OBIHOC - Middleware for ad-hoc network

Paulo Ferreira, Luis Veiga, André Conrado
Instituto Superior Técnico – Campus Taguspark, Av. Prof. Dr. Aníbal Cavaco Silva — 2744-016 Porto Salvo

Abstract: The appearance of new mobile devices with significant improvements in terms of processing, storage and wireless communication capacity, suggests the possibility of creating ad-hoc networks. In view of this possibility, the need of developing applications for these networks increases. However this development implicates that programmers master areas of which, sometimes, they are not prepared for. Therefore this work proposes the creation of a middleware that releases the programmers of these duties, implementing them in a correct and effective way – OBIHOC. The OBIHOC middleware is a solution that proposes a data replication mechanism that allows maintaining unaltered the application's functions. The use of this mechanism allows a more efficient management of the resources used by the application (e.g. memory), as well as the possibility of using it on an ad-hoc environment, supported by multiple wireless communication solutions (Wi-Fi and Bluetooth) - Furthermore, it implements a DGC mechanism coherent in the presence of replicas as well as the use of SPL language, for specification and security policies implementation.

1. INTRODUCTION

In this world where technology advances by leaps and bounds, the technical developments that arise are numerous. If the computers are practical, without which the reality we know today would be totally different.

Through these technological advances, the paradigms of these devices have suffered many changes. One of the paradigms most popular is the mobility, i.e., users no longer have to move to their desktop, because these devices will be permanent with users. A practical case, and has a huge use, mobile phones. Who hasn´t one?

Although this new paradigm had brought the requirement for new features, the limitations of these devices to the energy level, memory and processing are a problem to resolve. Beyond these equipment's limitations, other priorities such as the ability to be always connected don´t be ignore.

To gather this priority, currently we are dependent on structured networks – Wi-Fi [1], Bluetooth [2], GPRS [3]. This limitation sets rather the ability to create a customizable network, i.e., independent of the generic service that is offered to all members of that network as well as their availability. In addition, these networks are mostly associated direct costs for the user.

As an example, consider two people are in a garden and one of them wants to share some content from her digital library. At this place does not have access to any external network infrastructure. In a real case that would be difficult to realize this change of content. But if the contents are in mobile devices and these devices are within range of each other, which need to connect to an external network if the person you want to swap the contents is right next to me?

Thus, it is natural to consider that the ad-hoc networks facilitate the development of new applications. However, developing these applications is complex because it is necessary to deal with various problems in low-level layers of the system - system-level - for which most programmers are not ready. In addition, programmers would be forced to deviate from the application logic, for which they are prepared and which should be concentrated.

For these reasons, this was designed, implemented and evaluated a middleware OBIHOC, which has the following characteristics:

- **Paradigm Flexibility:** allows programmers to develop applications using either RMI, object replication, or mobile agents, according to the specific needs of applications.
- **Automatic Replication**: supports distributed memory management capable of dealing with object replicas automatically (incremental replication).
- **Distributed Garbage Collection (DGC)**: supports the automatic reclamation of useless replicas.
- **Security Policies:** supports the definition and enforcement of history-based security policies well adapted to agencies needs.
- **Ad-hoc network**: supports the entrance and exit of devices on the network.

2. RELATED WORK

This piece of work can be related to several different systems currently in existence which supports remote object invocation, replication, DGC, security and ad-hoc networks. However most of them just provide some of the mechanisms, not all. OBIHOC provides paradigm flexibility (RMI, replication and mobile agents), automatic replication, DGC (consistent in the present of replicated object) and security politics.

CORBA's [4] aim is to develop and specify an infrastructure for distributed systems based on objects (reutilization, portability and interoperability between heterogeneous distributed systems). It is a system drawn to provide support for replicated objects on a wide scale network, such as World Wide Web. The key word is transparence, based on client / object communication. This system does not provide DGC mechanism, security politics or implementation ability on an ad-hoc network.

Javanaise [5], a platform developed on JAVA, offers support to collaboration between Internet' distributed applications. With this platform, programmers develop their

applications as if they were centralized. After this, they build up the application to became distributed, don´t being necessary any change on font code. Then, an automatic mechanism generates all the needed data structure. Nevertheless, Javanaise does not allow incremental replication (although programmer can define a group of objects), nor definition of security politics, mobile agents or DGC.

Thor [6] is an OO data base management system. Developed for heterogeneous distributed systems, it makes possible programs written on different languages to share objects. This system allows users to store and manipulate objects that capture the semantics of their applications. Besides, it provides users a range of persistent objects. These can refer other objects, allowing the use of objects structures, such as graphics and trees. In order to support heterogeneity Thor has a specific language, independent of the ones of programs. This language allows the development of hierarchic systems to hold applications evolution. Notice Thor provides a limited DGC mechanism once it does not include replica existence.

Object Space Voyager [7] is a mobile agent system based on Java. An agent is an object with the ability to move on a network. On this system, is introduced the concept of virtual object, this is, a representation of an object or agent using a proxy. With OSV, it is possible to transform any object on a mobile agent, using a virtual compiler. However, this system does not make available automatic replication mechanisms or DGC.

