POLITECNICO DI MILANO
DEPARTMENT OF ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

LIQDROID: A MIDDLEWARE FOR DIRECT INTERACTION BETWEEN MULTIPLE PROXIMAL ANDROID DEVICES

Doctoral Dissertation of:
Anita Imani

Supervisor:
Prof. Luciano Baresi

Tutor:
Prof. Carlo Ghezzi

The Chair of the Doctoral Program:
Prof. Andrea Bonarini

2017 – XXX

This work has been partially funded by Telecom Italia S.p.A., Services Innovation Department, Joint Open Lab S-Cube, Milano.

Acknowledgement

Studying Ph.D. while having diabetes was one of the biggest challenges that I faced in my life. As the nature of the Ph.D. will bring you high levels of stress because you don't know what will happen next in your path. This sometimes made me be worried about my health and put me in the situation to make a decision but what always brought me back to the path to continue my Ph.D. was my interest in learning new things and bring hope to the life of those people that have this disease. I would say besides all the difficulties that it can put in our way, but, we are more powerful than it. I want to say thank you to all those people that make me feel stronger than always to be able to follow this path. First of all, I would like to express my special appreciation and thanks to my advisor, professor Luciano Baresi for his continuous support, patience and precious advices. I would like to thank you for giving me the chance to be your Ph.D. student, encouraging my research and for allowing me to grow as a research scientist, your immense knowledge was always the best motivation for me to learn more.

I would also appreciate the financial and technological support of the Telecom Italia for my Ph.D. And also all the people who are/were working in the Joint Open Lab (S-CUBE) more specifically Cristina Frà and Massimo Valla. I can say the collaboration with the university and industry while is one of the really exciting and helpful experiences but can also be very difficult, and Cristina made it easier for me by her professional behaviour and knowledge, and I really appreciate it.

My sincere thanks also goes to Prof. Gerardo Canfora and Prof. Ivano Malavolta for accepting to review my manuscript and giving their valuable comments.

And the last but not least my beloved family and friends that always are there for me and besides all their worries let me follow my dream and have never stopped supporting me.

To my lovely father that always remains alive in my heart and my beloved family

Abstract

NOWADAYS, the speed of technology improvements in computing devices is very fast. The functionality and ease that using these different devices bring to us has influenced the way we are living and enabled us to benefit from them carrying out various daily tasks. But despite the improvements that have occurred in the field of mobile technology and the connection protocols, the current situation as regards multiple-device interaction techniques is still far behind what it could be, and these computing devices are still mostly working in isolation. Current multi-device interaction solutions enable the user to continue a task on another device, but their dependency on a specific set of devices and software artifacts (applications) on these devices limits their usage and forces the user to act as a bridge between the devices. This entails the user having to perform some preliminary and time-consuming steps to configure the next device on which to resume the task. While benefiting from the **direct** interaction between the multiple proximal devices, the user can interchangeably benefit from them in the execution of the different parts of the task he desires to carry out, exploiting the maximum potential that exists in these devices, to achieve better results and more user satisfaction.

This dissertation covers the motivation, design, and development of a novel paradigm to support multiple-device direct interaction through distributing the Android execution. The proposed solution to support this novel paradigm is a middleware infrastructure that will manage the **direct** interaction between a dynamic set of proximal devices to let the user seamlessly distribute the execution of a task between them, while also enhancing the

process of the interaction and integration of these devices. The proposed middleware, which is called **LIQDROID**, will distribute the Android operating system between a set of proximal Android devices to create a bigger **Android ecosystem**. The proposed ecosystem transforms the current pattern of single-user single-device to a fully cooperative environment that will empower a user to start a task on his device and be able to interchangeably and collaboratively benefit from the potential that exists on the proximal devices or other users during its execution and reach his final goal with better results. More technically speaking, LIQDROID is an Android service which benefits from the features available within the Android framework to solve the challenges that already exist in multiple-device interaction to manage the execution of a distributed task, such as finding the best capable proximal Android device to perform a task, and synchronizing the state of the integrated devices and the data management. This will provide the required framework for developers to easily be able to distribute the execution of a task on proximal devices and be relieved of the underlying complexities, and instead put their focus on designing and developing more innovative distributed applications.

