Cloud patterns for mobile collaborative applications

Nadir Guetmi, Abdessamad Imine

Cloud Patterns for Mobile Collaborative Applications

Nadir Guetmi†

LIAS/ISAE-ENSMA, Poitiers University
Chasseneuil, France
E-mail: nadir.guetmi@ensma.fr
† Corresponding author

Abdessamad Imine

Université de Lorraine and INRIA-LORIA Grand Est
Nancy, France
E-mail: abdessamad.imine@loria.fr

Abstract: Deploying collaborative applications (e.g. group editors) over mobile devices is problematic because these devices will always be resource-poor and with unstable connectivity and constrained energy. To overcome these limitations, one straightforward solution is to leverage mobile collaboration via the cloud, which is an emerged model based on virtualization for efficient and flexible use of hardware assets and software services over a network without requiring user intervention. However, designing collaborative applications with flexibility and reusability has become a hot topic in mobile cloud computing as no mature models have been proposed yet. In this paper, we describe cloud patterns (i.e. extension of classic design patterns) focusing on the description of mobile real-time data sharing through the cloud. Our design model consists of two levels: the first one provides self-protocol to create clones of mobile devices, manage users' groups and recover failed clones in the cloud. As for the second level, it supports group collaboration mechanisms for data sharing between mobile users via their clones. Our patterns have been used as a basis for the design of (i) MIDBOX a platform for supporting mobile collaboration over a private cloud, and (ii) OPTICLOUD a cloud service for scalable real-time editing works.

Keywords: Mobile data sharing, Collaboration, Mobile cloud computing, Cloud pattern, Cloning middleware, Synchronization.

1 Introduction

Nowadays, mobile devices are increasingly part of our everyday lives, taking on more and more tasks. A variety of services (such as real-time data streaming, mobile commerce, social networking and ad-hoc collaboration) are available through mobile applications that take benefit of the increasing availability of built-in communication network and better data exchange capabilities of mobile devices. In addition, in terms of flexibility and mobility,

mobile devices provide the tool of choice for people to collaborate with family members, friends and business colleagues in order to achieve a common goal.

However, even though mobile devices hardware and network modules continue to evolve and improve, mobile devices will always be resource-poor, less secure, with unstable connectivity, and with constrained battery life. Resource deficiency is a main issue for many applications, and as a result, computation on mobile devices will always involve a compromise [1]. More precisely, managing collaborative works (e.g. editing a shared document or a friend list) in real-time through ad-hoc peer-to-peer mobile networks is often costly in terms of energy consumption and network traffic and it may lead to the congestion of mobile devices [2, 3]. Moreover, it is not possible to ensure a continuous collaboration due to frequent disconnections.

To deal with the mobile resources limitation, one straightforward solution is to delegate the majority of mobile intensive computations to the cloud. Thus, enjoying the benefits of the virtualization in the cloud enables us to extend the mobile device resources by offloading execution from the mobile to the cloud where a *clone* (or *virtual machine*) of the mobile is running. Cloud computing allows users to build virtual networks "à la peer-to-peer" where a mobile device may be continuously connected to other mobiles to achieve a common task. The goal of this cloning is to provide self-configuration capabilities such as on-demand resources provisioning (e.g. memory size) without requiring user intervention.

Defining reusable designs is more challenging in new collaborative applications for cloud-supported mobile data sharing services, because the combination of mobile and cloud environments raises many design issues such as the management of real-time data synchronization. These mobile collaborative applications require suitable design patterns for modeling communication and synchronization services that are intended to be deployed in the cloud and several mobile devices, while delegating the majority of inherent tasks to the cloud. In addition, other processes such as installation, deployment, configuration, monitoring and management of software modules must be well modeled to fully provide the collaborative service to mobile users in the cloud. Indeed, managing an efficient virtualization ensuring continuity of collaboration in the cloud (especially for peer-to-peer groups) is a very difficult task. Indeed, the dynamic aspect of the groups (where virtual machines can join, leave or change virtual networks) and a vulnerability to failures can affect the collaboration. Thus, the offloading model should cover all steps of the complete lifecycle related to the deployment of software modules or virtual machines (namely, installation, configuration of network interfaces, management of group membership, fault detection and restoration) in a manner ensuring continuity of collaboration.

In this paper, we present cloud patterns for mobile data sharing that are regarded as specific design patterns used to describe mobile collaboration in the cloud. We provide an extensible model for implementing purely decentralized synchronization mechanisms in order to preserve the consistency of the shared resources under constraints of mobile applications, namely the short-life battery and the connection instability. Accordingly, design solutions are illustrated for addressing the challenge of modeling the cloning and collaboration services in the cloud. The proposed design patterns are for two main levels: the first one provides self-control for creating clones of mobiles, managing users' groups and ensuring the smooth functioning in the cloud platform. The second level presents group collaboration mechanisms for data synchronization in real-time, without any central role. As proof-of-concept, we present (i) the deployment middleware MIDBOX a platform for supporting mobile collaboration over a private cloud, and (ii) OPTICLOUD a cloud service for scalable real-time editing works in peer-to-peer mode.

Our model provides a set of abstract and interface classes that comprise the essential methods and parameters for implementing the required tasks for cloning mobile devices, managing peer-to-peer virtual private networks and sharing resources in the cloud. This gives developers a generic and abstract framework to develop mobile collaborative applications in the cloud. Reusability of this model implies a concrete redefinition of different methods of interfaces to adapt them to well-defined virtualization environments (i.e. cloud) in order to achieve flexible collaborative works. Note that MIDBOX and OPTICLOUD can directly be exploited by end-users via APIs that are derived from reusing our cloud patterns. Note that our cloud patterns have been reused to implement a cloud service-based platform for real-time collaborative editing works [4, 5]. In addition to network traffic and energy consumption criteria extensive experimental evaluation has demonstrated that this service enables us an enormous gain in response time and it scales very well.

This work aims to provide design patterns for offloading mobile collaborative applications in the cloud. The proposed architecture is distributed over two levels on the following environments: (i) Mobile Cloud Computing (MCC) which focuses on the mobile works offloading with additional tasks such as mobile data backup and restore, virtual networks management and fault tolerance. (ii) Collaborative environments that involve synchronization mechanisms for maintaining consistency of shared data. However, the heterogeneity of MCC environments that are based on different mobile and virtualization technologies (e.g. VirtualBox, XEN, Android, BlackBerry, etc.) is considered as a major concern for the modelling process. In addition, the diversity of collaboration forms (synchronous and asynchronous works) and the existence of several consistency preservation approaches (e.g., Operational Transformation (OT) [6, 7] and Conflict-free Replicated Data Types (CRDT) [8]) must be considered. To satisfy all these requirements, the offloading and collaboration models must be reusable and extensible. Accordingly, our contributions are related to MCC, collaborative environments and reusability. To our knowledge, this is the first effort aimed at defining reusable designs for cloud-supported mobile collaborative applications.

The remainder of this paper is organized as follows: Section 2 presents the global model and its requirements. Section 3 describes design patterns of the proposed architecture. In Sections 4 and 5, we illustrate respectively the implementation of MIDBOX (the deployment middleware) and OPTICLOUD (a collaborative editing application). We discuss the related work in Section 5 and conclude in Section 6.

2 Mobile Collaboration via the Cloud (MMC)

In this section, we present a model for Mobile Collaboration via the Cloud (MMC) to enhance mobile data sharing under limited resources constraints and we describe the design requirements underlying this model.

MMC Model. First of all, our model consists in offloading the most of mobile intense tasks (computation and communication) to the clone (i.e. virtual machine) in the cloud. Thus, users can expect to benefit from the cloud computing advantages for properly managing their uninterrupted collaborative works. This is done through Virtual Private Networks (VPNs) formed of clones of their mobile devices. Each user owns two copies of the shared data (e.g. text, calendar or friend list) with the first copy stored on the mobile device and the second on its clone (at the cloud level). The user modifies the mobile copy and then

sends local modifications to the clone in order to update the second copy and propagate these changes to other clones (i.e. other mobile devices). Moreover, the user can continue working even when disconnected (in offline mode) through the mobile device copy.

Model Requirements. Given the particularity of mobile applications constrained by limited resources and connection availability, strengthening mobile data sharing is tightly related to a specific requirements list such as: user's interaction flexibility (or scalability), user's interaction protection[a], communication, heterogeneity and interoperability, autonomous interaction-support services, user awareness, and data consistency [9]. Moreover, we must add another important requirement, namely the fault tolerance for managing recovery from failures. In the following, we enumerate requirements related to the proposed cloud-based design for reinforcing data sharing over distributed mobile collaborative applications:

- *Data consistency.* A collaborative application should be responsive in online and offline works. It means that frequent disconnections should not affect the consistency and availability of the shared data. Therefore, lightweight and decentralized synchronization mechanisms (based on explicit data replication between mobiles and their clones) are required.

- *Communication.* Synchronization for maintaining data consistency relies on message communication. However, ad-hoc peer to peer networks are expensive and suffer from frequent disconnections. To overcome this problem, intense communication tasks are pushed to the cloud. Our MMC model allows a direct communication for each mobile with its clone. On the other hand, clones form virtual networks and bear the most of communication burden.

- *Scalability.* Shared data availability and integrity should not be affected by the dynamic aspect of collaboration environments where users can create, join or leave groups and participate on-demand to collaborative work sessions. As we will see subsequently, our proposed MMC model is well suited for transparently achieving highly scalable mobile collaborative applications.

- *User awareness.* Collaborators must be able to share their real-time status information. Like [10], we use the following mechanisms: user's reachability (i.e. connected/disconnected) and user's availability (i.e. available/busy) in which each mobile user is notified via the clone on the presence/availability of other mobile users.

- *Heterogeneity.* Diversity of mobile devices and their operating systems is considered one of the major obstacles for mobile data sharing. Our model is based on a cloning middleware composed of web services for creating mobiles' clones with a unified operating system (i.e. Android OS). In this case, the clone/clone heterogeneity problem is eliminated. On the other hand, the mobile/clone heterogeneity problem is solved using communication based on standard SOAP (Simple Object Access Protocol [11]) through the cloning middleware. SOAP is a standard communication protocol. In comparing with REST architecture, it offers more flexibility for the implementation of mechanisms for exchanging messages between two heterogeneous nodes (eg, an Apple mobile that communicates with an android clone). Thus, SOAP

[a]The user's interaction protection is outside of the scope of this paper and we plan to include it in future work.