OBIWAN [8] is a middleware platform that allows develop distributed applications without the programmer is compelled to deviate from the application logic. With the OBIWAN the application allows to decide, at runtime, the mechanism by which object should be invoked: remote call, local call or mobile agent. The creation and management of replicas is done transparently, however the programmer can control, at runtime, the amount of replicated objects. Concerning DGC, the OBIWAN provides a mechanism that resolves some of CGD caused by replication of objects, for example, the possibility of replicas. This system supports the use of mobile agents and consequently a security mechanism. This mechanism uses the definition and implementation of security policies based on previous events. This support is extremely important because it allows the organization security policy, in which the agent's past behavior influences the permissions.

7DS [9] is a system that allows exchange of information between devices that have an intermittent connection to the Internet. In this system some of the participants in the network obtain data from several servers on the Internet and store data. After this operation the stored data can change with devices that do not have Internet access. The data changed could be web pages, maps, short clips, music and another type of object with small size. 7DS provides some mechanisms for data replication in ad-hoc network. However, it provides no mechanism for DGC, or security policies or the flexibility of paradigms.

Babylon [10] is a system that provides facility development, implementation and parallel and distributed management of Java applications mobile. This system supports object migration, asynchronous method invocation and downloads remote classes without authentication. Furthermore, was given the possibility of application to create and interact with remote objects, and can also protect these objects from other applications. This system has not support for security policies neither implementing an ad-hoc network.

Ajanta [11] is a system based on Java mobile agents. The main security concern of this system is seen to protect the devices from mobile agents by implementing an access control mechanism to protect the device. Have not considerer past events.

SPL [12] is a system that defines a vast group of political commitment and obligation, very efficient and controlled by a safety monitor. Allows the creation of complex security models such as access control based on preceding events, discretionary access control and policies based on obligations. Allows the definition of different kinds of entities: objects, groups, rules and policies. Note that the rules restrict the relationships between objects and groups and the policies resulting from the decomposition of multiple rules and groups.

3. ARCHITECTURE

OBIHOC it’s a middleware peer-to-peer, in the since that all devises may work for clients or servers at any given moment. This way, an application used on a devise, may carry out tasks of server (supply objects for replication) or client (demand the replication of objects from other devises). Its Architecture was though out in a general form so that any programmer could integrate it in its own application very easily. Furthermore, it is easily adaptable to innumerous options of existent hardware.

From the point of view of the programmer, his application will communicate directly with the other application, however, this communication is established by OBIHOC, allowing an abstraction form the communication method, as well as the management of the objects to by replicated. However, this abstraction cannot be total because it remains to be an application that defines certain parameters, such as the technology used in the communication (Wi-Fi or Bluetooth for example) as well as the objects that may be replicated (for example, choosing the depth of the replication).

As possible of verifying in Fig. 1, the architecture of OBIHOC as three modules:

- **Objects management** – module responsible of managing the objects that may be replicated. It is also in this module that DGC control takes place.

- **Security** – module responsible for validating the security policy.
- **Communication** – module responsible for the transfer of objects between devices. It allows you to establish connections with other applications and perform all information exchanges needed.

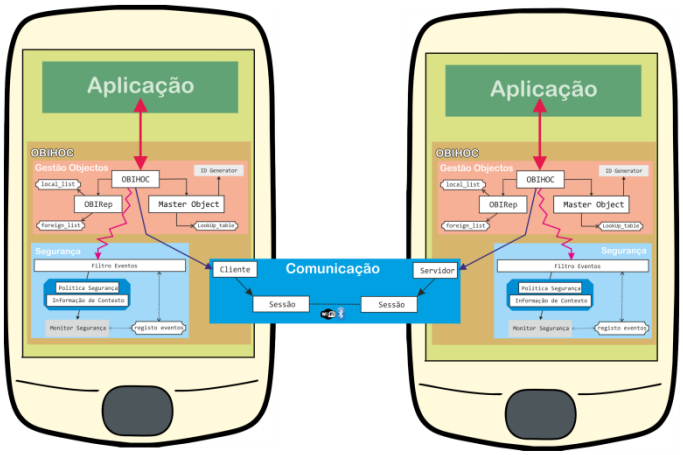

Fig. 1 - OBIHOC architecture

3.1. OBJECTS MANAGEMENT

For the objects management, the architecture allows (1) storage of information about which objects are suitable of being replicated to other devices, (2) managing the replication mechanism and invocation of local objects and/or remotes and (3) guaranty the correct mechanism operation of DGC.

This function uses five modules:

- **OBIHOC** – makes the connection between appliance and middleware. It's the central module of the whole architecture, as it is he who unchains the majority of processes that allow middleware operation.
- **OBIRep** – stores a data structure that allows the management of objects replications. Furthermore, this module has an active rule in the DGC mechanism.
- **Master Object** –stores information about the graphs of objects that are suitable for replication.
- **ID Generator** – generates unique identifiers

The OBIHOC allows the programmer of the application to define which objects odd to be invocated remotely and locally. This way the programmer has the ability of defining the best way of invocating the object, having in consideration the needs of the application and the resources used by the device.

