on the specific application or ser-
vice which the middle-process is attached to. However, the basic functionality concerns the queuing of traffic to manage disconnections and weak connectivity. Depending on the position of the middle-process, traffic going to or coming from the mobile device is queued by the middle-process to be sent when reconnection.

A more advanced role allows the mobile device to submit computations to a middle-process executing on the fixed network and wait for the results [22]. This may be useful to manage disconnections as the mobile unit can be disconnected while computations are performed on the middle-process and to preserve battery life.

Weak connectivity is another issue that can be addressed by this model. The middle-process can manipulate data prior to their transmission to or from the mobile device to optimize the utilization of the wireless medium. This can be performed 1) by applying lossy compression on data [8, 18] or by filtering information embedded on protocols [24] to reduce the amount of traffic on the wireless link; 2) by re-ordering message delivery to place important data ahead of less important data [14]; and/or 3) by batching requests [4].

Managing weak connectivity is also achieved by caching and prefetching data on (or near) the mobile computer. Indeed, some middle-processes store frequently used data and are responsible for maintaining their consistency which can be done with [5] or without [17] applications assistance. On the other hand, periods of good or cheap connection [20] can be utilized to store in the mobile computer cache data intended to be used in the near future. Again, the help of the user may be required to provide the middle-process the notion of good and cheap connections.

### 3.2. The Intercept Model

The key elements of the approaches following this model are depicted in figure 2. Two middle-processes are virtually inserted in the data path coming from and going to the mobile device: one on the mobile device and another within the wireline network. As in the previous model, both middle-processes may provide functionality at different levels which translates in different amount of required infrastructure and knowledge of application semantics.

![Figure 2. The intercept model.](image)

The mobile-device middle-process appears as a local server for applications on the mobile unit while the fixed-network middle-process appears as a client component that resides on the fixed network. Cooperation between both elements allows to perform optimizations to reduce data transmission over the wireless link, improve data availability and sustain uninterrupted the mobile computation. Therefore, this model can be viewed as a client/server one in which the pair of middle-processes minimize the effects of the wireless network.

Disconnections and weak connectivity are easily handled by this model. Disconnections can be managed by caching frequently accessed data on the mobile computer [10]. Moreover, cache misses may be queued by the mobile-device middle-process to be served upon reconnection. Similarly, requests to the mobile unit can be queued at the fixed-network middle-process and transferred to the mobile device upon reconnection.

Support for weak connectivity can be provided by the mobile-computer middle-process by performing background prefetching during periods of strong connectivity [16]. The fixed-network middle-process, on the other hand, can filter or compress data prior to their transmission to the mobile device as in the middle-process model.

The intercept model provides for a clear distinction and splitting of responsibilities between the mobile-device and the fixed-network middle-processes and, at the same time, it provides for high degree of collaboration between the pair of middle-processes. All operations related with the management of wireless issues are handled by both processes thus shielding applications from considering them on their own code. This model is more appropriate for heavy-weight mobile devices with enough computational power and secondary storage to support the mobile-computer middle-process.

### 3.3. The Dynamic Model

The extension of existing applications proposed in the previous models, implies to statically split the functionality between the mobile device and the fixed network. Although they provide means for some type of adaptation, approaches following these models are not flexible enough to provide a good solution for the highly variable wireless environments. The dynamic model constitutes a further step towards the construction of flexible enough software for such environments. In this model, functionality intended to manage wireless issues can be dynamically moved from the mobile device to the fixed network and vice versa depending on environmental conditions.

The dynamic model may be combined with the previous paradigms. One can have a middle-process model such as in the DIANA approach [13] in which the middle-process executes on the fixed network while connected and moves to the mobile device prior to disconnections allowing for sustaining mobile computations. On the other hand, the intercept model can run on top of the dynamic model with
the mobile-device and fixed-network middle-processes utilizing their dynamic nature to perform different tasks. This model provides the most flexible approach to wireless computing since it allows for dynamically splitting the workload between both middle-processes based on the various environmental conditions.

Further, more advanced solutions to manage disconnections can be provided by approaches following the dynamic intercept model. Disconnections can be managed by both, moving functionality to and from the mobile-device middle-process. Extending the functionality of the latter allows the mobile user to work while disconnected. Conversely, extending the functionality of the fixed-network middle-process allows for computations to proceed even during disconnections.

Also, middle-processes in the dynamic intercept paradigm can be responsible for relocating tasks based on application preferences and environmental conditions, thus shielding applications from the details of adaptability and awareness.

Due to its inherent adaptability, the dynamic intercept model has lately been payed attention [12, 23]. However, in all cases, functionality to be relocated should be provided by applications which still constitutes a great burden for application developers.