A stable prototype of LIQDROID has already been implemented in Java, which is the main language of developing Android applications, and is ready to use. Also, a set of use case scenarios has been designed and considered in this thesis, based on the needs of real case scenarios. The developed versions of these use case scenarios in the shape of LIQDROID-compatible applications, along with LIQDROID, was tested on real devices to better evaluate and explore their strengths and weaknesses. The results were satisfying and also helped to apply the required changes in LIQDROID to improve it. These experiments have shown that LIQDROID introduces an innovative way of interacting between multiple proximal Android devices, which has the potential to introduce new and even more comprehensive features in the way of direct interaction between multiple proximal devices.

Contents

1 Introduction **1**
1.1 Distributed Android Execution 1
1.2 Research Problems 7
1.2.1 Homogeneous Platform vs Cross Platform 8
1.2.2 Supporting New Devices and Technologies (transparency) 9
1.2.3 Managing the Execution of the Distributed Tasks (adaptability) 10
1.2.4 Synergistic Specificity of the Applications' Parts . . . 11
1.3 Research Questions 12
1.4 Research Objectives 14
1.5 Major Contributions 15
1.6 Thesis Structure 16

2 Related Work **19**
2.1 Models and Technologies in Distributing User Interfaces . . 21
2.1.1 Supporting Multiple Devices 22
2.1.2 Supporting Multiple Platforms 25
2.1.3 Supporting Multiple Users 25
2.1.4 Supporting Multiple Contexts of Use 26
2.1.5 Supporting Multiple Modalities (input / output) . . . 28
2.1.6 Discussion and Comparison 29
2.2 Models and Technologies in Distributing Services 31
2.2.1 Multi-channel Services 32
2.2.2 Crossmedia Services 32

2.2.3 Discussion and Comparison 36
2.3 Middleware Technologies 37
2.3.1 Conductor 37
2.3.2 Panelrama 40
2.3.3 Multi-Device Interaction with Dynamically Migrating Engines 41
2.3.4 AllJoyn 43
2.3.5 Sip2Share 46
2.3.6 Remote Service Call 47
2.3.7 Middlewares on Android Binder extension 48
2.3.8 Google Play Services 50
2.4 Conclusion 50

3 Background About Android Framework **53**
3.1 Android Application Components 53
3.1.1 Activity 53
3.1.2 Service 56
3.1.3 Content Provider 58
3.1.4 Broadcast Receivers 58
3.2 Android Inter Process Communication 58
3.2.1 Intent 58
3.2.2 Intent Filter 59
3.3 Conclusion 59

4 Proposed Middleware Architecture **61**
4.1 Connection Layer 63
4.1.1 Advertisement and Discovery 63
4.1.2 Group Formation 65
4.1.3 Communication Channel 66
4.1.4 Device Abstraction 67
4.2 Interaction Layer 68
4.2.1 Intent Manager 68
4.2.2 Task Execution Manager 69
4.2.3 Artifact Manager 73
4.2.4 Event Manager 76
4.2.5 Service Manager 78
4.2.6 Communication Manager 79
4.2.7 Settings 80
4.3 Conclusion 81

5 Implementation Details and Technical Descriptions **83**
5.1 Advertisement and Discovery 84
5.2 Group Formation 86
5.3 Communication Channel 89
5.4 Device abstraction 91
5.4.1 Device Level Discovery 91
5.4.2 Application (Components) Level Discovery 92
5.5 Intent Manager 92
5.5.1 Task Execution Manager 95
5.5.2 Categories of devices' interactions 97
5.5.3 Managing the devices during the interaction (State Synchronization) 99
5.6 Artifact Manager 104
5.6.1 Transferring data through the cloud infrastructure . . 105
5.6.2 Transferring data Directly through the local Wi-Fi network 108
5.7 Event Manager 108
5.7.1 Device Level Events 108
5.7.2 Interaction Level Events 109
5.8 Service Manager 117
5.9 Communication Manager 120
5.10 Settings 120
5.11 Conclusion 121