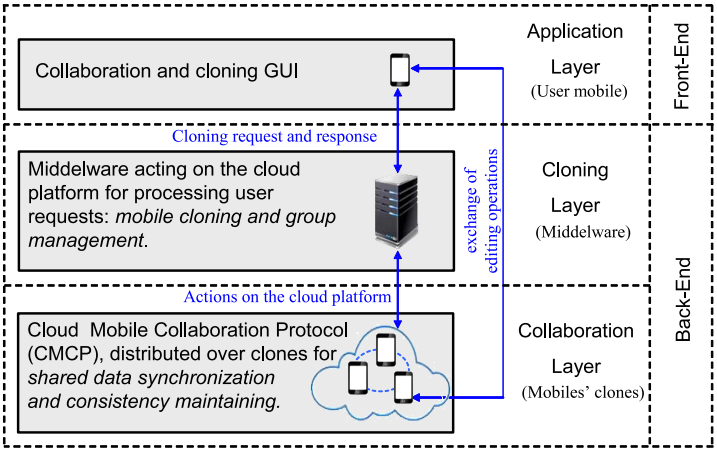

Figure 1 Architecture for mobile data sharing on the cloud

with its version 1.2, offers a specialized framework for the exchange of messages. But, this is an exception, as android devices can directly communicate with their clones[b].

- *Failure recovery.* Users have to recover easily all shared data when technical hitch (e.g. crash, theft or loss of mobile device, or clone failure) happens, and continue seamlessly the collaboration. To tackle this problem, designing a manager for the complete life cycle of clones is necessary. This manager will detect failed clones and restore their states without affecting collaborative tasks.

3 Design patterns for MMC

This section presents a reusable cloud-based model for the mobile data sharing. First, as shown in Figure 1, a global layered architecture for MMC is given. Next, cloud patterns for designing both main layers, cloning and collaboration, are described.

3.1 Layered Architecture

For resource limitation and frequent disconnection problems (e.g. leading to the collaboration interruption), the cloud is considered as a good solution to relieve mobile devices and ensure permanent bonds between collaborators. However, automating actions of a middleware for: (i) creating clones, (ii) managing dynamic groups through virtual networks deployed in the cloud and (iii) transparently dealing with failures is not a simple task. The problem raised at this stage is:

> **Issue 1:** How to devise an interface (Back-end) implementing a self-cloning mechanism to achieve previous actions (i)-(iii)?

After creating a new clone, it must be able to perform a self-initialization for beginning the collaboration phase. This clone must autonomously act for acquiring the required

[b]Note that, at present, our cloning middleware does not consider data backup from non-Android devices.

collaboration parameters (detailed below) to: (i) integrate the collaborative group and (ii) exchange (communicate) with its mobile device and other clones. Note that during the collaboration phase, concurrent access to shared data between clones can result in inconsistent views. Furthermore, an offset for applying requests between the clone and the real device is unavoidable. This is another concern:

Issue 2: How to devise protocols for maintaining consistency of shared data?

Therefore, the collaboration protocol (distributed over clones) must offer fully decentralized synchronization (clone/clone and mobile/clone) mechanisms without any central role for avoiding a single point of failure. On the other hand, the proposed model should be reusable for supporting any shared data type (e.g. document, video, table, ...).

To deal with **Issues 1** and **2**, Figure 1 presents services that are grouped in different layers where three main layers are considered. The application layer provides Graphical User Interfaces (GUI) for interacting with the remaining two layers. Interaction with cloning layer (the cloning middleware) allows for processing the "cloning and group management" users requests, whereas interaction with the collaboration layer enables user to start a synchronization between the mobile and its clone. The cloning and collaboration layers represent the "Back-End" part of the system and include reusable solutions (detailed in sub-sections 3.2 and 3.3) for **Issues 1** and **2**, respectively.

3.2 Cloning Patterns

The cloning solution consists of a middleware based on web services acting on the cloud platform (collaboration layer) for: (i) cloning mobile devices, (ii) managing virtual networks and (iii) supervising the smooth running of the clones. It should be noted that this solution is reusable in the sense that developers can redefine interfaces methods for adapting the cloning middleware to any virtualization environment (e.g. VirtualBox[c] and XEN[d] Figure 2 describes our design patterns of the Cloning Middleware that contains three parts:

(A) Cloning Engine. This solution allows implementing the cloning engine as web service for encapsulating its actions (see Figure 2 (part A)). The first action leads to creating a user profile by: (i) saving the introduced user information and (ii) generating the required clone parameters (i.e. the clone identifier, the clone IP address and the group identifier where the user intends to collaborate). Next, an optional backup is proposed to the user. Once this backup is over, the creation of a virtual machine based on Android operating system is launched. After this step it is necessary to configure the internal and external network interfaces to ensure an efficient and continuous communication between the mobile user, its clone and the remaining members of the group.

(B) VPN builder. Like the cloning engine, web services are used for implementing this solution (see Figure 2 (part B)). The VPN builder process will be triggered by a user request in order to create a new group. Then, the virtualization hypervisor will be started for: (i) building a new virtual network, (ii) activating a DHCP (Dynamic Host Configuration Protocol) server and (iii) assigning a specific broadcast address to this VPN.

(C) Failure manager. This solution is based on the heartbeat principle. The cloning middleware uses listeners for receiving periodic messages from clones (see Figure 2 (part

[c] `https://www.virtualbox.org/`
[d] `http://www.xenproject.org/`

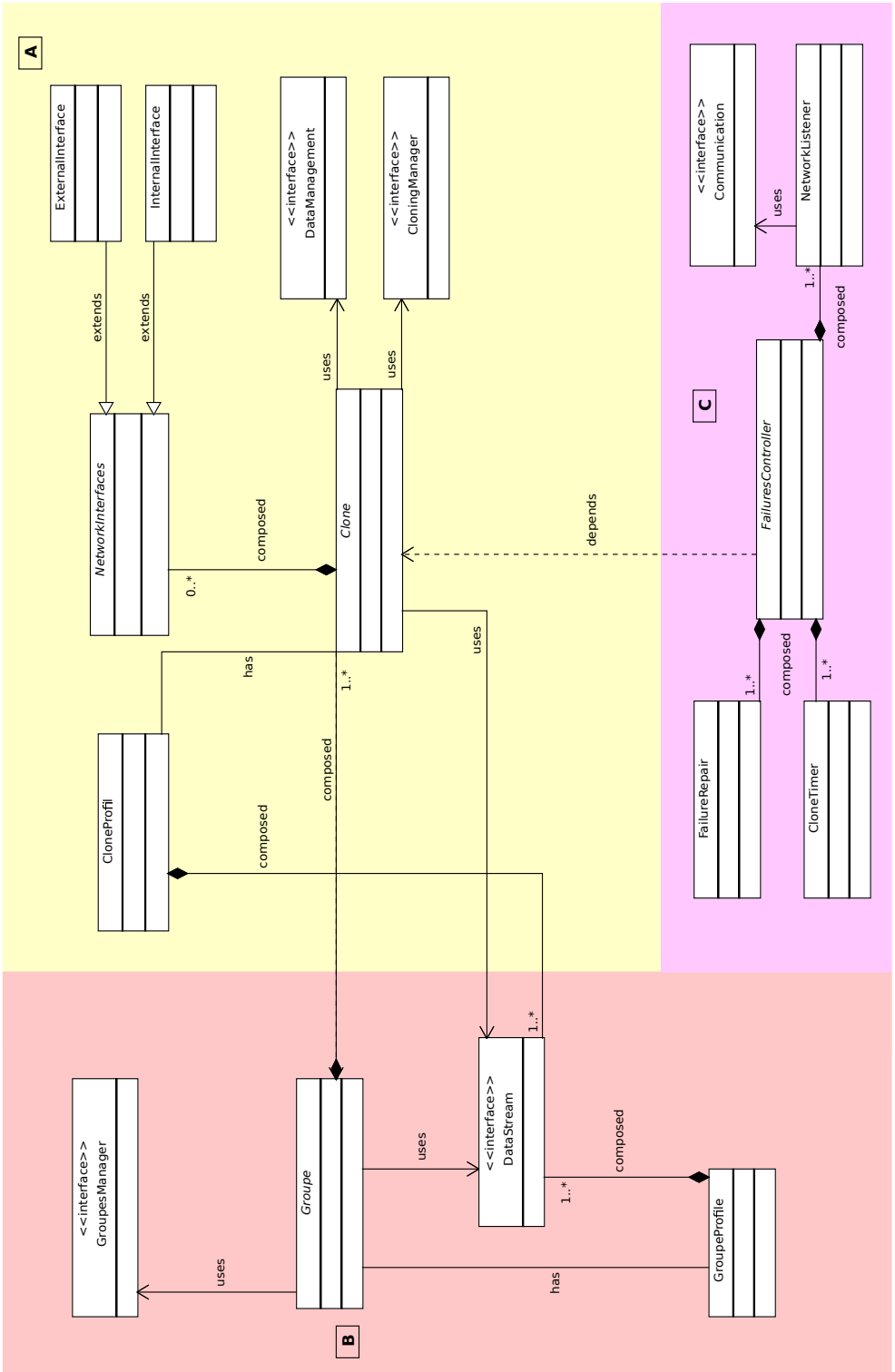

Figure 2 Cloning class diagram.

B)). Once a timer expires, the repair process is triggered by calling the cloning engine for creating a new clone and restoring all saved parameters related to the failed clone.

3.3 Collaboration Patterns

Our collaboration solution offers a protocol for sharing data in the cloud. It uses synchronization mechanisms (based on optimistic replication scheme [6, 7]) for enabling the reconciliation of divergent resource (or data) copies in a decentralized manner. For maintaining the consistency of the shared resources in real time, each clone must be simultaneously synchronized with the other clones and its mobile device. Figure 3 describes our design pattern of the the Collaboration Protocol that contains four parts;

(A) Clone integration. Each clone is pre-configured for starting automatically the collaboration protocol just after the first clone boot (see Figure 3 (part A)). Thus, the following initial integration tasks will be performed: the first action consists in calling the cloning middleware web services for requesting the previously generated parameters (i.e. user and group identifiers and broadcast address). After receiving and applying these parameters, a broadcast listening is launched for receiving requests from other clones. Next, the clone will proceed in initializing and updating the shared resources. Finally, this phase will end by sending a "*clone ready message*" to the Cloning Middleware. This message will be retransmitted to the user for reporting the clone eligibility.

(B) Communication. This solution is achieved by implementing network primitives for simultaneously listening and sending messages (see Figure 3 (part B)). It should be noted that these network primitives are provided in two modes: (i) the unicast mode for receiving/sending messages from/to mobile devices and (ii) the broadcast mode for receiving/broadcasting messages from/to other clones.