### 4. MolèNE Characteristics

The dynamic intercept model has been presented as the most suitable way to build highly adaptable wireless applications. However, developing such a model may require skilled programmers in order to design the mechanisms for adaptation. Considering this model to build each application may lead to a great programming effort and, what is very important, a poor software reuse. Moreover, if the adaptation mechanisms are not well isolated, they may obscure applications functionalities making them more difficult to develop and maintain. Therefore, using a middleware seems to be the most adequate approach to tackle with adaptation issues. The middleware might run between the applications and the storage system so that accesses to the data are adapted depending on applications requirements and environment conditions. Moreover, the middleware might be generic so that it could be used with all types of storage systems and could adapt its behavior to all applications using the storage system.

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 comes from the abstraction of functionalities previously identified as being common to the application domain covered by the framework. Abstractions translate into abstract classes, which ease the customization of the framework, while implementations translate into concrete classes which facilitate code reuse. 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 abstract methods of the framework. On the other hand, inheritance and polymorphism mechanisms1 of object-oriented systems are greatly used by frameworks to manage abstract and concrete classes adequately.

The ease of customization of object-oriented frameworks and the adequacy of a middleware to tackle with adaptation issues, have led us to design MolèNE as an object-oriented framework to build middlewares for data management that use the dynamic intercept model to achieve adaptation. MolèNE offers a set of services as a generalization of concepts that are usually utilized by existing approaches dealing with data management on wireless environments. Such services include caching of data on the mobile computer, data consistency management, and data filtering/compression. Services are generic so that they can be customized for different storage systems². Moreover, their functionality is dynamically distributed between the mobile computer and a fixed station so that their behavior can be adapted to environment conditions depending on applications requirements.

#### 4.1. Customization of MolèNE

The application domain covered by MolèNE concerns data management systems on wireless environments. In these systems, data items are stored, removed, read or written from a data repository depending on applications requirements. Different storage systems may provide a different interface to applications and may use different units of information (e.g. blocks of files for a filesystem, rows for a database management system). Customizable MolèNE services are designed so that a middleware developer is able to provide enough information to allow MolèNE to manage this heterogeneity. As an example, the management of the data items heterogeneity is achieved by providing the abstract class Data which contains meta-information concerning the units of information managed by the storage system. Each of the objects of this class represents one of the units of information managed by the particular storage system. The attributes of this class concern information such as the loading time, last read time, size of the data, etc. The methods allow the particular data item to be stored, removed, read or written from the data repository and thus are abstract methods.

---

1 Inheritance is a mechanism enabled when one class is defined in terms of another class in an "is kind of" relationship. Polymorphism allows a concrete subclass to be used wherever the abstract superclass is expected.

2 We will refer as MolèNE middleware, an instance of the proposed framework for a particular storage system.
Although section 5.2 will detail the customization process of MolèNE, we give here an example of customization of the Data class. Consider the case of a particular database that works on rows of tables as the unit of information. In this case, the middleware developer for the database, creates a class Row that inherits from Data and implements its methods. These methods correspond to simple SQL requests (insert, delete, select and update) and allow MolèNE to carry out other tasks much more complicated such as the management of the contents of the mobile computer cache.

Figure 3. The two-level strategy design pattern for the DataFormat Service.

4.2. Adaptation in MolèNE

Another goal of MolèNE is to provide means for adaptation to changing conditions. In order to ease this adaptation, we have identified three types of objects in MolèNE that implement the provided adaptive services. Each of these services is represented by a Service object which role is divided into subtasks, referred in the paper as microServices, that are 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 strategy design pattern (see figure 3). This pattern “define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it” [9]. By encapsulating in an object implementations for a particular abstraction and establishing a composition between the implementation and the abstraction, the modification of the latter becomes simple. This pattern is applied in MolèNE at two levels. At the service level, the structure of a service can be easily changed by changing the composing microServices. At the microService level, a composition is established between MicroService and Implementation objects so that it is possible to dynamically change the implementation of the former.

In order to show the high degree of flexibility of MolèNE, let us consider the example of the adaptation of the data traveling through the network. This is accomplished by the DataFormatService 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 goal of this service is to adapt the data coming from or going to the network interface depending on the characteristics of the execution environment such as network bandwidth and mobile computer display.

As shown in the figure, two microServices can be configured to be used by this service: a data compression microService and a data scaling one. The former is performed by the DataCompression object. When the system decides that the data should be formatted in order to reduce bandwidth consumption, this microService is utilized by MolèNE. Moreover, different compression algorithms exist each consuming resources in a different way and offering different quality of the data. Therefore, the utilized compression method can be chosen depending on its resource consumption and application preferences.

On the other hand, 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 object. Depending on the resources availability on the mobile computer, it may be executed either on a fixed station or on the mobile computer thus changing its location.