The OBIHOC module unchains all processes needed for the proper operation of middleware. Considering that an application can be either used as client or as server or even both, this module contains all properties for each case, for example, an addition of objects graphs is a process typically associated to a server, where as the request of an object replication is typically associated to the client process. For the operation of these, two modules are used– OBIRep e Master Object.

The Master Object is responsible for managing the objects identifiers, meaning, every time a new object is added to this module a unique identifier is created. Furthermore, it possesses a list that associates the graphs name to the object identifier of the rout of that graph – LoohUp_table

The OBIRep is responsible for managing local and remote objects, meaning that it manages the local objects as well as the objects that were replicated from other devices. This module consists of two lists. The local_list allows storing, just as suggested by its name, data structures of local objects that are suitable to be replicated for other devices. The foreign_list is a list of objects that were replicated for other devices. The utilization of this list allows the mechanism DGC to be controlled, for as long as an object stays on the this list, that means it was replicated e for that reason should not be erased without the verification that it is in fact suitable for being erased.

All objects stored in this list have:

- A numerical identifier (Long);
- A PropEntry – a data structure that allows associating various attributes to that object;
  - PropState: flag that allows to differ the various status that the object might have – NONE, PROP_IN e PROP_OuT.
  - ObjRef: a reference for the local object;
  - foreignID: identifier oh the position held by the object in the foreign_list, after it has been replicated. If this identifier has a value of -1 that means it still hasn't been replicated

If an object is marked with a flag NONE, that means it still hasn't been loaded, meaning, that it is necessary to obtain the local object where it was stored, for example, a data base. This flag is used in application that required persistent data storage. The flag PROP_OUT allows identifying an object as being local and suitable for being replicated. The flag PROP_IN is used in devices to who the objects was replicated, meaning, that during the process of replicating an object, if this is successfully done, than that object is marked with this flag. For the replication mechanism, the use of these three flags is enough. However, more flag can be added according to the security policy applied (for example to prevent the object to be replicated outside the devise that created it).

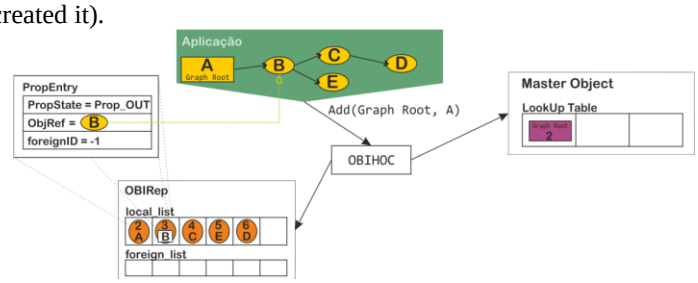

Fig. 2 - Adding object graphs

For the replication of objects it's necessary to make a request to an application so that it lets it be available. This request consists on the graphs name and the reference to the rout. Afterwards, starts an automatic process that analyses every objects of that graph and if identified as an object suitable for replication, it is given a unique identifier created by the ID Generator and a PropEntry data structure is created and a flag PROP_OUT is attributed. After that it is added to the local_list which connects that identifier to the data structure. On the example presented in Fig. 2 you may see in detail that object B has been added with identifier 3 and respective PropEntry.

In the end of the process, the graph e finally added to the LookUp_Table list (Fig. 2), in the Master Object module.

From that moment on, this graph stays available to be replicated for other applications.

3.2. REFERENCE MANAGER

The replication of object might cause some problems on the application's behaviour, since it's necessary the execution of a lower level mechanism in comparison to the application. From de point of view of the application all objects are locally invocated, however, in some cases this might not happen, because the invocated object might not have been replicated. It's in these situations that reference management is important, because this way it's possible to maintain intact the application's behaviour. In the cases where the objects are local, reference management is useless, because the application might invocate them directly. Meanwhile in cases of objects replication, this management gains extreme importance, and that is one of main focus of this work.

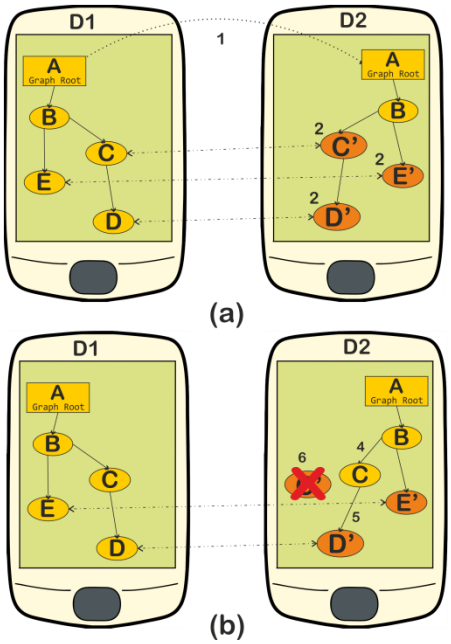

Fig. 3 - Replication of "Graph Root" from D1 to D2, with deep 1

Consider the situation in which a device D1contains a graph of objects prepared to be replicated to device D2. This graph of objects is named "Graph Root", being the root of the graph object A.