6 Evaluation **123**
6.1 Usage of LIQDROID with Available Android Applications . 126
6.1.1 Sample Scenario 126
6.1.2 Application's Architecture and Complexity 126
6.2 Use case scenario 1: Joint Meeting Application 127
6.2.1 Sample Scenario 128
6.2.2 Application's Architecture and Complexity (Presenter Version) 129
6.2.3 Application's Architecture and Complexity (Participant Version) 131
6.2.4 Application's Architecture and Complexity 134
6.3 Use Case Scenario 2: Cameo Application 136
6.3.1 Sample Scenario 136
6.3.2 Application's Architecture and Complexity 137
6.4 Use Case Scenario 3:Take and Edit Image Application . . . 138
6.4.1 Sample Scenario 139

6.4.2 Application's Architecture and Complexity 140
6.5 Use Case Scenario 4: Home Video Player and Controller . . 141
6.5.1 Application's Architecture and Complexity 143
6.6 Use Case Scenario 5: Music Player Service 144
6.6.1 Sample Scenario 144
6.6.2 Application's Architecture and Complexity 145
6.7 Use Case Scenario 6: Inside Shoe Store Application 145
6.7.1 Sample Scenario 146
6.7.2 Application's Architecture and Complexity (Insert Products) 146
6.7.3 Application's Architecture and Complexity (Search Products) 147
6.8 Users' perspective about LIQDROID 150
6.8.1 Discussion of the Results: 153
6.9 Developers' perspective about LIQDROID 155
6.9.1 Discussion of the Results: 156
6.10 LIQDROID's Performance at runtime 157
6.11 LIQDROID's Overhead of energy consumption at runtime . 160
6.12 LIQDROID's Scalability 163
6.13 Other Possible Domains of Use and Comparisons 164
6.13.1 Multiplayer Games 164
6.13.2 Video Streaming Application 165
6.13.3 Distributed PDF Reader 167
6.13.4 City Guide 167
6.13.5 Museum Guide 168
6.14 Conclusion 169

7 Conclusion and Future Directions **173**
7.1 Answers to Research Questions 173
7.1.1 Research Question One [Q.1] 173
7.1.2 Research Question Two [Q.2] 175
7.1.3 Research Question Three [Q.3] 176
7.1.4 Research Question Four [Q.4] 177
7.2 Future Directions 178

Bibliography **181**

List of Figures

1.1 Percentages of consumers using different devices - the mobile devices have the higher usage.[1] 2
1.2 Multitasking view - Other devices are used simultaneously with the smartphone. [42][p. 2] 3
1.3 Number of mobile application downloads worldwide in 2016, 2017 and 2021 (in billions). [2] 4
1.4 The majority of professional developers choose Android over other mobile platforms.[3] 6
1.5 An example of distributing the execution of a movie inside an Android ecosystem which including different devices. . . 7

2.1 A sample of serial multi-device workflow of an interior design profession [64]. 27
2.2 Their proposed framework architecture. 42
2.3 Architecture of AllJoyn bus.[4] 45
2.4 High level operations [7] 46
2.5 Possible interactions [7] 46

3.1 A simplified illustration of the activity lifecycle. [5] 55
3.2 A sample view of how the activity task stack works in a device while the components belong to the same package or different. [6] 56

List of Figures

3.3 The service lifecycle. The diagram on the left shows the lifecycle when the service is created with startService() and the diagram on the right shows the lifecycle when the service is created with bindService(). [7] 57

4.1 A General Overview of LIQDROID Proposed Features . . . 63

5.1 Architecture of LIQDROID (Modules' dependencies) 84
5.2 LIQDROID provided alert dialogue for group formation upon receiving connection acceptance. 87
5.3 Advertising and Discovering proximal devices through Google Nearby Messages [8] 88
5.4 Finding the desired devices through available groups' members or list of all the available proximal devices. 89
5.5 Transferring the execution request(intent and its required data) through the unitMessage. 91
5.6 Sequence diagram of distributing the task's execution through the Intent Manager Module 94
5.7 Showing the Chooser list (of capable components) and launching the desired component(s) on the destination device(s) . . 101
5.8 LIQDROID will share artifacts between different devices through Firebase Storage. [9] 106
5.9 Sequence Diagram of launching a component on the destination device while the source device's role defined as Controller. 113
5.10 Sequence Diagram describing when the activity on the source device goes to the pause state while the device has defined its role as Controller. 115
5.11 Sequence Diagram describing an example of when an activity goes to the pause state on the Client device. 116
5.12 Sequence Diagram describing when an activity has launched by LIQDROID is resumed. 117
5.13 Binding an activity belongs to the Device 1 to a Service that belongs to the Device 2 (an example). 119