Therefore, adaptation in MolèNE can be achieve by changing the implementation and/or location of a particular MicroService and by changing the structure of a service by adding or removing its composing MicroService objects. Section 5.2 will show in detail the way MolèNE carries out these types of adaptation.

5. The MolèNE Architecture

In order to maximize the reuse of components and to ease the adaptation, the MolèNE architecture is divided into two levels as depicted in figure 4. The MolèNE toolkit contains services which need not to be customized for a particular application, for the supporting architecture (storage system, external devices) nor the environment (user location). They are executed at the computer level thus, being shared by all applications, and will be referred as tools in the paper.

Figure 4. The MolèNE architecture.

The second layer, the MolèNE framework, offers services which functionality should be customized for a particular application and its environment. These will be shared
by applications utilizing a MolèNE middleware and will be reused when developing new instances of the MolèNE framework. They use the tools to perform their tasks and will be referred as services in the paper. The next sections present in detail each of the components of the architecture.

5.1. The MolèNE Toolkit

The main tools available in MolèNE are depicted in figure 5. All of them are related to the adaptation process. The functionality of the Migration Tool is utilized when computations coming from remote sites are requested to be executed locally. This tool verifies execution permissions and is used by the MolèNE framework when the adaptation requires changing the location of a microService.

A final application, i.e. one that uses the middleware built using the MolèNE framework, initiates a registration process at the beginning of its execution. This process is intended, among other things, to give the Global Adaptor some information concerning the application and its adaptation strategy. Information about the application includes a unique identifier (usually a simple name given by the programmer of the application) and a priority (usually given by the user of the application). This information is used by the Global Adaptor to decide about which application should adapt first in case of conflict.

![Figure 5. Services and tools of MolèNE.](image)

The adaptation strategy is given to the Global Adaptor as a set of boolean conditions on existing resources monitors. This information is passed to the Detection/Notification Tool that will notify the Global Adaptor when one of the conditions is verified. In this case, the Global Adaptor decides if the application should adapt or not and if so, informs the MolèNE middleware used by the application about the condition of adaptation and the identity of the application to adapt.

5.2. The MolèNE Framework

Several services have already been identified in the MolèNE framework and are depicted in figure 5. Two types of relations are shown in the figure. Data relations correspond to the usual cooperation among services and tools to manage data items while control relations refer to operations intended to configure the MolèNE framework and can be considered as meta-operations.

Available Services and their Customization The Networking Service establishes a reliable communication channel between the middleware on the mobile computer and that on the fixed station so that services are able to send/receive requests to/from the network interface despite disconnections. In order to achieve this, one of the microServices associated with the Networking Service is responsible for storing in a generic queue, that we call a Log, requests that have already been sent but that have not been acknowledged. Therefore, requests that have been sent while disconnected, are stored in the Log and are re-sent to their destination upon reconnection.

Information sent by each of the services in MolèNE should be managed by the corresponding service in the destination middleware. For example, a client-side Caching Service is able to ask for some data to be loaded into the cache to a server-side middleware. This “caching information” should be managed by the Caching Service in the destination middleware in order to ensure correctness of the treatment. In order to achieve this, the MolèNE framework offers a set of generic data types. These generic classes and their relation with the specific ones for the Caching Service are shown in figure 6. The Action generic class is subclassed by each of the services. The Caching Service for example, manages two types of Action objects: 1) LoadRequest objects that encapsulate the name of the element containing the units of information to be loaded into the client cache and the load request to be submitted to the storage system; 2) LoadReply objects that encapsulate the requested data.

An Action object is encapsulated in a Request\(^3\) by the Networking Service when sending it through the network and is extracted from it when received. The extracted Action object is delivered to a special service, called Dispatch Service, that performs a cast on the type of the Action and delivers it to the corresponding service.

Three services are available in MolèNE which goal is the management of data items on the storage system: the

---

\(^3\) A Request object contains useful information to deliver the information to its destination. This information includes a unique identifier for the request.
The set of operations final applications can perform on data should be the same as the one provided by the original data management system. As different storage systems offer a different set of operations, Molène should provide the middleware programmer with means for defining this set for his particular storage system.

The type of the accesses performed by final applications should be known by Molène so that it is able to correctly manage cache contents and data consistency. Four types of operations are identified in Molène: read accesses, update accesses, create accesses, and delete accesses. Depending on the type of the operation Molène carries out different actions. A read access for example may interest the Caching Service while an update one may interest the Consistency Service. As Molène is not able to establish the relation between an operation and its nature, means should be provided to the middleware programmer to establish it.