In any given moment, D2 takes all the steps to initiate the replication of the graph "Graph Root" from D1 to D2, with a depth of 1. After completing this process, we get a situation like the one depicted in Fig. 3 (a) in which the root of the graph as well as object B have been totally replicated from D1 to D2 (1).The desired depth was 1 and for that reason the remaining objects of the graph haven switched for proxy's (represented by "C", "D", and "E") in D2 (2). These proxies' allow the application to continue to function even if it doesn't have the actual object, because these implement and interfaces of the original object. Furthermore, the proxy's maintain a reference for the object of the device of origin (3), allowing the application to do remote invocations on that object.

In the case of an invocation of an object represented by a proxy not being explicitly defined and a remote invocation, a new process is automatically unchained to replicate that object. Continuing the previous scenario but referring to Fig. 3 (b), the application makes a new invocation to object "C", which is in fact proxy C. A replication request for that object is sent to D1. After receiving the new object, the process reference update is initiated. In this case a method is invocated on object B in order for this to pass referencing object C, recently replicated (4), and the next C stars to point at D, which is a proxy from de original object D from D1(5). Finally, the proxy C stops being reachable and can now be deleted (6). From that moment, any invocation to object C will be done locally, allowing its repercussion on object C of D1, and for that an update request is sent.

The OBIHOC allows the replication of a set of objects, depending on the chosen depth, by doing so it minimizes the resources used by the device, because instead of an object being treated individually, they are grouped and replicated together, avoiding needless references updates.

3.3. DGC

The use of resources on devices most be controlled to guarantee that they are used in the proper way. This attitude might lead to situations in which the objects are marked as garbage, since they are no longer unreachable locally. However, this situation should be treated carefully. The object should not be treated just locally, but in a global context, having in consideration all the replicas that might exist throughout the various devices. This situation demands that the referential integrity is maintained, meaning that the objects should maintain their references for the correspondent objects.

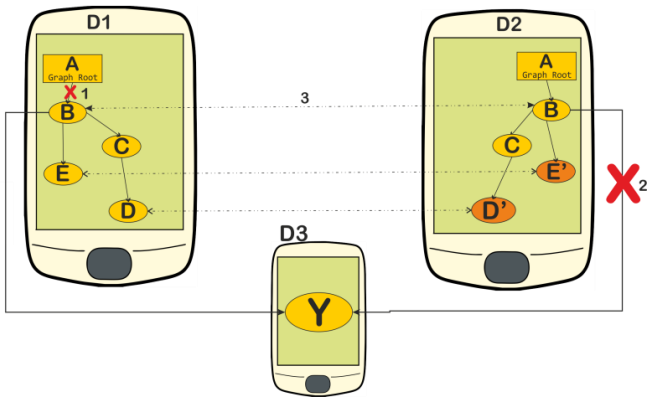

Fig. 4 - DGC scenario

Considering the scenario presented in Fig. 4, in which object B on device D1 has been replicated to D2. In this case, object B in D2 will also make a reference to Y in device D3, just like in object D1. During the execution of application D1, object B stops been reachable (1) and in D2, object B stops referencing Y (2). In this case should Y be considered inaccessible and garbage? In reality, object Y should not be considered garbage, unlike what the majority of GC would say. Object Y should not be considered garbage because object B in D2 might be updated and considering that in D1 object B maintains the reference to Y, this update would recreate the reference in D2 between B e Y (3). This way Y should not be considered as garbage, despite that in D1 B it stops being referenced and that D2 B, Stops referencing Y.

So, Y should only be considered garbage when the union of all replicas stops referencing Y.

In relation to the local environment, the treatment of inaccessible objects e left for the local garbage collector (GC). In case of an environment distributed beyond the local GC, collaboration between the GC and the DGC is needed.

For the proper function of this algorithm, existent lists form module OBIRep are used – local_list e foreign_list. When an object is replicated to another device, the module OBIRep executes two actions on these lists:

1. Creates a new PropEntry similar to the existent in local_list, however this one has a new field – hostID. This field is a list of all the other devices to which the object has being replicated to. Furthermore, each element of that list is assumes the device identifier as the local identifier of the object in that device.
2. The PropEntry of the object replicated from the local_list in updated, meaning that the value of the field foreingId comes to be the position of the object on the foreing_list.

With these two actions the collaboration between local GC and DGC is possible because when an object is locally inaccessible it is marked has being garbage. However, before the object is definitely eliminated, the DGC mechanism will verify if there is any existent reference for that object on the foreing_list.

If it exists that means that the object was replicated to another device and therefore should not be eliminated without verifying if it is globally inaccessible. For this verification a message is sent to the device for which the object has been replicated so that it can verify if it is inaccessible. In case of a positive response, the next device to which the object has been replicated will be contacted. If there is a positive response from all the devices, the object is considered globally inaccessible and, therefore, suitable of being locally removed.

However if one of the devices gives a negative response, then the object should not the removed, because it might be updated and this way becoming accessible again, just like represented in the example on Fig. 4.