6.1 Discovery and group formation using LIQDROID 125
6.2 (a)Set of available features on the Participant component during an ongoing meeting, (b) The possibility to share various files during the meeting (c)Possibility to select and share the materials with a sub set of a group's members 134

6.3 (a) Possibility to select the proper device and the proper component on it, at the same time (b) Capturing an image through the desired component on the destination device, (c) Reviewing the image on the source device. 139
6.4 (a)Captured image by the device that has better Camera, (b) List of the available applications for editing an image, (c) The availability of the captured image on the third destination which proposes a better application for editing it. 140
6.5 (a)Sharing a video with your friend (b) LIQDROID will show the list of the available video players on your fiend's device upon it uploaded the file successfully to the cloud storage (c)The desired video player will be launched on the destination device while playing the video that has been downloaded successfully by LIQDROID. 142
6.6 (a)Through the Music Player activity the user chooses to run the service using on the other proximal devices, (b) The user selects the Music Player Service on one of the connected devices, (c) The user on the other device receives a permission request to let the other user runs the Music Player Service of his device. 145
6.7 Inserting unstructured data through LIQDROID 147
6.8 (a)The user selects to launch the activity to add products on the Sellers' devices. (c)(d)(e) The selected sellers will add the product info in the launched activity through their devices. (b) The manger later is able to apply query and view all the inserted results through LIQDROID in the database on his device. 148
6.9 Query unstructured data through LIQDROID 149
6.10 Multi-player Games Sample - Crossword Game. [20] 165
6.11 Shared YouTube browser [80] 166
6.12 City Guide Sample - (a) is the guide version with all the features visible and (b) is the tourist version with the hidden features. [20] 168

List of Tables

2.1 Comparison between LIQDROID and the available technologies in the field of multi-device interaction 20

5.1 Provided features by LIQDROID for managing the seamless interactions between the devices through the Task Execution Manager 96
5.2 Supporting the occurrences of different events in the devices and also during the seamless interactions between the devices through the Event Manager Module of LIQDROID 110

6.1 Types and the number of Android devices that each participant has. 151
6.2 Reasons proposed for using multiple devices by the participants. 152
6.3 Levels of the participants' engagement with the proposed use case scenarios 153
6.4 The participants' perspective about the usability of LIQDROID 154
6.5 Performance of LIQDROID at run time (with sharing data) . 159
6.6 Performance of LIQDROID at run time (general usage) . . . 160

CHAPTER 1

Introduction

> "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*"
> Mark Weiser

In this chapter, we briefly explain the concept of multi-device direct interactions and the scope of the work.

1.1 Distributed Android Execution

Device cooperation is one of the very important subjects these days, as users mostly benefit from different computing devices to perform their daily tasks and duties. Due to the fast trend of improvements in mobile technologies, we are in a transition phase from the desktop and personal computers to the world where mobile computing is touching every aspect of our daily lives and is accessible any time and everywhere (Figure 1.1.) Nowadays mobile technology enabled devices are available in different forms, sizes, capabilities, and functionalities, as smartphones, tablets, wearable devices and car dashboards. The heterogeneity of these devices, their availability, and the differences that exists between them related to their software

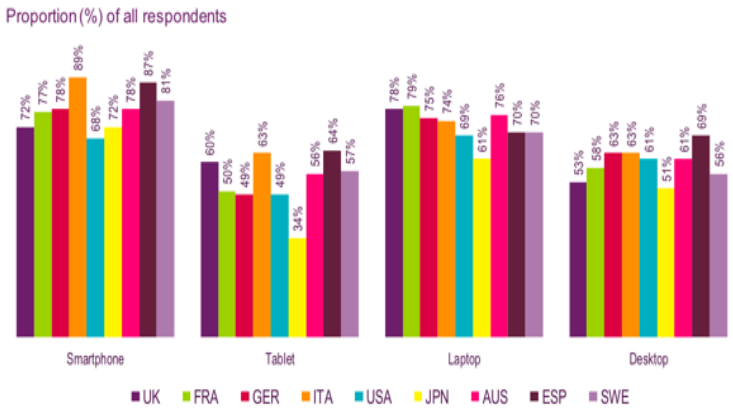


Figure 1.1: *Percentages of consumers using different devices - the mobile devices have the higher usage.*[1]