(C) Synchronization. This solution offers decentralized synchronization mechanisms for maintaining consistency and reconciling the resource copies divergence (see Figure 3 (part C)). Two synchronization steps are necessary for each clone. On the one side, the clone/clone synchronization can be performed using any decentralized synchronization technique [6, 7] by redefining methods of the "*CloneCloneSynchronization*" interface class. This will allow implementing different synchronization methods known in the literature (e.g. Operational Transformation [6, 7]). On the other side, the clone/mobile synchronization is ensured by a mutual exclusion-based method used between the mobile and its clone ("*HandleTokenSynchronization*" class that implements the "*CloneMobileSynchronization*" interface in the collaboration pattern, see Figure 3). The shared resource will be considered as a critical section when the mobile tries to commit/synchronize w.r.t its clone. Only one of them will have the exclusive right to access in synchronizing mode to its resource copy.

(D) Resource Management. This solution allows for creating and editing multiple types of simple or compound data resources (e.g. documents, images, tables, collections) (see Figure 3 (part D)). Developers can redefine the interface methods to adapt them with any type of data.

3.4 Patterns relationships

Figure 4 illustrates the relationships between the previously presented patterns graphically. These patterns are organized in a layered model with two layers. At upper layer, using

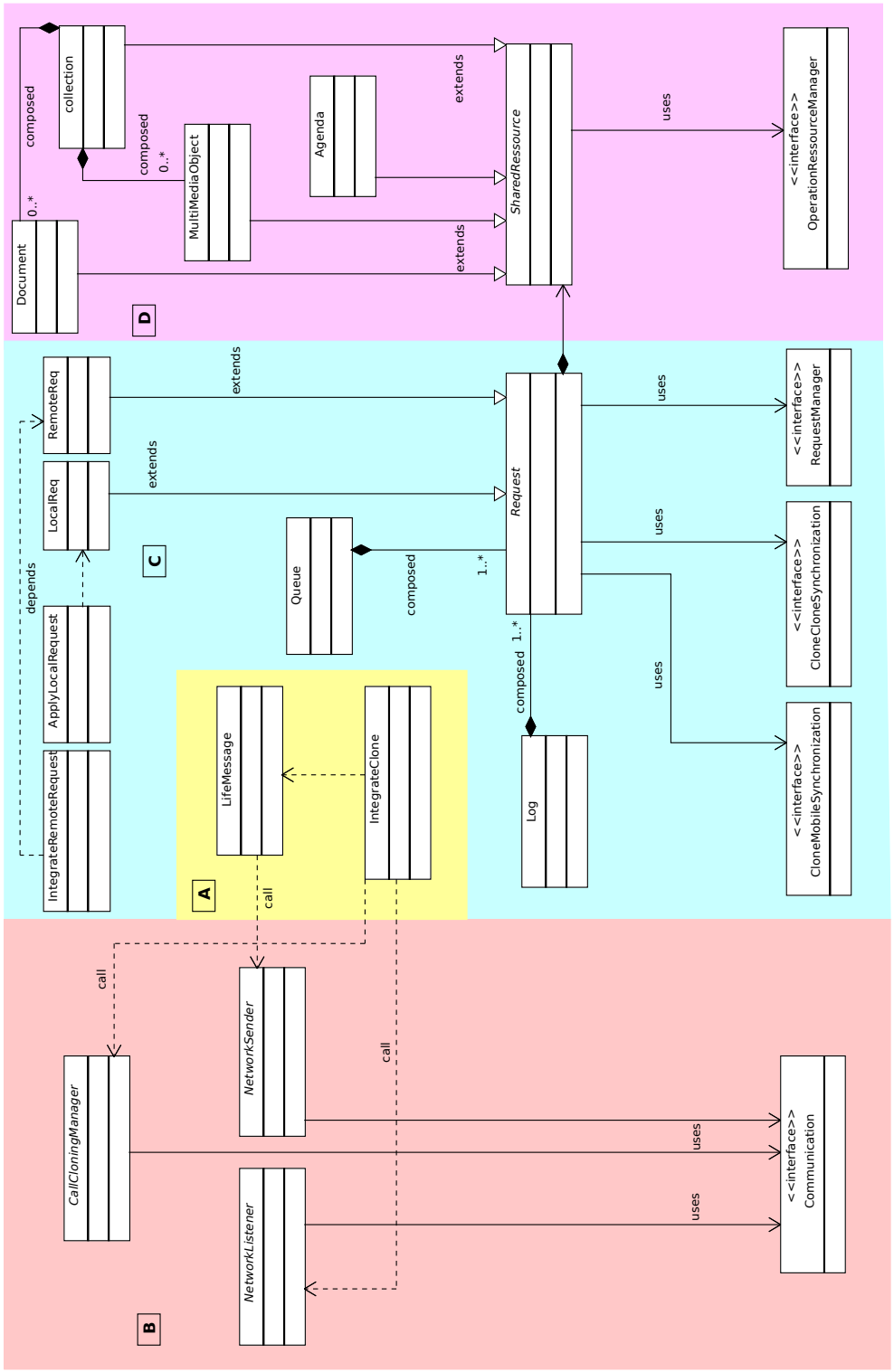

Figure 3 Collaboration class diagram.

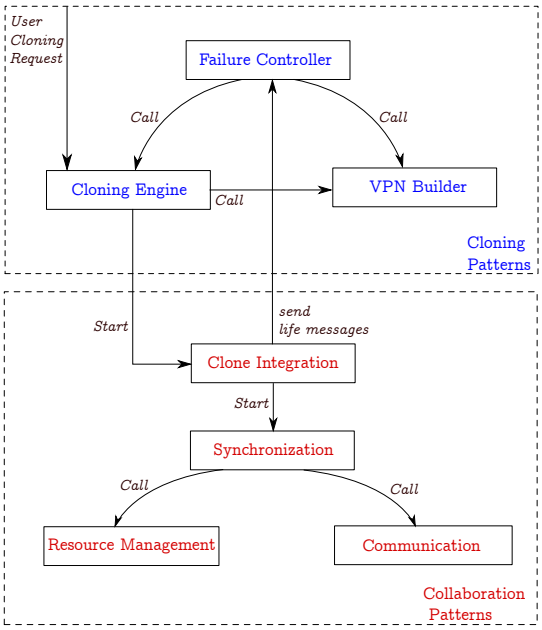

Figure 4 Patterns relationships

"*cloning engine*", a process is invoked when a user intends to clone the mobile device. This process can in turn appeal to the "*VPN builder*" for the construction of a new virtual private network. Both components "*cloning engine*" and "*VPN builder*" are under monitoring by "*Failure Controller*" for detecting and repairing any dysfunction related to clones or virtual networks.

At lower layer, any collaboration session is triggered by the cloning process (upper layer). Indeed, when a clone is created and activated, the component "*Clone Integration*" starts the synchronization processes after initializing all required collaboration parameters. Synchronizing a collaborative work needs means for mobile-to-clone and clone-to-clone interactions and shared resource updating that are ensured by components "*Communication*" and "*Resource Management*" respectively. On the other hand, the component "*Clone Integration*" is also responsible for sending life messages to "*Failure Controller*" in order to monitor the proper functioning of the clone.

3.5 Patterns vs. design requirements

Matrix shown in Figure 5 illustrates the requirements (presented in Section 2) of mobile collaborative applications satisfied by our cloud patterns. This matrix provides more flexibility in terms of development of based-cloud mobile collaborative applications. Indeed, developers can choose patterns that are suitable to their customized applications' requirements.

In the following, we illustrate how different patterns meet the requirements imposed by the design of collaborative applications in the cloud:

Data consistency. The "Communication", "Synchronization" and "Resource Management" patterns are designed to implement all ingredients of known consistency preservation approaches (e.g. OT [6, 7] and CRDT [8]).

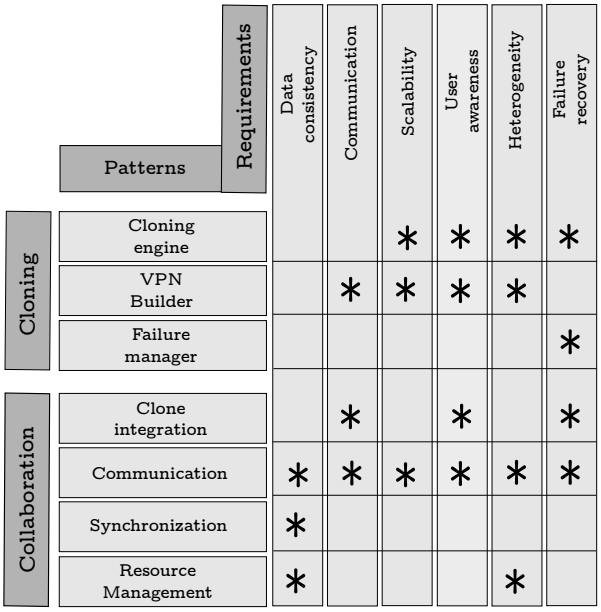

Figure 5 Patterns vs. design requirements

First, pattern "Communication" offers primitives used to exchange update operations (executed on different copies of the shared data) between clones and between the clone and its mobile device. This pattern is considered as a support for the synchronization of the performed updates.

Pattern "Synchronization" provides a general coordination framework to maintain consistency. Applied operations are exchanged via requests (by the "Request" class in the collaboration pattern) that may be local (i.e. generated by the mobile) or remote (i.e. coming from other clones). This pattern is based on the implementation of two main interfaces:

(i) The "CloneMobileSynchronization" interface. The implementation of this interface allows managing the synchronization between the clone and mobile. This mechanism is based on the permutation of tokens.

(ii) The "CloneCloneSynchronization" interface. It includes required methods for synchronizing updates between clones. For example, redefining the "IsReady" method enables to check if a remote request is ready to be executed. When it is not ready, it will be inserted in the waiting queue (the "Queue" class in the synchronization pattern). As for the redefinition of the "Transform" method, it is used to implement OT approach by transforming ready remote requests to include effects of previously applied requests. Thus, an overall consistency model is presented in Figure 6.

Communication. The "Clone Integration" pattern allows each new clone to acquire the necessary parameters for its communication with other clones in the same VPN. This virtual network is built by the "VPN Builder" pattern. As for the "Communication" pattern, it offers primitives for exchanging messages.

Scalability. The "Cloning Engine" and "VPN Builder" patterns (via their "CloningManager" and "GroupsManager" interfaces) offer a set of virtual groups management services, where users can create, join, change or leave VPNs.

