ABSTRACT

Mobile applications are required to operate in environments in which the availability for resources and services may change significantly during system operation. As a result, mobile applications need to be capable of adapting to these changes to offer the best possible level of service to their users. However, conventional middleware is limited in its capability of adapting to the environment changes and different users' requirements. Computational reflection applied to middleware design has introduced a new research field, reflective middleware. In this paper, we propose a reflective middleware architecture which can be used to develop adaptive mobile applications. In order to validate the architecture proposed, we developed a prototype using the Web Services technology which focuses on the problem of adapting on a set of attributes in a coordinated manner.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures - domain-specific architectures.

Keywords
Mobility, Adaptability, Middleware, Reflection, Web Services, Reflective Middleware.

1. INTRODUCTION

Recent advances in wireless networking technologies and the growing success of mobile computing devices are enabling new classes of applications which present new kinds of problems to designers. These mobile applications have to be aware and adapt to variations in the system's environment such as fluctuating network bandwidth, low battery power, slow CPU speed and low memory [1].

In the past decade middleware technologies, which reside between the operating system and the application, have enhanced the design and the implementation of distributed systems. Middleware hides from the programmer the complicated details about networking communications, remote method invocation and naming providing an easy platform to build complex distributed systems. However, conventional middleware have been designed for stationary distributed systems. It does not provide support for dealing with the new dynamic aspects in which mobile applications need to operate nowadays. The next applications generation require a middleware that can be adapted to changes in their execution context and customized to fit in many kinds of devices [7]. In this direction, researchers have proposed a next generation middleware that conforms to this aim. This next generation middleware should be run time configurable and allow inspection and adaptation of the underlying software. They propose an approach based on computational reflection and components technology which permits to create a middleware layer that can adapt itself to changes of the environment.

In this paper first is introduced the computational reflection concept (Section 2). Section 3 analyzes a set of requirements that future middleware should incorporate in its architecture to support adaptive mobile applications. Section 4 discusses several adaptation techniques which can be employed in mobile applications to support adaptability. Section 5 presents a reflective middleware architecture created to support adaptation in mobile applications. Section 6 discusses and evaluates the prototype which was developed to validate the architecture proposed. In the section 7 we present the conclusions and the future works.

2. REFLECTION

Reflection refers to the capability of a system to reason about and act upon itself [8]. More specifically, it is the ability of a system to watch its computation and possibly change the way it is performed. In this direction, a reflective system is one that provides a representation of its own behavior, which can be used to inspection and adaptation and is causally connected to the
underlying behavior it describes. Causally connected means that changes made to the self-representation are immediately mirrored in the underlying system’s actual state and behavior and vice-versa.

An object-oriented reflective system is logically structured in two or more levels, constituting a reflective tower. The first level is the base-level and describes the computations that the system is supposed to do. The second one is the meta-level and describes how to perform the previous computations. The entities (objects) working in the base-level are called base-entities, while the entities working in the other levels (meta-levels) are called meta-entities.

Each level is causally connected to adjacent levels, i.e., entities working into a level have data structures reifying the activities and the structures of the entities working into the underlying level and their actions are reflected into such data structures.

A reflective computation can be separated into two logical aspects: computational flow context switching and meta-behavior [8]. A computation starts with the computational flow in the base-level; when the base-entity begins an action, such action is trapped by the meta-entity and the computational flow rises at the meta-level (shift-up action). Then the meta-entity completes its meta-computation, and when it allows the base-entity to perform the action, the computational flow goes back to the base level (shift-down action).

3. ADAPTIVE ARCHITECTURE
REQUIREMENTS

Efstratiou et al. [4] suggests that there are limitations of current approaches for supporting adaptive applications. Specifically, these approaches lack of support for enabling applications to adapt to numerous different attributes in an efficient and coordinated way. So, he asserts that a new approach is required, which must provide a common space for the coordinated, system-wide interaction between adaptive applications and the complete set of attributes that could be used to trigger adaptation.