and hardware features, such as power consumption (battery life), storage, memory, and portability, as well as the users' preferences, means that the user prefers to possess several of them and use them interchangeably every day [36, 64]. Most of these devices, although they are in close proximity to each other, are still working as standalones, and their capabilities and potential are not fully exploited. On the one hand, it is not possible to put all the features that this diversity of mobile devices offers in a single device from the hardware and software complexity point of view, and also the cost; and more importantly, users prefer to have different devices to pursue their purposes instead of having a single all-in-one device [10]. In this study, Dearman et al. have studied why and how people use multiple devices, and they derive these primary findings: First, users prefer to separate their work and personal activities. Second, users assign different roles to devices as each user may use these devices in various forms and conditions. Third, spanning the activities on several devices is more comfortable for the user than handling all of them in a single device. For example, a user may start searching a video on his cellphone, then use the TV to watch it and his tablet to read the reviews and write his own review. Currently eighty-six percent of users involve other technologies to listen to music, read news, watch movies or use the internet, engaging their smartphones (Figure 1.2).

[1] `https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/`

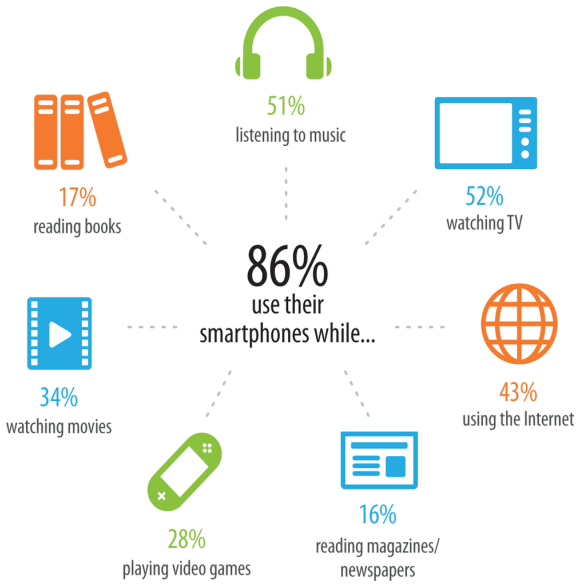


Figure 1.2: *Multitasking view - Other devices are used simultaneously with the smartphone. [42][p. 2]*

The existence of several devices in a user's proximity and moving between them to perform different tasks makes the user feel more and more the need to have an infrastructure that enables seamless interaction between these devices in order to benefit from another device during the execution of the task at hand. Although different platforms and solutions have been developed to enhance such cooperation, by decreasing the gap between these devices and bringing them together as a system, these devices are mostly working in a standalone scenario. So the user needs to move between them and perform some preliminary steps for the communication (connecting devices together through protocols, wire or third devices), configuration or synchronization of the devices (the new device reaching the same state) when the user needs to use a different set of devices at the same time, such as during formal meetings, a gathering of family/friends, availability of several proximal personal devices etc. Without interaction between devices, a lot of time and resources are wasted, whereas by having a good infrastructure that handles cooperation between the devices, we could assign the time and better manage resources to improve the work and increase productivity.

On the other hand, based on the context that the user belongs to and the task requirements, the user's access may be restricted to a specific device to perform the task. The word "task" here refers to the predefined sequence of

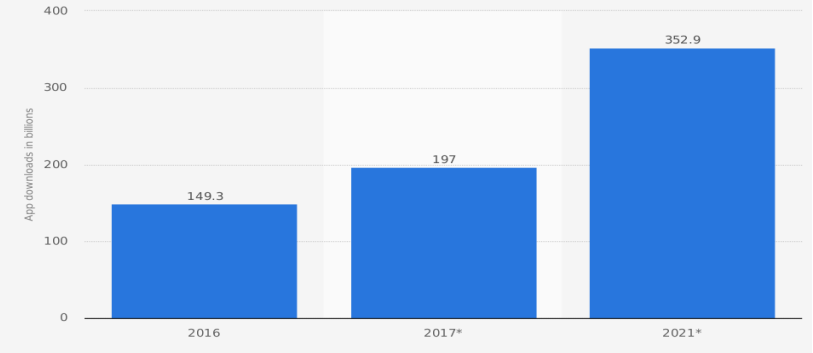


Figure 1.3: *Number of mobile application downloads worldwide in 2016, 2017 and 2021 (in billions).* [2]