The Adaptation Meta-Service Adaptation can be considered as a meta-service in Molène as it acts on the framework components to modify their structure. This meta-service is provided as a set of cooperating objects that take adaptation decisions depending on their knowledge of the framework structure. The Global Adaptor at the toolkit level decides which application to adapt as discussed in section 5.1. Adaptation decisions taken by this tool are transferred to the Local Adaptor object in the framework as a set of conditions indicating the reason for the adaptation (e.g. "TheAccessType Method class nor its subclasses contain no attribute nor methods. They are utilized by Molène to obtain the type of an operation."

---

**Figure 6.** Data types involved when loading some information into a client cache.

**Figure 7.** Classes involved in the generation of the DataAccess Service.
scarce bandwidth, scarce memory or, on the contrary, new available memory).

The Local Adaptor object decides about modifications of services behavior when the Global Adaptor indicates a need for adaptation. In order to achieve this, the Local Adaptor keeps two graphs of Service, microService, and Implementation objects (see section 4.2). The first one represents the current structure of the utilized services and the second one concerns the possible organizations of the services. Thanks to the encapsulation properties of the classes composing the graphs, the Local Adaptor is able to know resources consumption of the services and, based on this knowledge, it decides about which type of adaptation should be performed on which of the services.

Figure 8 shows the Service, MicroService, and Implementation classes and their interaction at the adaptation meta level. Each of these classes encapsulates a specific Adapter object that provides the required functionality and information to adapt the class behavior. The Service Adaptor provides methods allowing to change the structure of the service and to change the location of one of the composing MicroService object. The MicroService Adaptor encapsulates information related to the adaptation at the microService level. This information includes the parameters that characterize its adaptation, and a method to change its implementation. The first type of information is used by the Local Adaptor to consider application preferences when performing adaptation decisions as not all the “types” of microServices can be used by all applications. The second one is called by the Local Adaptor when the implementation of the MicroService object should be changed.

Let us consider again the example of caching of data items. One of its composing MicroService objects is responsible for deleting data items from the client cache. This microService is characterized by two parameters: which and when to remove some data from the cache. A number of strategies are possible in order to decide which data should be removed. The most frequently used strategy considers the time of the last read of the data. However, other strategies may be considered in a wireless 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 former, by which the MicroService object chooses the data to be evicted when new data should be cached. In the second case, data are evicted from the cache when the amount of used memory exceeds a threshold.

The Implementation class contains information about the resources utilized by the concrete implementation of the microService and the value of the parameters for this implementation. Values for a traditional replacement strategy will be “LRU” for the first parameter and “on demand” for the second one.

Figure 8. The adaptation meta-service.

Providing these classes with the necessary information to perform the adaptation ensures MolèNE extensibility. Applications can create their own implementations for a particular microService by inheritance of the Implementation class. They only have to specify the resource consumption and the value of the characterizing parameters of the microService. As all the useful information is encapsulated in the class, new implementations may be easily taken into account by MolèNE.

6. Conclusion and Future Work

In this paper, we have presented the dynamic intercept model as the most suitable way to build highly adaptable wireless applications. Moreover, we have argued that to optimize code reuse and to reduce application developers task, this model should be applied on a middleware level. MolèNE is a system taking into account both criteria: it utilizes the dynamic intercept model and is designed as an adaptive middleware intended to provide applications with a quality of service (in terms of data availability) similar to that obtained on a fixed environment.

MolèNE offers a set of generic services that can be customized for a particular data management system and adapt to changing environment conditions. MolèNE architecture maximizes the reuse of services by considering their generic nature. The toolkit offers services which behavior is independent of the particular storage system and is shared by middlewares executing at a station level. The framework provides services that should be customized for the particular storage system and that can be adapted to environment conditions.

Adaptation in MolèNE is provided by the encapsulation properties of the Service, MicroService, and Implementation classes that made up the available services. Adaptation information is encapsulated in these classes so that extensibility of MolèNE is ensured and applications adaptation strategies can be taken into account. Further, the strategy design pattern applied between the objects of the framework makes MolèNE a highly flexible system unique in the wireless computing domain.

The development of two middlewares working on different domains has helped us to identify the abstractions and
services offered by Molène. The first one, MFS [3], allows the utilization of the NFS distributed filesystem on wireless contexts. The second one, METIS [2], concerns transactional applications. Both experiences have shown the interest of the services provided to manage data availability and, even more important, that it is possible, and desirable, to provide generic services that can be customized for the particular implementation.

Another experience is underway to validate already identified services and to identify new ones. The goal of the experience is to provide doctors with access to their patients related information on a mobile device while carrying out the daily visit to patients rooms. The mobile computer cache should be dynamically loaded with information concerning the nearest patients.

We are currently working on a complete implementation of Molène in Java. The Networking Service is already available and the Caching and Consistency services are being developed. They will be used to validate the Molène architecture.

References