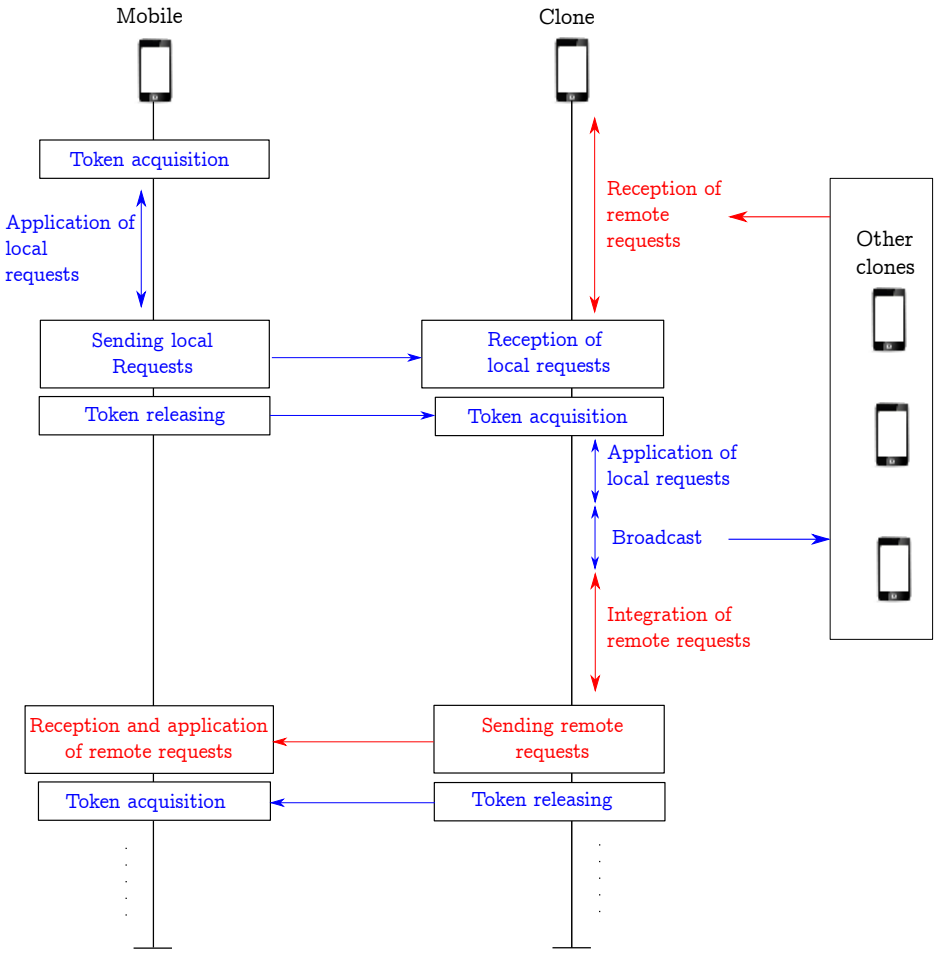

Figure 6 Consistency model.

User awareness. The "Cloning Engine", "VPN Builder" and "Communication" patterns participate in notifying users through the clones by detecting the participation/absence of user in collaboration works.
Heterogeneity. The "Cloning Engine", "VPN Builder", "Communication" and "Resource Management" patterns provide an abstract level for defining clones, virtual networks and shared resources. This level can be adapted to multiple heterogeneous environments of "MCC" by implementing the interfaces of such patterns.

4 MIDBOX: a cloning deployment middleware

In line with our overall goal to alleviate mobile devices by delegating the most intense tasks to their clones, this section is dedicated to the presentation of a specified design pattern of our cloning deployment middleware, called MIDBOX. The class diagram of this middleware shown in figure 7, is derived from reusing the architecture of mobile collaborative applications in cloud environments, presented in the previous section. It aims

to extend the basic architecture and offer a set of packages containing a set of classes (coded in Java language) for implementing the different interface patterns.

The objective of MIDBOX is to design, develop and implement a deployment middleware, acting on the VirtualBox[e] virtualization hypervisor (i.e. private cloud), to build a virtual collaborative platform mainly composed of mobile devices' clones and private networks. The different packages of this extension are highlighted with a gray background color in Figure 7.

The added classes implement the different interfaces by redefining their basic methods in order to adapt them to the VirtualBox virtualization hypervisor. For example, method *CreateAndroidVM* redefines method *CreateClone* of the *CloningManager* interface to create a new Android virtual machine on VirtualBox (see Figure 7). Accordingly, these classes are grouped into the following main packages (see Table 1):

1. *MidBox.Cloning*. This package is dedicated to managing the lifecycle of Android virtual machines on VirtualBox. It implements the cloning process and includes clone creation, network configuration, start-up and integration of clones to their groups. Note that the autonomous processes monitoring clone failures (see Figure 7 (C)) are considered as complement to the cloning process and they help in managing the full life cycle of each clone. Indeed, the failure controller ensures the smooth functioning of the system as a whole via a continuous observation process leading to the detection of different failure situations. In the case of such detection, this controller calls the present package classes to create a new instance of the failed clone.

2. *MidBox.Group*. This package is used to handle virtual private networks in which many Android virtual machines can be grouped. This allows to build VirtualBox "Host-Only" VPNs type, and generate an IP address range and a specific broadcast address via a setup and activation of a DHCP server dedicated to each VPN.

3. *MidBox.Data*. This package includes classes for backup and restoration of data between mobile devices and their clones.

4.1 Cloning process

The cloning process can be triggered via different proposed web services (see Figure 8): (i) *subscribe(userName, PW*[f]*, mail, MobileIP, GroupChoice)* allows the user to clone her/his mobile device and join a collaborative group, (ii)*changeGroup(ID-OldGroup, ID-NewGroup)* provides a service to change a group, and (iii) *leaveGroup(ID-Group*) enables the user to leave a group. In the case of the subscription web service, the cloning process proceeds by the following steps (blue tasks bounded by blue dotted lines in Figure 8):

(1) Generation of clone parameters. This action is achieved by instantiating the *GenCloneParams* class that implements the *CloningManager* interface. Generated parameters will be gathered into a single object that is issued from instantiating the *UserProfile* class and these parameters are from two sources. A part of the profile data is introduced by the user as input parameters of the *subscribe* called web service, namely: the user name, password, e-mail and the IP address of the mobile device. The other part of parameters concerns the clone and is automatically generated during this phase of the

[e] `https://www.virtualbox.org/`
[f] Password.

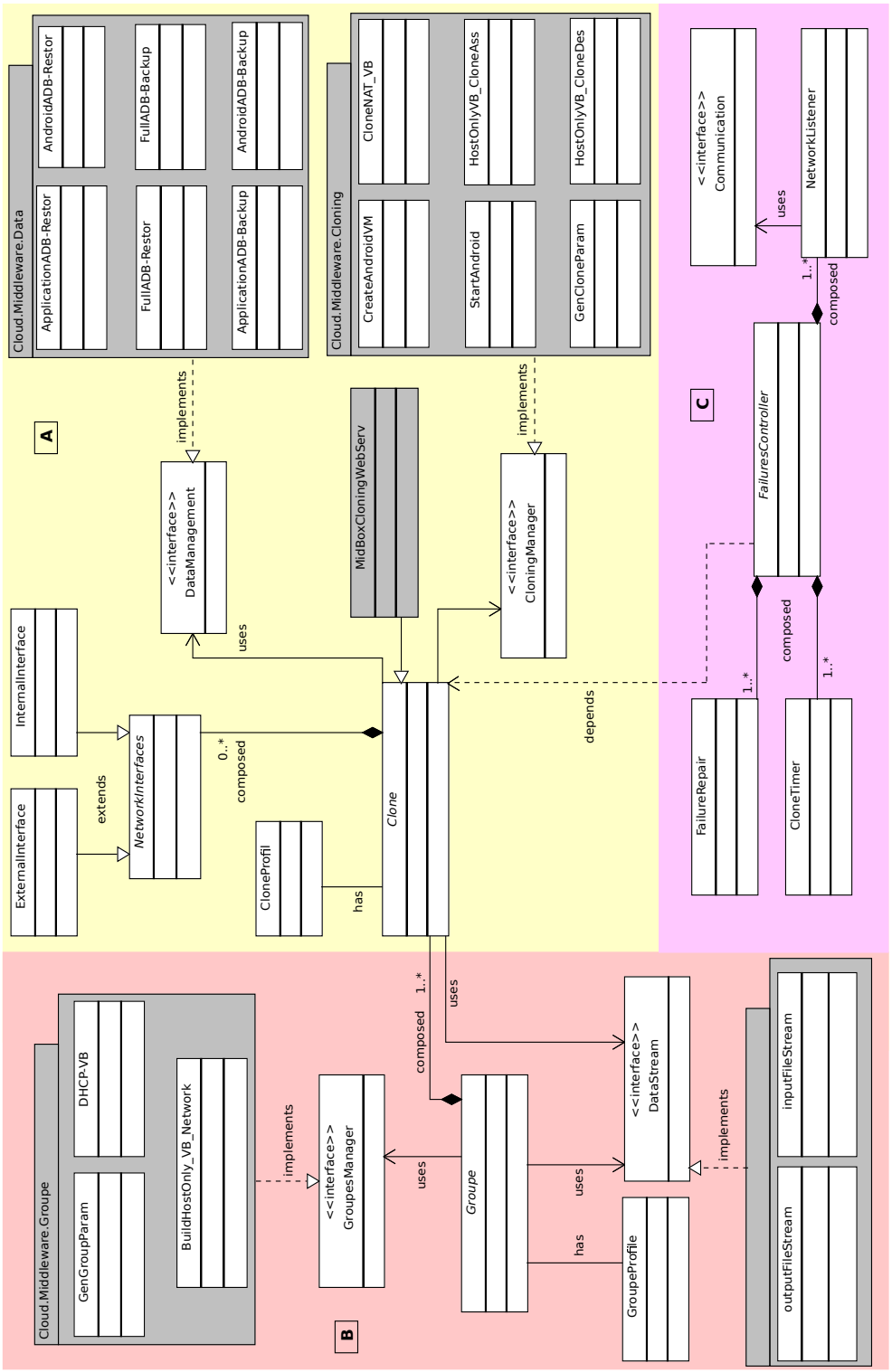

Figure 7 MIDBOX class diagram.