steps that the user needs to follow to achieve the final situation (or result) which is his desired goal. To perform different steps of a task the user benefits from the software artifacts which are installed on these devices and commonly called mobile applications. One of the main reasons that helped mobile technologies become pioneers are these mobile applications which are small, loosely coupled software components which provide limited and specific functionality. Nowadays mobile applications have entered into different aspects of the user's life, ranging from entertainment, shopping, education, and business, up to new and interesting fields such as virtual and augmented Reality. Figure 1.3 shows a forecast of the increasing trend of mobile applications downloads worldwide in 2016, 2017 and 2021. This has caused a huge increment in the time that people spend on interacting with their devices and will also change the way that people will interact with them in the near future.

Currently, because of the limited hardware resources available on mobile devices, multitasking has been avoided. The constraints that these mobile devices have, such as the power consumption and the nature of their portability may mean that they won't always be accessible for the user, who cannot have the freedom to choose on which devices to perform his desired task, based on his preferences. These device limitations also will have an effect on the execution of the applications that are installed on them. If there exists some limitation in the hardware available on the device, this can

[2] `https://www.statista.com/statistics/271644/worldwide-free-and-paid-\ mobile-app-store-downloads/`

cause an installed application to become completely or partly useless, for example, if the storage on your device is full, you will not be able to use your camera application to capture any image. In addition, as the applications are still limited to the boundaries of a single device and are working in an isolated environment, they can only interact and benefit from the services that the applications installed on that device can provide during their execution. Various studies put their main focus on improving the usability models at the application level [28, 33, 53, 72], such as PACMAD usability model [28]. It introduces seven attributes which are important in measuring the usability of an application: effectiveness, efficiency, satisfaction, learnability, memorability, errors and cognitive load. They believe that beside the limitations that exists in the nature of these mobile devices, that we have also mentioned-above, that will limit us to the design of the applications, the context of use such as using the applications while the user is moving also have a very important effect on the usability of the applications. So considering the above-mentioned attributes in the design and test phase of the application development will improve the quality of its overall design, how the user will perceive and interact with it and finally brings higher levels of success and user satisfaction.

But despite all the improvements that are introduced to apply in the design phase and to the functionalities of these applications based on the proposed models, there is still a gap between the functionality that they offer and what can potentially be achieved. This gap is understandable, considering the constraints and limitations that mobile devices have put in the way of the applications' usage, forcing users to move between the diversity of devices to perform different tasks and pursue their goals. The limitations that the mobile nature of these devices poses to the user could be counted as dynamic change in the context, connectivity problems (slow and unreliable), and different data entry types (error-prone and slow rate of data entry in movement), which will cause a huge reduction in the usability of the applications, as well the satisfaction of the final user [28].

We believe that by providing an infrastructure that lets these devices become integrated and cooperative as a single unit, we can support these two views at the same time, which will enable different components of an application to flow between these multi-integrated devices, and have access to a broader pool of resources. This possibility of distributing applications' components will cause that the maximum functionalities and capabilities of these devices will be achieved, some of the current limitations will be removed and will result in higher levels of application usability, productivity, and user satisfaction that currently exist.

The current potential that exists in the Android framework, as an open source framework with a wide range of different available Android devices and the high attention that it achieved from application developers (Figure 1.4) persuaded us to think about Android as a great opportunity to benefit from it in the design and implementation of our solution and being capable of reaching the novel interaction paradigm in multi-device interaction that is needed.

The proposed paradigm will create an Android ecosystem among all the available proximal Android devices. This bigger Android ecosystem will let the user start a task on his device, based on the availability of the devices in his proximity and the constraints that the context of use may impose to him be able to combine different sets of proximal Android devices dynamically and distribute the execution of the task between them as the figure 1.5 shows. As a sample scenario to propose better this Android ecosystem, considering that the user selects a movie on his smartphone to watch, he starts watching the movie, when he is at home he is able to resume the video (distributes the execution of the video) on his tablet and TV. But if he is inside the car, he is able to resume this movie on the car's tablet or share with his friends' Android devices. Or while he is on the move he is able to watch it through his smartwatch. So these devices are not separate entities any more but they are all part of a general entity (which we call it as Android ecosystem) which allows the user's task to easily distribute