In the case of an object been replicated, already being registered in the foreing_list, meaning that it has already been replicated to another device, that information will also transferred in the process of replication. Besides that, all devices to which the object has been replicated will be notified. This situation creates two rules:

- Before a copy is sent, all devices to which the object has been replicated should be notified, to add that fact in the hostID field of the respective PropEntry.
- After a copy is delivered, the device should notify all devices to whish the object was replicated, informing the local identifier of that object, to confirm the delivery of the object.

Summarizing, this DGC mechanism allows an object to only be considered garbage, when globally inaccessible, meaning, when all the existent copies won't reachable that object.

3.3. OBJECT MIGRATION

To support the possibility of objects migration, namely the use of mobile agent, the architecture used is similar to the objects replication situation. However there are a few changes due to the characteristics of these agents. One of the main characteristics of mobile agents is the capacity of moving throughout the network to execute their functions. Therefore, is crucial that this movement is taken into consideration within the architecture of OBIHOC. When the application creates a mobile agent, two other data structures are created and associated to that agent:

- **Home Agent** – this module allows a interaction between the application and the mobile agent and is located on the device from which the agent was launched. It's in this module that the information about the agent's location is kept.
- **Mobile Agent** – this module interacts with the home agent module and moves side by side with the agent, meaning that it's logged in the same device as the mobile agent.

These data structures allow the application to interact with the Mobile Agent without it needing to know the location of the Mobile Agent, because the communication e mediated by the Home Agent and the Mobile Agent. So we can see the Mobile Agent as an extension of the Home Agent in the device visited by the agent. With this mechanism the application doesn't need to know the location of the agent to interact with it. This interaction is made by the OBIHOC, making in completely transparent to the application.

3.4. COMMUNICATION

The replication of objects needs the existence of two components: (1) a server to arrange objects and (2) one or more clients that may interact with that server.

Considering that this work is focused for an ad-hoc environment, various aspects have to be taken into account:

- An application should have the possibility to perform the role of the server and client at the same time or one of the cases separately
- The characteristic of mobile devices, namely in terms of processing and memory
- The communication technology being used

The architecture for this component is composed of three modules:

- **Session** – module responsible for the connection. In this module some parameters are established, such as the technology being used and the properties of the connection.
- **Client** – module that implements the protocol communication, meaning that it creates types of data to be sent in requests to the server.
- **Server** – module that receives the coming request from the various clients. In the case of response to the receive requests, this module is responsible for the creation of data to be sent.

Of the previously presented modules, the session module is the one that needs a configuration from the application. Considering the existence of various forms of wireless communication – Bluetooth and Wi-Fi – it's necessary that the OBIHOC is more flexible as possible so that it allows the use of these communication technologies. The technology choose should by made the application, because only this way the application can know its needs, so this module to be created and configurated by the application. During this process the application should indicate which technology should be used, as well as the respective properties.

During the execution of this process there are some compulsory parameters, in the previous case, the IP e the server's port. However there are some optional parameters that may be defined:

- Timeout – total time in which the session is valid. In the precious example that time would be five minutes. The use of this parameter allows the application defines a reasonable time line for the connection to stay active. This way its possible reduce needles use of the resource in the case of a time line not being defined, a predefined time line (5 minutes) is set.
- User – Identifier of the network application. In the case of this parameter being omitted, it will be generated by using a device property, for example de IP address.
- Security protocol– The security protocol being used during the. This protocol should be defined by the application e can include for example, authentication and/or cipher de switched data. In the case of this parameter being omitted no security protocol will be used.

Considering the countless applications that may use this middleware, it was necessary to create a communication architecture that was flexible enough so that it can adapt itself to each applications needs. For example, the use of a security protocol on an application in which the data exchange doesn't need to be secure, this would be using resources that are important for the device.

To start server, the application should indicate the port where this will be waiting to receive the requests. From this moment the server is adequate to receive object graph from application and letting them be available for the other devices, without it needing to interact with the server.

The establishment of the security protocol should be made by the application's programmer, because only it can know the best security mechanism to be used (for example, an authentication method and the algorithm cipher).

3.5. SECURITY

For the security policies definition, the architecture OBIHOC uses a SPL language. Due to the previous announced characteristics the use of this language was the most suitable considering the environment of the applications use.

Due to the characteristics of this work, namely the migration of objects between devices, it's essential to be able to offer a security mechanism of the own device and of the objects present in the migrations. In the specific case of OBIHOC, the main concerns are related to:

- Interaction between Mobile Agents and devices – The inappropriate use of resources from a ill intentioned agent on a device and vice-versa;
- Interaction between agents – unwanted interaction between mobile agents.

The security component has five modules:

- Events Filter – module allows to detect which events can unchain the need of analysing the security policies;
- Security policy – the security policy to be presented for validation;
- Context information – information associated to the policy that is going to be used;
- Security monitor – module responsible for analysing the security policy presented. For this analysis this module may resort to context information presented along with the security policy. After the analysis, the policy will be validated or rejected.
- Events register – module responsible for the storage of important events and will be used for situations in which passed events influence the validation.