Package	Classes	Description
MidBox.Cloning (Implementation interface: *CloningManager*)	*CreateAndroidVM*	create Android Virtual machine
	GenCloneParam	generate clone parameters
	CloneNAT_VB	associate clone to external network
	HostOnlyVB_CloneAss	associate clone to a VirtualBox VPN
	HostOnlyVB_CloneDes	desassociate clone from a VirtualBox VPN
	StartAndroid	start Android VM (clone)
MidBox.Group (Implementation interface: *GroupesManager*	*BuildHostOnlyVB_Network*	build a Host-Only VirtualBox VPN
	GenGroupParam	generate goup parameters
	DHCP-VB	activate a DHCP server for VPNs
MidBox.Data (Implementation interface: *DataManagement*)	*ApplicationADB-Backup*	backup applications from mobile to clone
	AndroidADB-Backup	backup Android OS from mobile to clone
	FullADB-Backup	full backup from mobile to clone
	FullADB-Restor	full restore from clone to mobile
	ApplicationADB-Restor	restore applications from clone to mobile
	AndroidADB-Restor	restore android OS from clone to mobile
MidBox.Stream (Implementation interface: *FileStream*)	*InputFileStream*	used to retrieve objects describing clones and groups profiles
	OutputFileStream	used to save objects describing clones and groups profiles

Table 1 Description of the MIDBOX implementation packages.

process (such as ID of the clone, the virtual machine name, and the VPN serial number) or provided by the clone after its start (such as IP address of clone). Note that the VPN serial number serves for referencing the membership of the clone. Once this phase is completed, the user will be asked to confirm its agreement for the mobile data and applications backup.

(2) Backup of mobile data. Backup of mobile data and applications is optional. Different backup types are available to the user: full backup, mobile applications/data and android system that are respectively the instantiation of the classes *FullADB-Backup*, *ApplicationsADB- Backup* and *AndroidADB-Backup*. These classes implement the *DataManager* interface. In case of a full backup, the created clone will be an exact copy of the real mobile device.

To provide this backup service we have used the "*Android Debug Bridge*" (*ADB*)[g], which allows users to communicate with emulators or Android devices through lines command. It is a client-server program that offers tools for deploying/retrieving data to/from mobile devices. Classes of the *MidBox.Data* package, that implement the *DataManager* interface, define processes that are based on these ADB commands.

(3) Creation of the Android VM. For each mobile device, MIDBOX builds a new Android x86 virtual machine in VirtualBox. In our approach, instead of creating this virtual machine from scratch using an IZO file, we choose to import a pre-configured virtual machine. This choice is motivated by the fact that the deployment of clones will be easy and fast. The collaborative application is also incorporated in the virtual machine. The latter is equipped with a pre-configuration for the automatic execution of the collaborative

[g] `http://developer.android.com/tools/help/adb.html`

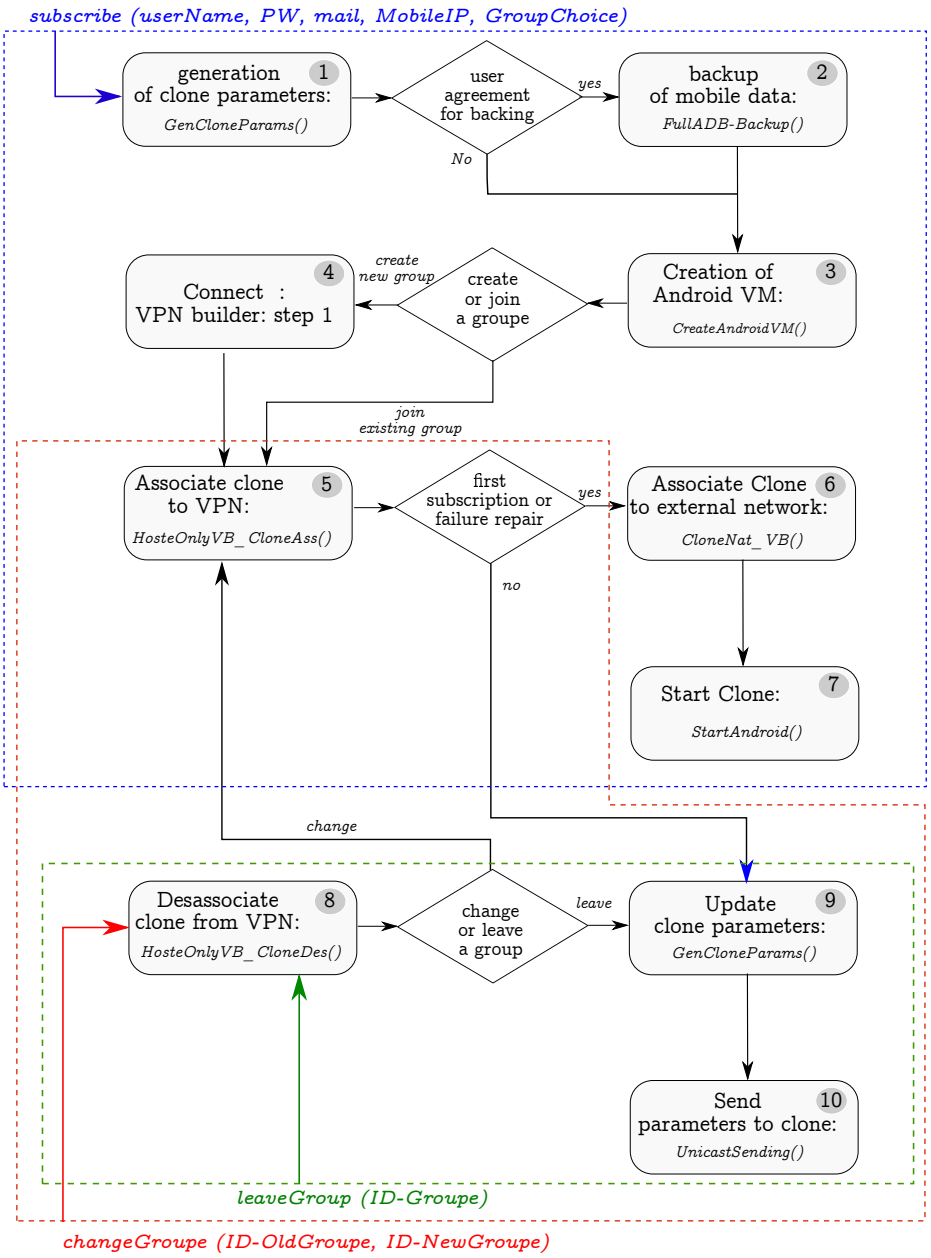

Figure 8 Actions diagram of the cloning process.

application with the first start of the clone. Note that the newly created clone requires a network configuration. This will set the virtual network interfaces (i.e., virtual network cards) that can be activated and associated with different types of virtual networks provided by VirtualBox. This creation is achieved by instantiating the *CreateAndroidVM* class that implements the *CloningManager* interface.

(4) Connect: VPN Builder. To associate a network interface of a clone to a VPN, the cloning process takes into consideration the choice of belonging to a collaboration group, given by the user as a call parameter of the cloning web service. In the case of creating a new group, and before proceeding to the association phase, the cloning process uses the VPN builder (by connecting at its first phase) to build a new VPN which will then be associated with a clone network interface.

(5) Associate clone to VPN. In the case of belonging to an existing group, the cloning process ignores the previous phase. It instantiates the class *HosteOnlyVB_CloneAss* that implements the *CloningManager* interface for associating a network interface to an existing VPN. This VPN corresponds to the user choice introduced as an input parameter when calling the *subscribe* web service.

(6) Associate clone to external network. After completion of the previous phase, the cloning process will proceed to the association of another clone network interface to an external network allowing access from the outside (from the real mobile device). This association is achieved through an instantiation of the *CloneNAT_VB* class that implements the *DataManager* interface. This enables to define port forwarding rules used by a NAT (i.e, Network Address Translation) network type. These rules are based on IP address translation mechanism. With this network mode, VirtualBox is constantly listening on certain ports sources that are defined in the redirection rules. Following the identification of the transmitter by its IP address, VirtualBox redirects the received packets to the target virtual machine (i.e, clone) identified by its IP address and the destination port that are also defined in these port forwarding rules.

(7) Start clone. This action starts the created virtual machine, reset its network interfaces and auto-start the collaborative application. A confirmation message is then sent to the user who can connect to mobile clone. Note that the new instance of the deployed clone has a pre-configuration allowing it a self-reaction just after its startup. It will call MIDBOX to provide its IP address and retrieve the necessary parameters for the collaborative works. This action is considered as an initialization task for integrating the new clone to its collaboration group in the cloud.

4.2 Implementation of MIDBOX

The deployment of middleware MIDBOX is considered as an interface between users and mobile collaborative applications that are deployed in the cloud. We have used the Axis2 Web server[h] deployed in the Tomcat web container[i] to implement the different services of our deployment middleware.

As shown in Figure 9, a cloning web service is provided to enable a user's online adhesion via a web interface. This service prompts the user to enter her/his profile data (Part A1 in Figure 9) and choose of belonging to an initial group collaboration (Part A2). As already mentioned, this information corresponds to the input parameters of the subscription web service. This cloning web service is deployed in Axis2 web server and is based on the standard WSDL (Web Services Description Language [12]). Figure 10 shows a part of the WSDL file describing the different parameters of this web service, namely:

[h] `https://axis.apache.org/axis2/java/core/index.html`
[i] `http://tomcat.apache.org/`

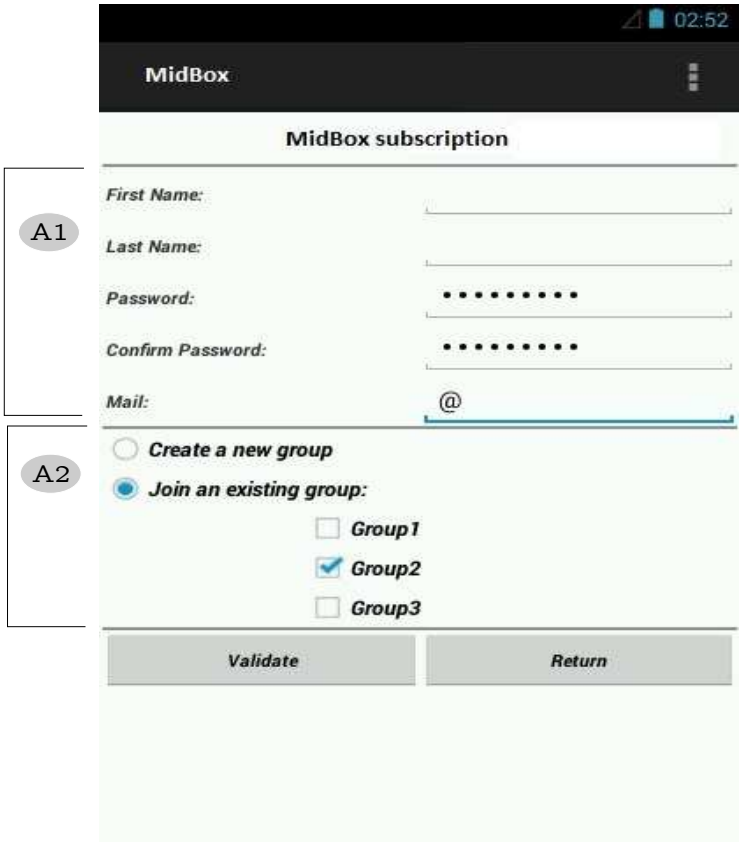

Figure 9 Subscription interface to the cloning service via MIDBOX.