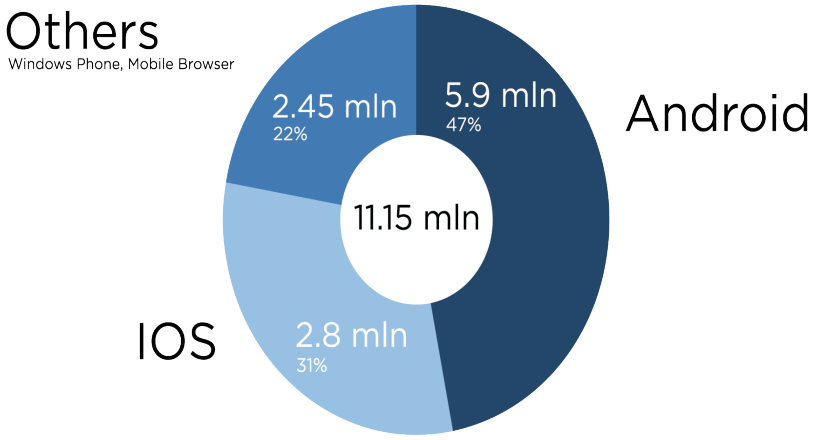


Figure 1.4: *The majority of professional developers choose Android over other mobile platforms.*[3]

[3] https://scand.com/company/blog/mobile-world-turning-toward-android-\ios-vs-android-development-statistics/ Or https://goo.gl/zJxHsS

(flow) among them. This proposed ecosystem will be a big step toward reaching a fluid computation vision [6] in the sense of having a system which lets information and tasks become available everywhere and anytime to support spontaneous human usage while is appearing invisible to the user. This solution will enable the multiple-devices to be more cooperative and goes further than just synchronizing the devices through accessing the last version of the data or just connecting the devices. But the components of applications become distributed between the devices. This new seamless interaction paradigm will have a great effect on the way that users conceive and interact with their environment, each other and other available devices.

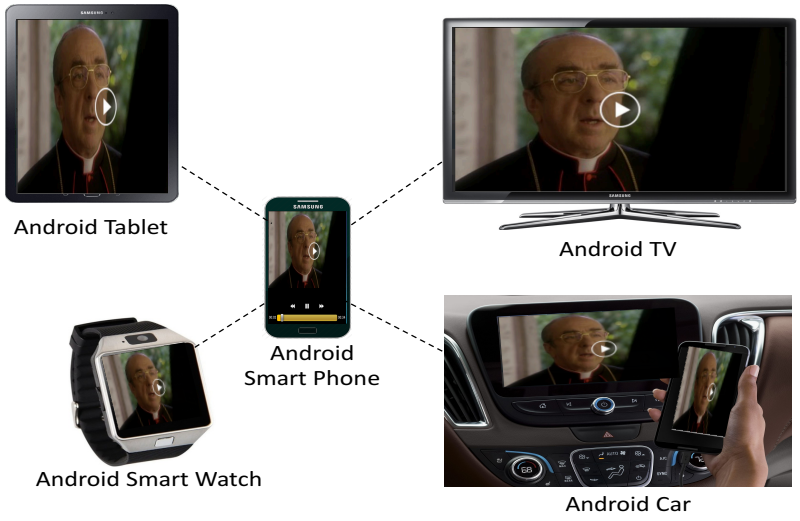


Figure 1.5: *An example of distributing the execution of a movie inside an Android ecosystem which including different devices.*

1.2 Research Problems

Although the current speed of improvements in the communication protocols and computational capabilities of the devices are very fast, still they are not able to help the user exceeding the boundaries of a single device and use these advancements more adequately. This will cause that the user finds himself with several single-purpose devices and the first consequence of this isolated usage on a single device is that the users won't be able to exploit the real capabilities of devices. Lack of a proper infrastructure that can adopt these technologies and make them cooperative based on the each task's requirements is one of the main reasons that causes this isolation problem

which consequently will put the user far behind in properly benefiting from these new technologies. In this section, we are going to describe this problem and the reasons that cause it in more detail.