This new approach is based on a set of requirements that could be used to develop an appropriate architecture for supporting adaptive applications. The first key requirement of the architecture is to provide a common space for handling the adaptation attributes used by the system in which new attributes can be introduced as and when they become important. The second requirement is to be able to control adaptation behavior across all components involved in the interaction on a system-wide level. A further requirement is to support the notion of system-wide adaptation policies that should enable a system to operate differently given the current context and the requirements of the user. A final requirement arises from the fact that most mobile applications operate in a distributed environment, reason for what the adaptation mechanism need to coordinate all elements involved in the system distributed operation.

4. ADAPTATION STRATEGIES

Friday et al. [6] suggests that future mobile environments will require software to dynamically adapt to rapid and significant fluctuations in the communication link quality, which implies in the fact that software will have to include adaptation techniques in its design and implementation. In this direction Friday et al. [6] presents several adaptation techniques which can be triggered in all levels of an adaptive application, from system level to user level. In the middleware level three approaches can be identified. In the first approach, the middleware services can attempt to reduce application bandwidth requirements by using data compression techniques before transmission for example. The second approach is developing services which can prefetch and cache data during periods of bandwidth high availability in preparation to future bandwidth reduction or service disconnection. The third approach is to rebind clients to services available in the local context until network connectivity can be established.

Welch [9] asserts that the mobile computing paradigm brings new challenges in computer operating systems development. These challenges include frequent network disconnections, communications bandwidth limitations, resource restrictions and power limitations. In the last item context, Welch presents several adaptation techniques which can be used in mobile computer systems to reduce the power consumption of mobile computing devices, such as CPU and hard drive. With the aim of reducing CPU power consumption a mobile operating system or application can adjust CPU clock frequency for example. About hard drive, the mobile operating system or application can turn off (spindown) drives during extended periods of non-use of this device.

5. A REFLECTIVE MIDDLEWARE
ARCHITECTURE

Welling [10] asserts that adaptive techniques must be decoupled from basic application functionality due to the complexity of building adaptive applications for mobile computing. Such principle allows both applications and adaptive techniques be designed and implemented independent of each other.

In this direction Zhang [11] observes that middleware platforms architecture has been evolving exactly by the necessity of a software layer that decouples applications from the concern of handling the complexity related to distributed computing environments.

The architecture proposed in this work employs this principle of decoupling adaptation techniques (meta-level) from application basic functionality (base-level), as can be seen in Figure 1.

The Adaptation Manager module is responsible by deciding which adaptation strategies should be executed from data supplied by the inspection task of each resource managed by the same. Attention might be given to this module in the architecture. Due to the fact that conflicts may occur during the execution of a specific
adaptation strategy, rests to this module to guarantee that such conflicts can be solved in a coordinate manner. As can be seen, this module depends on two modules which responsibilities are complementary. It should be noted that the set of attributes must be extensible; by the way new attributes can be added in the proposed architecture.

The Adaptation Mechanism Manager module is responsible by loading the adaptation policies as well by updating dynamically these policies using statistical learning methods. By it is time, the Adaptation Task module is responsible by inspecting and by adaptation of the following attributes: connectivity, network bandwidth, power and memory. It should be noted that such attributes are inspected and adapted by the following modules: Connection Manager, Network Manager, Power Manager and Memory Manager.

The Connection Manager module is responsible by monitoring the connectivity between the application and the Remote Server. In case of disconnection, the data handled by the application are gotten from the Local Server module. The methods invoked towards the Remote Server module are stored in the local cache and will be deferred to execute by the Replication Manager adaptation task when the Remote Server was connected again.

The Network Manager module is responsible by monitoring the bandwidth network attribute. This module is responsible by compressing the body message which will be sent to the Remote Server. Besides, this module pre-fetches messages that were posted in the Remote Server and were not replicated to the Local Server yet.

The Power Manager module is responsible by monitoring the power attribute. If the amount of power in such moment were below a boundary expressed in the application profile, the interface between the application and the user is text based. Besides, the hard disk can be spindown and the CPU frequency can be scaled to save battery power. The devices responsible for ongoing power consumption in a mobile computer are, in this order: CPU, screen and disk [9]. Such observation justifies the adaptation techniques enclosed in this module.