B1: The cloning web service address involves the deployment path of the Web service. In the case of the implementation shown in Figure 10, this service can be called from the Axis2 web server via port 8080 of the local machine (i.e. localhost:8080).

B2: Name of the method that implements the different phases necessary for the cloning process. This method is called "*insert*" and it includes all the deployment actions which are applied to the the VirtualBox virtualization hypervisor (i.e. private cloud) in order to create clones and configure their network settings.

B3: The cloning web service input parameters are user-supplied and mainly include the user name, password, email address, and the choice of the initial collaborative group.

B4: The cloning web service response is returned to the user after completion of the cloning process. This response invites the user to connect to the clone of her/his mobile device in order to manage mobile data and participate in collaborative works with other members of her/his group.

On the other hand, Figure 11 presents the cloning process progress in responding to a user request (using Tomcat web container console):

C1: Creating a new Host-Only VirtualBox network as a VPN designed to collect and communicate clones of the same group collaboration.

Figure 10 WSDL file encapsulating the cloning process.

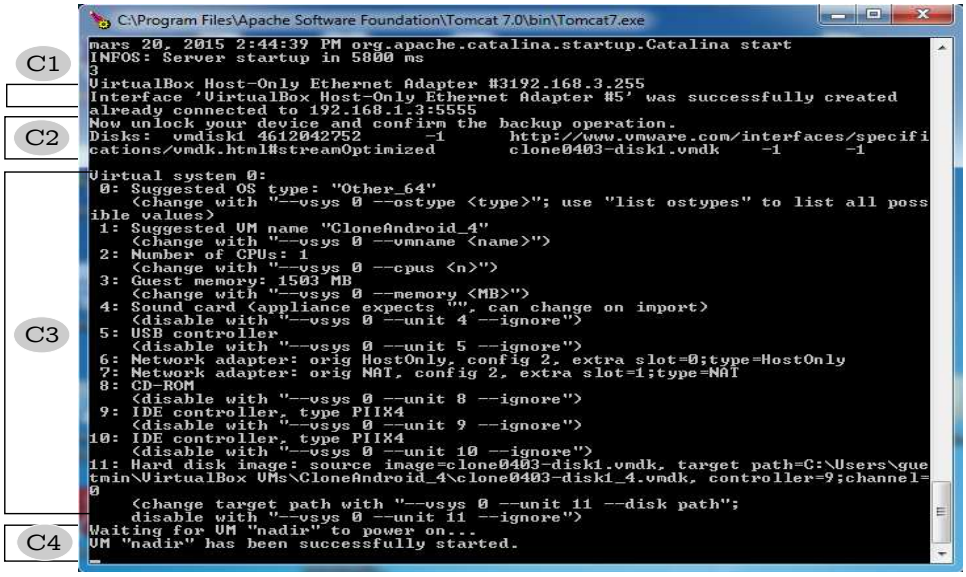

Figure 11 Execution progress of the cloning process on the Tomcat web container console.

C2: Mobile data backup.
C3: Setting-up of a new clone (Android virtual machine) on VirtualBox.
C4: Starting the clone.

The following listing presents the *MidBoxCloningWebServ* web service (see Figure 7) with the subscription method *insert* for responding to a user cloning request.

```
1  package MidBox.Cloning;
2  //Import Section
3  public class MidBoxCloningWebServ implements Serializable{
       //web subscription service class, called
       MidBoxCloningWebServ.
4  private static final long serialVersionUID = 1L;
5      public String insert(String u2, String mp2, String mail2,
           int choix2, String ip2){ // method of subscription web
           service, called "insert" with its input parameters
6
7      //Local parameters declaration...
8
9      //Clone parameters generation
10     GenCloneParam GPC = new
           GenCloneParam(u2,mp2,mail2,choix2,ip2);
11     GPC.GenCloneParams(u2,mp2,mail2,choix2,ip2);
12
13     //Mobile data backup
14     if (accord="yes"){
15     ADB_Connexion ADB_con = new ADB_Connexion(ip2);
16     FullADB_Backup save=new FullADB_Backup();
17     ADB_con.connexion(ip2);
18     save.FullBackup();
19     }
20
21     //creation of a new Android virtual machine
22     CreateAndroidVM newClone = new CreateAndroidVM()
23     newClone.CreateClone();
24
25     //Belonging choice to a group collaboration
26     if (choix2!=0) idgr=choix2;  //join an existing group
27                 else if (choix2==0) //create a new group
28
29                 {
30                 //call the Create_Groupe web service ...
31                                 }
32
33     //Asoociation to VPN configuration
34     HosteOnlyVB_CloneAss intern= new
           HosteOnlyVB_CloneAss(id_groupe, Clonename);
35     intern.AssociateCloneVPN(numVPN, CloneName);
36
37     //Configuration of translation IP addresses parameters
38     CloneNat_VB extern = new CloneNat_VB(numVPN, nomClone);
39     extern.CloneExternNetwork();
40
41     //Starting the new created clone on VirtualBox
42     StartAndroid launch = new StartAndroid(CloneName);
43     launch.StartClone(CloneName);
44     }
45 }
```

5 OPTICLOUD: a collaborative editing application

In this section, we present OPTICLOUD an architecture for mobile collaborative editing works in the cloud [4, 5]. On the mobile side, the energy consumption is largely reduced since all procedures for maintaining consistency of shared documents are executed on the clone side.

5.1 Design of OPTICLOUD

Figure 12 shows the architecture resulting from the reuse of our collaboration design pattern presented in Section 3.3. This architecture is part of development of a mobile application for collaborative editing of documents under mobile constraints (e.g. energy and connectivity). The collaboration protocol distributed through clones is inspired from Optic protocol [7]. It uses Operational Transformation (OT) approach to preserve the consistency of the replicated document [6, 7]. To this end, extension classes included in the added packages (shown with a gray background color in Figure 12) are used to redefine different interfaces' methods in order to adapt them to the OT approach. Given the importance and the key role of the added packages to the synchronization pattern (see Figure 12 (C)), tasks underlying to this synchronization are described as follows:

Requests management. Synchronization between clones is based on the exchange of user requests (e.g. adding or deleting text lines). In this context, classes of the *OptiCloud.ReqManager* implement the *RequestManager* interface to redefine its methods for generating, coding and interpreting requests. A query generation is a step that follows the application of a local operation and it consists in producing a *Request* composed of fields (or attributes): clone ID, group ID, operation serial number, the operation itself (e.g. insert or delete) and the dependency field of this operation to enforce causal order between requests [7]. The operation object is derived from the implementation of the $OperationResourceManager$ interface by classes of the $OptiCloudEditor$ package (see Figure 12 (D)). Once created, the clone will proceed to encode the request in the form of a message composed of strings.

The message issued from the request coding will be propagated and therefore received by other clones using broadcast primitives ($ServerBroadcastThread$ and $ClientBroadcastThread$) of the package *OptiCloud.Communication* that implements the *Communication* interface (see figure 12 (B)). Note that, as part of performance improvement in terms of latency and network traffic, we have decided to choose dynamic representation of messages where different fields with variable size (e.g, the serial number of a query) are followed by a special character indicating their end limit. On the other hand, after receiving a message, the clone will proceed with mining the request that message contains. Figure13 shows an illustrative example for the management of requests by a clone using OPTICLOUD.

Clone-Clone synchronization. For the clone/clone synchronization with our OPTICLOUD protocol, we choose to implement the OT approach[6, 7], generally defined for linear data structures (e.g. text documents). Classes of the package *OptiCloud.CCS* that implement the *CloneCloneSynchronization* interface allows redefining its different methods according to the OT approach. This mechanism is essentially based on two steps:

(i) After reception and interpretation of remote request, the receiver clone stores this request in the operation queue and verify its eligibility integration. This request will be

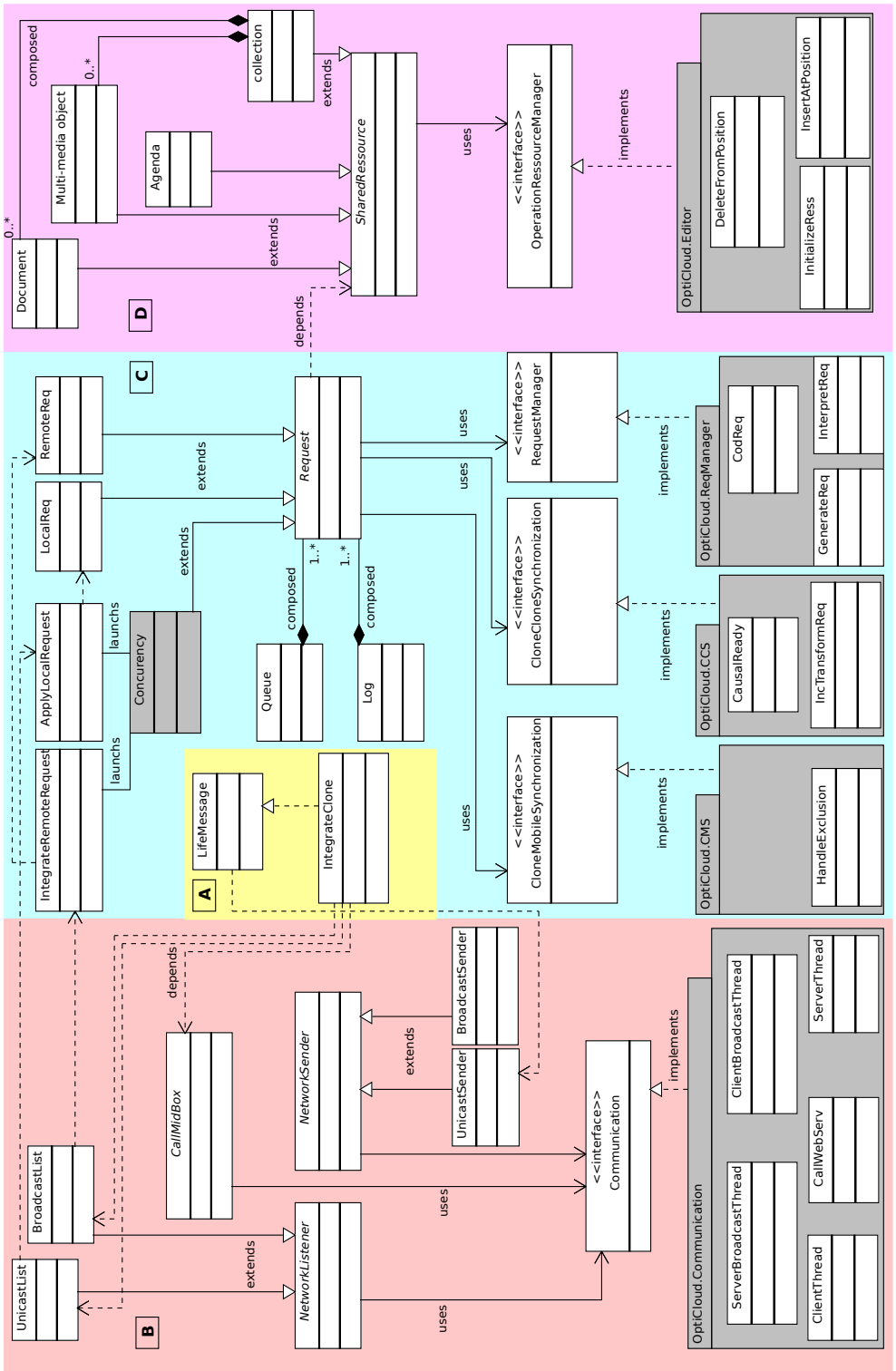

Figure 12 Collaboration class diagram for OPTICLOUD.