During the execution, OBIHOC may unchain several types of events and for that reason the use of an events filter is very important because it prevents the needles use of resources on the device. When an unchained event endangers the device (migration of remote objects or invocations are some of the examples) the security monitor receives an authorization request for the event's execution. After the analyses of the security policy and respective context information, the security monitor informs the system if the event is authorised or not, notifying the events register about the decision as well. For the security policies evaluation we resort to the language for defining SPL security policies.

The SPL language is based on 4 entities: Objects, rules and policies. The rules establish restrictions in the relation between objects and groups. The policy results in the composition of multiple rules and groups and these are the main module of SPL language.

4. IMPLEMENTATION

This work was implemented for the Android operating system [13]. This very recent operating system is focused primarily for mobile devices, and so, it was chosen to be the implementation platform. It is mostly an open-source platform and, therefore, very attractive for programmers developing new applications. It also has available a Software Development Kit (SDK) that allows the simulation the operating system behavior, replicating a mobile device.

The applications for this operating system are developed using Java programming language [14]. Therefore, OBIHOC was also developed in this language, running on a Java virtual machine [15] (JVM - Java Virtual Machine). Furthermore, the JVM supports several features to be used - RMI, sockets. One of the positive factors is that OBIHOC requires no changes to the JVM and this reason makes it easily portable.

4.3. CLASSES AND INTERFACES

To facilitate the implementation of OBIHOC in any application, it was necessary that these classes extend some OBIHOC specific interfaces as well as the creation of a new class - proxy. The required interfaces are:

- IMobihocObject – allows the objects to have a property that will be changed by the identifier generated by OBIHOC. Classes that implement this interface are considered possible to replicate and therefore should also implement the interfaces IXMLTranspor and IDemander / IDemandee.
- IXMLTransport – allows the conversion of a class into an object, possible to be replicated by OBIHOC along with its recovery in the device on which was replicated.
- IDemander – allows changing the references after the arrival of a replicated object in exchange with the proxy that replaced it.
- IDemandee – allows running the replication process automatically. Thus, when a proxy is invoked, starts automatically the replication process of the corresponding object.

For the proper implementation of OBIHOC on the application, the programmer must:

1. Define the interface of the class and implemented.
2. Implement, on the class, the interfaces (1) IMobihocObject, (2) IXMLTransport and (3) IDemander.
3. Create the class proxy that implements the interface of the object and (1) IMobihocObject, (2) IXMLTransport and (3) IDemandee.

To implement the communication process it was used a very simple and minimalist solution. In order to reduce the resources used on the communication devices, the communication is performed using the direct communication features offered by the JVM (sockets).

Given the similarities between SPL language and Java, most of compiler's actions are translation actions: each SPL policy is translated into a Java class, each rule in a function without parameters and SPL objects are translated into Java interfaces.

This work was developed using the Eclipse IDE, Galileo version. It was used the ADT plug-in (Android Development Tools) because it allows the integration of the IDE with the SDK, with the focus on two tools: (1) ADB (Android Debug Bridge) to debug and (2) DDMS (Dalvik Debug Monitor Server) for control and management of the processes and resources used in the emulator. The SDK version 2.2, the latest version available, was also used.

5. EVALUATION

The evaluation of the work was done in two stages: (1) definition of a microbenchmark and (2) performance analysis of an application developed using OBIHOC. Through this second method, it is possible to get credible results so that the

work can be evaluated in a realistic scenario. The evaluation will be based on the following aspects:

- Resources used in the device – Due to memory and processing limitations in mobile devices, it is essential to analyze the impact that OBIHOC's execution will have in these devices.
- Communication overhead – Since the use of ad-hoc networks implies data exchange between devices, it is important to minimize the overhead of data exchanged.
- Impact in the application – Despite the extra features given to the application, these should not influence the flow of the application, so that the desired transparency isn't lost.

In this chapter, the developed application is explained as well as how the executed tests have contributed to the gathering of relevant results for the evaluation of the work. In each of the following subchapters, the work flow for the gathering of data, its importance and analysis are explained.

5.1. DEMO APPLICATION

The demo application, called *Digital Library*, is an application designed and developed for Android version 2.2. This application allows the checking, adding, editing and erasing of different content in this library. Content can be of three types: (1) audio (music, voice recordings, etc.), (2) documents (books, reports, etc.) and (3) videos. This application shows the advantages of using it in an ad-hoc network, giving it extra features. Among the extra features, it should be highlighted:

- The possibility of sharing content library without the need for an external service, using just its own ad-hoc network.
- The possibility of having the application executing, through content replicas, despite communication failures. This situation will most definitely occur in an ad-hoc network since these networks are bound to have devices entering and leaving the network at any time.
- More efficient use of device capabilities. Since this application can transfer high quantities of data, it is important that this data is really necessary for the intentions of the user.

5.2. TEST SCENARIO

For the evaluation of this work it was created a test scenario which uses the application *Digital Library*. This test scenario is based on the access of all the contents in a remote library in a sequential way, in other words, first the properties and content of all audio are accessed, after these the documents and finally the videos. There is a file associated with each content, kept in the local machine with the following properties:

- Audio – .mp3 file with 484KB;
- Document – .pdf file with 100KB;
- Video – .mp4 file with 988KB;

The application was configured so that, during its execution, if some object is invoked but, it in fact represents a proxy, the process of replication is automatically started for that object and later, the wanted method is invoked on that recently replicated object. The depth to be applied in this process depends on the type of test being executed.