The Memory Manager module is responsible by monitoring RAM and disk space. If the amount of disk space in a moment were below a boundary expressed in the application profile, the messages posted in the Local Server as well as data from local cache will be stored in the RAM memory, not more in the hard disk. The deferred methods will be stored in RAM memory either. Obviously, since there exist space in the RAM memory to do it.

6. IMPLEMENTATION

The reflective middleware architecture proposed in this article was implemented in an open source software platform, the Red Hat Linux operating system. The following open source software packages were employed to build the prototype which was used as this article proof of concept: Apache Axis SOAP engine, Apache Tomcat Servlet/JSP container and Apache Xerces XML parser. It should be noted that with aim of testing our architecture, a simplified mail server was implemented based on the Web Services technology.

Web Services are loosely-coupled, encapsulated, components that accomplish a well-defined purpose and make their interfaces available via standard protocols and data formats so they can be invoked over an internal or external network [3].

More specifically, Web Services is a distributed middleware technology that uses XML to allow applications to exchange messages in the Web [2]. The basic component of Web Services is the Simple Object Access Protocol (SOAP), a XML based communication protocol for interacting with Web Services. The network services available are described using Web Services Description Language (WDSL). It describes where a service is located, which operations are supported and the format of messages to be exchanged between Web Services entities (service providers and service requestors). Universal Description, Discovery and Integration (UDDI) provides a mechanism for service providers to advertise their services in a standard and for service requestors to query services of their interest.

Web Services are built as a set of components deployed on application servers. A component is a unit encapsulating the code implementation and exposing public component features through clearly identified interfaces. The server component provides an interface used by client components to use the service.

Our application environment is composed by a Local Server and by a Remote Server. The Remote Server was implemented with the use of the Axis communication protocol. It is a Java class that exposes public methods for invocation and is responsible by the following operations: create mailboxes, delete mailboxes, delete messages, list messages, read a message and send a message. It should be noted that the Remote Server encapsulates in its public interface the semantics of the main SMTP commands as exposed
in the Simple Mail Transfer Protocol (RFC2821) as well as the semantic of the main IMAP client commands as exposed in the Internet Message Access Protocol (RFC3501). The Local Server by its time is responsible by delete messages, list messages, read a message and send a message. It implements partly both specifications in the local system context.

The functioning of the proposed system is as follows. At the first request the mailbox data from the authenticated user is copied from the Remote Server to the Local Server by the mail client. The read operations are always local and in this case the disconnections from the Remote Server are not important. The write operations are executed simultaneously in the Remote Server and in the Local Server. During its execution, the system executes the prefetch of data that is stored in the Remote Server and is not already in the Local Server.

The mail client is a Java class that implements the user interface and uses Axis client API to call the services publicized by the Remote Server. All the calls sent by the client are intercepted by the class that implements the Adaptation Manager. The Adaptation Manager is a multithread component that allows to explore parallelism in the data update operations. It has two threads: SendMessageRS and SendMessageLS. All messages received by the thread SendMessageLS will be sent to the Local Server, which executes the operation and returns a result. The thread SendMessageRS sends the messages to the Remote Server. It should be noted that the Adaptation Manager is responsible by loading and updating the adaptation policies as well as by loading, unloading and controlling the adaptation mechanisms, as described in section 5.

7. CONCLUSIONS

Due to the fact that conventional middleware technologies do not provide appropriate support for handling the dynamic aspects of mobile applications, the next generation applications will require a middleware platform that can be adapted to changes in the environment and customized to several computational devices.

Computational Reflection allows the creation of a middleware architecture that is flexible, adaptable and customizable. The architecture proposed in this article employs this concept in its design and implementation. The results appointed by the researchers in the reflective middleware field were validated in the prototype developed to implement this architecture. In fact, the prototype allows a user to choose the adaptation strategies which will be managed by the Adaptation Manager module. It should be noted that these strategies can be added and removed dynamically by the user during the prototype runtime.

About future work, it will be studied the possibility of employing the aspect oriented software engineering approach in the architecture proposed with the aim of reducing its complexity and to become its design more manageable.

8. REFERENCES