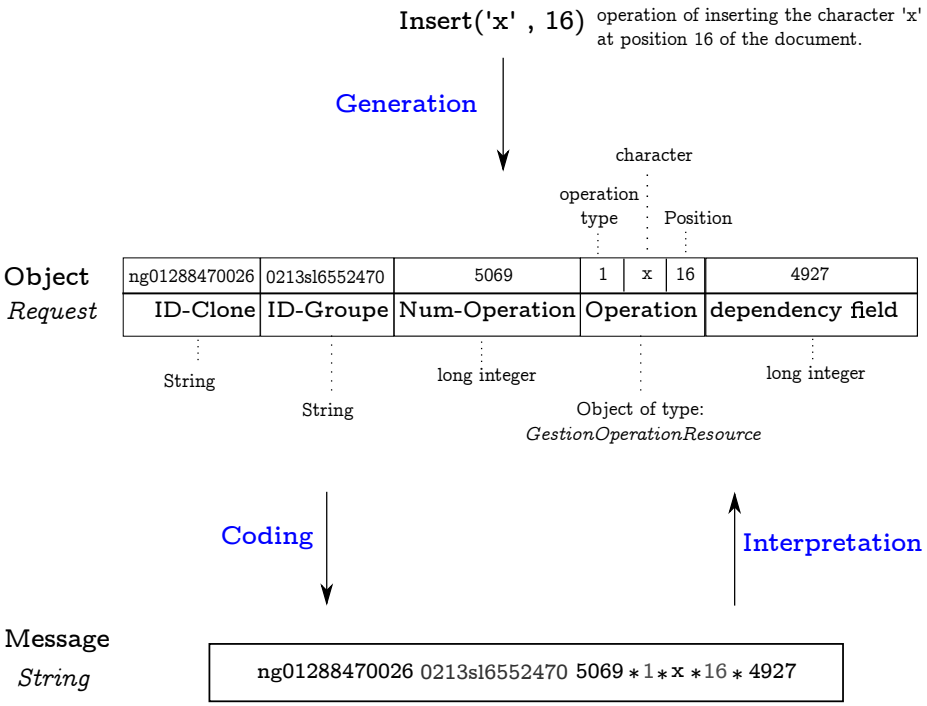

Figure 13 Requests generation with OPTICLOUD

dequeued when it is causally ready (i.e, the precedent request has been seen by the clone). For this purpose, it uses the class $CausalReady$ of the *OptiCloud.CCS* package.

(ii) When the request is causally ready, a forward computation is carried out on the operation log to maintain the consistency of the shared document. More precisely, using an OT function, the remote request is transformed against all concurrent (or not already seen) requests. This transformation will preserve and include effects of previously applied requests. To this end, the clone instantiates the class *IncTransformReq* of the *OptiCloud.CCS* package.

To better understand the role of operational transformation, consider the following example: given two clones, CLONE1 and CLONE2, starting from a common state of the document "XYZ". At CLONE1, a mobile user executes op-CLONE1 = $Ins(2, A)$ to insert the character 'A' at position 2 and end up with "XAYZ". Concurrently, a user at CLONE2 performs op-CLONE2 = $Del(1)$ to remove the character 'X' at position 1 and obtain the state "YZ". After the operations are exchanged among CLONE1 and CLONE2, if they are applied naively both clones get inconsistent states: CLONE1 with "AYZ" and CLONE2 with "YAZ". Operational transformation is considered as safe and efficient method for consistency maintenance. In general, it consists of application-dependent transformation algorithm, called IT, such that for every possible pair of concurrent operations, the application programmer has to specify how to integrate these operations regardless of reception order. Thus, at CLONE2, operation op-CLONE1 needs to be transformed to include the effect of op-CLONE2: op-CLONE1' = IT(op-CLONE1,

op-CLONE2)=$Ins(1, A)$. As for CLONE1, operation op-CLONE2 is left unchanged. Accordingly, both get the same state "AYZ".

Clone-Mobile synchronization. Mobile and its clone are two entities that are physically separated. Therefore, a delay of applying the same operations on both sides with the same state is possible. This may lead to document inconsistency. To overcome this problem, we propose a solution based on distributed mutual exclusion between the mobile and its clone. The shared document will be considered as a critical section (CS) when the mobile tries to commit/synchronize w.r.t its clone. Only one of them will have the exclusive right to access in synchronizing mode to its document copy. This distributed mutual exclusion protocol is achieved by the exchange of messages (i.e., token). Initially, the mobile device has the right to be the first to commit/synchronize with its clone. Whatever where the exclusive access right is, the mobile device and its clone can edit independently their local copies. The clone continues to receive remote operations from other clones to integrate them later on the local state. At the meanwhile, the mobile user can work on its copy in unconstrained way. But, once she/he decides to synchronize with its clone, all local editing operations are sent to its clone and the exclusive access right is released to enable the clone to start the synchronization with the mobile device. Thus, the clone performs the received operations on its local state, includes by transformation their effects in its local (and not seen by the mobile) operations, and sends the resulting operations to the mobile device in order to integrate them.

The *HandleExclusion* class that implements the CloneMobileSynchronization supports this synchronization between the clone and mobile.

5.2 Experimental analysis

In order to evaluate our OPTICLOUD application, we have compared it with Wikipedia, the famous collaborative editing system. Gains of energy and network traffic (mobile side) are measured. Each result is calculated by the average of ten repeated tests. For the energy consumption measurement, we have used PowerTutor [j] application. It allows measuring energy consumed by mobile running applications in real time. To measure the network traffic, we have used Wireshark[k] application, which allows detecting different available network interfaces and measuring network traffic. For mobiles cloning and preparation of the cloud collaboration platform we have implemented our cloning web services using the Apache Tomcat7 Web server [l].

To achieve this, we implemented and compared two systems. The first one is OPTICLOUD. It is composed of three mobiles that possess their clones in the private cloud (Virtualbox in our case). The three clones form a virtual peer-to-peer network and perform the total treatments for maintaining shared document consistency. The second system is composed of three separated mobiles which access and edit simultaneously the same Wikipedia page. Note that for random generation of edition operations with Wikipedia, we used JAVA API[m] for Wikipedia.

[j] http://ziyang.eecs.umich.edu/projects/powertutor/
[k] https://www.wireshark.org/
[l] https://tomcat.apache.org/
[m] https://code.google.com/p/wiki-java/

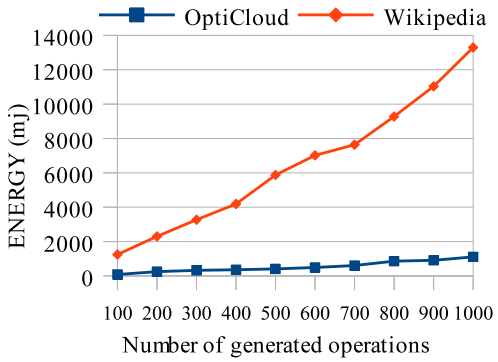

Figure 14 Energy consumption during the editing document phase with OptiCloud and Wikipedia.

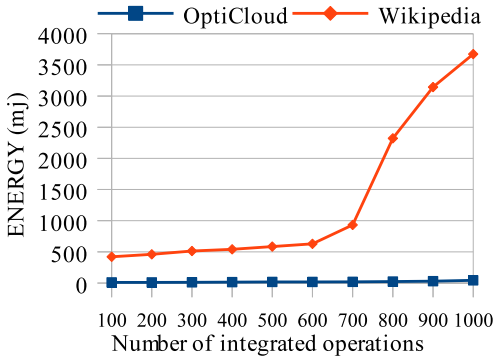

Figure 15 Energy consumption during the reconnection and update local state with OptiCloud and Wikipedia.

5.2.1 Energy

For the comparison of energy consumption, we tested the two systems over two phases: The first phase concerns the editing of the OPTICLOUD-document/Wikipedia-page, that initially contains 3000 characters. Each mobile executes a random generation script of edition operations which vary between 100 and 1000. As shown in Figure 14, the energy consumed by mobile, at Wikipedia side, is important. This important consumption is due to validation of executed operations in Wikipedia that consists in sending the whole page (even though it is altered by only one update).

The second phase concerns the document update after user reconnection. For this phase, we have observed that the energy consumption in Wikipedia side is considerable as illustrated in Figure 15; this is because of the expensive downloading of the entire Wikipedia page.

5.2.2 Network traffic

For the comparison of network traffic, each mobile executes a random generation script of edition operations which vary from 100 to 1000. The initial OPTICLOUD-document

/Wikipedia-page contains 3000 characters. We tested the two systems according to two worst cases. The first case consists in performing 100% insertion operations and the second one in executing 100% percent of delete operations. As shown in Figure 16, network traffic with OPTICLOUD is negligible compared to Wikipedia. This traffic is proportional with the size of Wikipedia page. It is huge in the case of large pages, even after validating a single edition operation. With OPTICLOUD, updating documents consists in sending only performed operations.

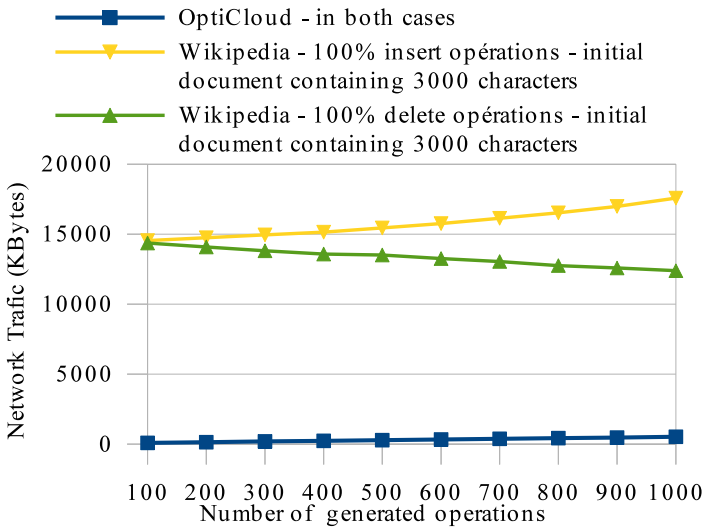

Figure 16 Network traffic with OptiCloud and Wikipedia.