This scenario was done using two emulators executed in a PC with a Intel® Core(TM)2 Duo processor, 3GB of RAM memory e with the operating system Windows 7 Professional. Due to the fact that the emulators were being executed on the same machine, the communication speed reached didn't match the reality. For this reason, the emulators were configured so that they showed a communication speed close to that of the communications protocol HSDPA, commonly known as 3.5G.

5.3. RESOURCES USED

Just like it was referred before, mobile devices have several limitations in resources. In this case, the access time for a certain object that hasn't been replicated yet will be analyzed as well as the total memory used in the device.

5.3.1. ACCESS TIME

To get this data results, the test scenario referred before was executed with a depth of 0, in other words, each object is replicated individually, leaving aside the situation where a group of objects is replicated. The replicated library had 10 audio contents, 10 documents and 10 videos. This scenario was executed 10 times, being the shown results the average of the execution of the scenarios.

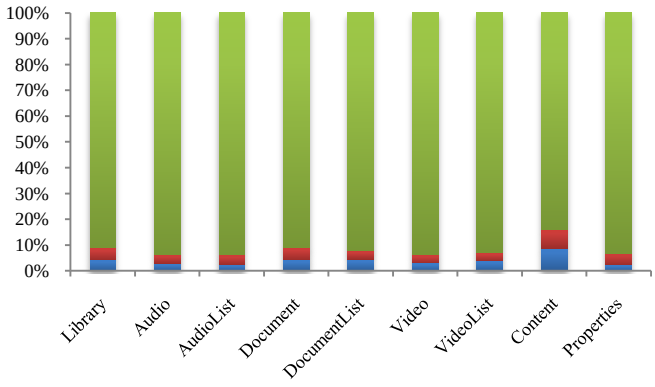

Fig. 5 - Access time to remote object

Fig. 5 represents the total percentage of time used by each process during the replication process. The processes represented in the graphic are: (1) creation of data structures to be sent from server to client (in blue), (2) conversion of the information, received from the server, to objects being used

by the application which includes the reference manager (in red) and (3) data transfer over the network (in green). As one can see, most of the time needed to access a file that hasn't been replicated yet, is used in the data transfer between devices. This let's one conclude that the use of this middleware does not affect directly the response time of the application, since the data exchange is independent from the middleware and essential for the application's proper operation.

5.3.2. MEMORY USAGE

To get this data results, the test scenario referred before was executed with a depth of 0, in other words, each object is replicated individually, leaving aside the situation where a group of objects is replicated. The replicated library had 20 audio contents, 20 documents and 20 videos.

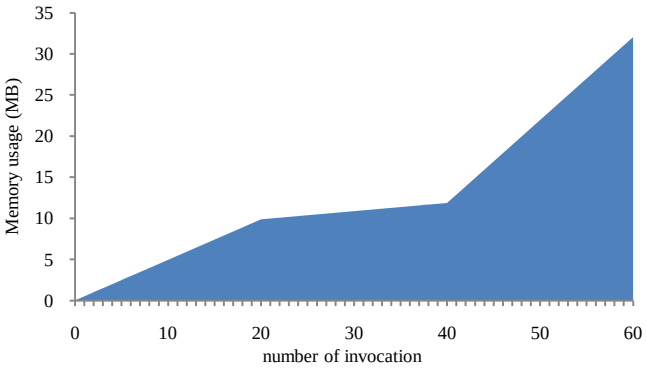

Fig. 6 -Memory usage on device

Fig. 6 represents the memory used in the device, during the execution of the test scenario described before. Looking at the graphic, one can see three distinct slopes. Each of these represent the invocation of each content type, in other words, the invocation from 0 to 20 are audio contents, from 20 to 40 are documents and the remaining are video content. Since the test used files with the same size for the same content type, the memory usage in the device is constant during the invocations of the same content type. For this reason, the three moments have different slopes that depend on the file type associated to that content.

With this data, one can see that the integration of OBIHOC with the application allows reducing the usage of unnecessary resources in the device, in a way that, only the necessary data for the user to interact with the application is transferred and stored in the device. Although this could be done by the application, it would require a lot of effort by the programmer. With the integration of OBIHOC, these features are easily associated with satisfactory results.

5.4. COMMUNICATION OVERHEAD

Since this application runs on a mobile environment, it is necessary to check data overhead exchanged between devices. For this reason, the test scenario was executed 10 times and the data collected indicated the quantity of information exchanged in which had also the quantity of data transferred that corresponds to the really useful data for the application.

The number of bytes for each data type exchange between devices was analyzed, in other words, the number of bytes received for each object and the number of bytes that ended up being useful for the application. Each value was obtained through the average of the execution of all test scenarios.

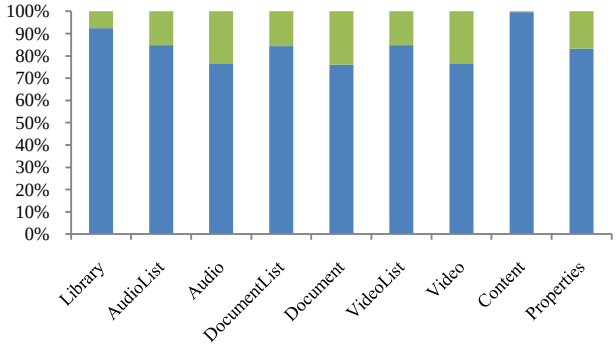

Fig. 7 - Overhead data transferred

Fig. 7 represents the percentage of data received (in blue) and the percentage of excess data (in green). The Content object shows a distinct behavior with a percentage very close to 100% for useful data. This is a normal value since the replication of this object implies the transfer of high quantities of data (data associated to a file). This excess data result from the data structure created by OBIHOC, in which the object to be replicated is inserted. This structure has several parameters that allow for the proper execution of OBIHOC, for example: identifier of the request type sent, identifier of the object and the list of the devices for which the object was already replicated.

6. CONCLUSION

Technological advances in mobile devices provide the possibility of incorporating new features. Due to significant improvements in processing, storage and communication capacity wireless mobile devices are increasingly present in our professional and personal lives. However there are still enough options to be explored, including the creation of applications that benefit from a distributed environment, for example an ad-hoc network.

However the development of applications in this environment means that programmers dominate areas for which, in most cases, are not prepared. Moreover, these areas differ from the logic of the application, which the programmers should be concentrated. Thus this study allowed developing a middleware that relieves programmers of these areas, implementing them effectively and correctly.

Throughout the study analyzed various technical aspects: 1) flexibility of paradigms, 2) automatic replication, 3) DGC and 4) security policies. In addition to this analysis was

carried out research work on another works: for example OBIWAN.

The evaluation work was performed in a controlled environment using emulators. However this is not an impediment to conclude, based on the results obtained, which allows the use of middleware rather improve the characteristics of applications, including: (1) integration on an ad-hoc network, (2) control of resources to be used in mobile devices, (3) using targeted security policies for each application, (4) maintaining a consistent state in the presence of replicas and (5) increase in functionality that will be available to users.

Concerning future work is important to implement the flowing features to improve this middleware:

- Developing a plug-in (for Eclipse) to assist the development of applications
- Integration of a mechanism to automate the generation and extension of code.
- Development of a graphical interface for defining security policies.
- Possibility of integration of a protocol for data consistency.
- Integration of the middleware in case on other mobile operating systems.

7. REFERENCE

[1] M S Gast and M Loukides, *802.11 wireless networks: the definitive guide*.: OReilly & Associates, Inc. Sebastopol, CA, USA, 2002.

[2] Brent A Miller and Chatschik Bisdikian, *Bluetooth Revealed*.: Prentice Hall PTR, 2001.

[3] R Kalden, I Meirick, and M Meyer, "Wireless Internet access based on GPRS," *IEEE Personal Communications*, vol. 7, pp. 8--18, 2000.

[4] S Vinoski, "CORBA: Integrating diverse applications within distributed heterogeneous environments," *IEEE Communications Magazine*, vol. 35, pp. 46--55, 1997.

[5] D Hagimont and D Louvegnies, "Javanaise: distributed shared objects for Internet cooperative applications," in *Middleware'98*, 1998.

[6] B Liskov, M Day, and L Shrira, "Distributed object management in Thor," *Distributed Object Management*, pp. 79--91, 1993.

[7] A Silva, M Mira da Silva, and J Delgado, "An overview of AgentSpace: a next-generation mobile agent system," *Lecture Notes in Computer Science*, pp. 148--159, 1998.

[8] P Ferreira, L Veiga, and C Ribeiro, "OBIWAN: design and implementation of a middleware platform," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, pp. 1086--1099, 2003.

[9] M Papadopouli and H Schulzrinne, "Seven degrees of separation in mobile ad hoc networks," *GLOBECOM-NEW YORK-*, vol. 3, pp. 1707--1711, 2000.

[10] W van Heiningen, S MacDonald, T Brecht, and M S Witter, "Babylon: middleware for distributed, parallel, and mobile Java applications," *Concurrency and Computation: Practice & Experience*, vol. 20, pp. 1195--1224, 2008.

[11] A Tripathi and N Karnik, "Protected resource access for mobile agent-based distributed computing," in *Proceedings of the 1998 ICPP Workshop on Wireless Networks and Mobile Computing*, 1998, pp. 144--153.

[12] C Ribeiro, A Zuquete, P Ferreira, and P Guedes, "SPL: An access control language for security policies with complex constraints," in *Proceedings of the Network and Distributed System Security Symposium*, 2001, pp. 89--107.

[13] Google Inc. (2007, Abril) Android.com - Android at Google I/O. [Online]. http://www.android.com/

[14] J Gosling, B Joy, G Steele, and G Bracha, *Java (TM) Language Specification, The (Java (Addison-Wesley))*.: Addison-Wesley Professional, 2005.

[15] T Lindholm and F Yellin, *Java virtual machine specification*.: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[16] The Eclipse Foundation. (2010) Eclipse.org home. [Online]. http://www.eclipse.org/