6 Related work

In this section, we review the most important works done in the context of our proposal based on four facets: (i) design patterns for cloud computing, (ii) design patterns for mobile collaborative applications, (iii) cloud offloading and (iv) real-time collaboration.

Design patterns for cloud computing. In [13], authors provide a standard catalog-based design patterns for the development of applications running in the cloud. This catalog can be used to describe cloud computing service models and application architectures in the cloud. Thus it allows users to select cloud providers that are adapted to their custom applications. Design patterns are spread over two main layers: (i) a load balancer that is responsible for the distribution of treatments, and (ii) a component manager that manages the elastic scalability and fault tolerance. While design patterns given in [13] are important, it does not take into account the peculiarity of MCC (i.e. limitation of mobile resources and offloading) and maintaining consistency for the real time collaboration.

In [14], authors give an overview of existing works around cloud patterns that are specific design models related to cloud computing. Cloud vendors [15, 16] have published *specific patterns* that are strongly linked to the proprietary platform where they can

be deployed. In [17], *generic patterns* are provided regardless of any particular cloud environment. Both kinds of cloud patterns are interesting but their development is still in early stage and no models or reusable architectures support precisely MCC.

Design patterns for mobile collaborative applications. The area related to the mobile data sharing has experienced several research works but few works have presented a reusable model for designing collaborative mobile application. Furthermore, to our knowledge, there is no work that proposed a reusable model specifically designed for MCC.

In [18], a general architectural model was introduced to help developer for designing mobile applications. The goal of this model is to abstract high-level structural components of mobile applications while providing patterns to refine these components. This work does not consider collaboration across mobile devices constrained by their resource limitations.

A study on the intersection of two areas: software architecture and mobility, was presented in [19]. Since this work has borne more interest on the architecture of single-user mobile applications, it is not suitable for mobile data sharing in the cloud.

Based on a global reference architecture for collaborative systems [20], CLOVER is an architecture comprising a set of classes corresponding to different services that are required by a groupware application. This work does not consider the mobility and limitation of mobile resources constraints.

The work in [21] presented a reusable architecture for instantiating mobile workspaces, but resource limitation constraints was not considered. This architecture has been reused for the implementation of three mobile collaboration frameworks: building inspections, hospital work, and urban emergency responses.

In [22], a reusable structural design for mobile collaborative applications was presented. This model is mainly based on coordination and communication services. However, mobile constraints previously described (i.e. short-life battery and connection instability) were not considered by this design. This may negatively impact on such collaboration systems that are not intended for cloud environments.

An aid framework composed of contextual elements which can be used to help developers during different design phases of mobile collaborative applications was presented in [23]. This framework abstracts three levels of contextual information that respectively may be associated with the conception, analysis and architectural design phases of mobile collaborative applications. Although the mobility and collaboration aspects are supported through modeling a set of contextual elements, this work does not consider the limitation of mobile resources.

Cloud offloading. Treating classic problems related to limited mobile resources was the subject of multiple research works. Solutions presented in [24, 25, 26, 27, 28] are based on offloading techniques of intense computing tasks to the cloud. In [29] a Mobile Cloud Middleware (MCM) is proposed in order to perform dynamic resource allocation mechanisms for asynchronously migrating mobile tasks to heterogeneous cloud platforms. However, offloading techniques are based on monitoring process of the mobile resources use. Background monitoring processes and cloud workload can be as costly for mobile devices in time and energy consumptions [30] . Moreover, these approaches do not support the cloud mobile data sharing, since they require constant mobile/cloud connection. In contrast, our proposed cloning middleware enables static offloading of computing tasks.

Real time collaboration. Several research works have proposed collaboration systems specifically designed for editing shared documents in cloud environments. System SPORC [2] is a collaborative system that offers several users to edit shared documents.

To maintain their consistency, SPORC relies on a technique based on Operational Transformation (OT) approach and the use of a single server to give global order to concurrent user updates. As the synchronization logic is based on one server, this leads to bottleneck and a single point of failure. Indeed, a large number of messages are exchanged between users and the server. Based on this analysis, we can conclude that SPORC is not well-suited for mobile devices constrained by their limit battery life. System CloneDoc [31] enables (like SPORC) a centralized collaboration for mobile devices that are cloned in the cloud in order to alleviate the burden of collaborative editing works on mobile devices. Unfortunately, a server failure could stop the collaboration between mobile devices. Moreover, as mobile device and its clone are two entities physically separated, a delay when executing the same operations on both sides is possible. This may cause inconsistencies of the shared documents. Clonedoc addresses this problem through additional processing of OT on the mobile side. But this will cause supplementary energy consumption.

7 Conclusion

Modeling data sharing systems through mobile applications is considered as a challenge, since it must take into consideration the resource deficiency. However, recent research works on design models for mobile applications are not intended for data sharing in the cloud. Furthermore, other solutions are supported by the cloud environments, but without providing a reusable design. In this paper, we presented design patterns for mobile data sharing in the cloud under resource limitation constraints. The designed system makes abstraction of two main levels. The first one is the cloning middleware that allows for preparing a platform of mobile clones. This protocol offers web services, where end users can create and control clones of their mobile devices and manage their collaborative groups. The second level is the collaboration protocol that provides decentralized mechanisms for mobile data synchronization in the cloud. We also presented how we can easily reuse these design patterns for implementing two mobile frameworks: MIDBOX and OPTICLOUD. As future work, we plan to provide a reusable design for security within mobile data sharing applications in the cloud.

References

[1] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[2] Ariel J Feldman, William P Zeller, Michael J Freedman, and Edward W Felten. Sporc: Group collaboration using untrusted cloud resources. In *OSDI*, volume 10, pages 337–350, 2010.

[3] Sokol Kosta, Vasile Claudiu Perta, Julinda Stefa, Pan Hui, and Alessandro Mei. Clone2clone (c2c): Peer-to-peer networking of smartphones on the cloud. In *5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud13)*, 2013.

[4] Nadir Guetmi, Moulay Driss Mechaoui, Abdessamad Imine, and Ladjel Bellatreche. Mobile collaboration: a collaborative editing service in the cloud. In *Proceedings*

of the 30th Annual ACM Symposium on Applied Computing, pages 509–512. ACM, 2015.

[5] Moulay Driss Mechaoui, Nadir Guetmi, and Abdessamad Imine. MICA: Lightweight and mobile collaboration across a collaborative editing service in the cloud. *Peer-To-Peer Networking and Applications*, 2016.

[6] Clarence Ellis and Simon J Gibbs. Concurrency control in groupware systems. In *Acm Sigmod Record*, volume 18, pages 399–407. ACM, 1989.

[7] Abdessamad Imine. Coordination model for real-time collaborative editors. In *Coordination Models and Languages*, pages 225–246. Springer, 2009.

[8] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.

[9] Valeria Herskovic, Sergio F Ochoa, José A Pino, and H Andrés Neyem. The iceberg effect: Behind the user interface of mobile collaborative systems. *J. UCS*, 17(2):183–201, 2011.

[10] Valeria Herskovic, Sergio F Ochoa, José Pino, et al. Modeling groupware for mobile collaborative work. In *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on*, pages 384–389. IEEE, 2009.

[11] SOAP W3C standard. `http://www.w3.org/TR/soap/`.

[12] Web Services Description Language (WSDL) recommendation. `https://www.w3.org/TR/wsdl20-primer/`.

[13] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014.

[14] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. Mapping design patterns to cloud patterns to support application portability: a preliminary study. In *Proceedings of the 12th ACM International Conference on Computing Frontiers, CF'15, Ischia, Italy, May 18-21, 2015*, pages 50:1–50:8, 2015.

[15] AWS cloud design patterns. `http://en.clouddesignpattern.org`.

[16] Windows Azure application patterns. `http://blogs.msdn.com/b/jmeier/archive/2010/09/11/windows-azure-application-patterns.aspx`.

[17] Cloud computing patterns. `http://cloudcomputingpatterns.org`.

[18] Oleksiy Mazhelis, Jouni Markkula, and Markus Jakobsson. Specifying patterns for mobile application domain using general architectural components. pages 157–172, 2005.

[19] Nenad Medvidovic and George Edwards. Software architecture and mobility: A roadmap. *Journal of Systems and Software*, 83(6):885–898, 2010.

[20] Prasun Dewan. Architectures for collaborative applications. *Computer Supported Co-operative Work*, 7:169–193, 1999.

[21] Juan Rodríguez-Covili, Sergio F Ochoa, José A Pino, Valeria Herskovic, Jesus Favela, David Mejía, and Alberto L Morán. Towards a reference architecture for the design of mobile shared workspaces. *Future Generation Computer Systems*, 27(1):109–118, 2011.

[22] Andrés Neyem, Sergio F Ochoa, José A Pino, and Rubén Darío Franco. A reusable structural design for mobile collaborative applications. *Journal of systems and software*, 85(3):511–524, 2012.

[23] Rosa Alarcon, Luis A Guerrero, Sergio F Ochoa, and José A Pino. Analysis and design of mobile collaborative applications using contextual elements. *Computing and Informatics*, 25(6):469–496, 2012.

[24] Muhammad Shiraz and Abdullah Gani. A lightweight active service migration framework for computational offloading in mobile cloud computing. *The Journal of Supercomputing*, 68(2):978–995, 2014.

[25] Sehoon Park, Qichen Chen, Hyuck Han, and Heon Y Yeom. Design and evaluation of mobile offloading system for web-centric devices. *Journal of Network and Computer Applications*, 40:105–115, 2014.

[26] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[27] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[28] Feng Xia, Fangwei Ding, Jie Li, Xiangjie Kong, Laurence T Yang, and Jianhua Ma. Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16(1):95–111, 2014.

[29] Huber Flores and Satish Narayana Srirama. Mobile cloud middleware. *Journal of Systems and Software*, 92:82–94, 2014.

[30] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[31] Sokol Kosta, Vasile Perta, Julinda Stefa, Pan Hui, and Alessandro Mei. Clonedoc: exploiting the cloud to leverage secure group collaboration mechanisms for smartphones. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 19–20. IEEE, 2013.