The first obstacle is that most of the time the user is not aware of all the devices which are available in his proximity and the capabilities that they have, not just in the sense of hardware features, such as whether it is a smart phone or a watch which can, for example, include different sensors, but also the features such as the applications which are available on them that would be useful for the user's intended task, as well as a prototype that can help him to overcome the complexities that integrating these wide range of devices may cause. While a lot of mobile technology enabled devices are increasingly entering a user's daily life, our current situation as regards interaction techniques is still far behind this trend, which reveals a series of standalone and isolated devices. Much research has been done to advance the usability of multi-device interactions, though knowledge about the experiential side of their work as applied to real situations is still insufficient. The results of current experiments show that users are still struggling with the weak performance that these solutions provide regarding fundamental aspects such as controlling the interactions, the users and the context of use [32], and this may cause the user to lose his interest in benefiting from these solutions. Therefore, innovative paradigms should still be designed and developed that comprehensively consider all aspects of multi-device interactions which can be a different permutation of the dimensions, including the interactions between the user and other users, the user and devices, and between devices [18]. Below we mention some of these fundamental problems that exist in the current solutions, the reasons that cause them and the attempts that we need to make to eliminate them.

1.2.1 Homogeneous Platform vs Cross Platform

Different solutions such as web-based frameworks or middleware technologies such as Alljoyn [1], which we will talk about them in more details in the section 2, have been proposed to handle the interaction between devices, but they mostly focus on being cross-platform as the key feature in their solution. These approaches need to offer suitable generality in order to be adaptable by several platforms. Otherwise they will be limited to support the high-level interactions between the devices such as sharing the final results of the tasks, messages, and notifications. But focusing on a homogenous platform will enable us instead of considering each of the devices as a single entity while is imposing its own limitations and rules, can consider all the

devices as a single entity and the framework as the shared infrastructure that is distributed between all of them. This homogeneity between devices will help us to better analyze the requirements, formulate the problem, and propose more comprehensive and extensible solution. As this comprehensive solution will have wider access to the lower level details of the execution of the tasks is capable of proposing more features such as allowing the user to be able to fully customize the execution of a task based on his preferences and the task's requirements, improving the final results of the task by exploring and proposing better available resources for that task and result in a better outcome, while it also will be easily capable of supporting those high-level interactions mentioned above. For example, if you want to take a photo and send it to your mother and you know that the camera of your tablet is better than your phone, so you take the photo with the tablet and then as your tablet is not capable of sending messages, you send back the image to your phone and then send it to your mother through the message application on your phone. Or you are watching a movie on an Android TV, and when it has finished, you want to watch another related video that the YouTube application on your phone has suggested, so you start the video on your phone and then transfer it to the TV.

1.2.2 Supporting New Devices and Technologies (transparency)

Nowadays, the speed of improvements in mobile technologies is very fast: every day new developments in the mobile technology field happen which enhance the current capabilities of mobile applications and enable them to support a new area of the user's life, ranging from business and health care, to art, education, etc. This trend will add extra requirements and expectations which force the current solutions to be revised or replaced with new solutions that are better capable of supporting these newly added values and features.

Dealing with this aspect involves the fact that the proposed solutions should be flexible enough to accept lately joined devices. This does not mean how many devices from different platforms we are capable of handling but how we are able to integrate these devices to achieve the maximum capability that they can propose. This is completely opposed to the view of the cross-platform solution, as the nature of a task and applications currently in different platforms, mainly Android, Apple and Windows, has an entirely different development infrastructure. To tackle this problem, the solution should propose adequate levels of abstraction regarding the task's structure, its execution requirements and its dependencies on the platform. Focusing

on a single platform will decrease the level of complexity and increase the proposed features and potential of the solution to be more adoptable to the incoming devices and technologies in the future.

1.2.3 Managing the Execution of the Distributed Tasks (adaptability)

A key component in maintaining multi-device environments is how we are capable of controlling the interactions. Currently available solutions which let the user move between different devices seamlessly, mostly focus on centralizing the access to the underlying data used by the applications so be able to support the task distribution and synchronizing the state that the user had on different devices, based on letting all the applications have access to the last copy of the data. To achieve this approach, the data was kept in a centralized place, mostly in the cloud, and it was shared with all the applications installed on different devices. But the first problem with these approaches is that they are applicable only for some applications that have been developed by specific platform vendors or device vendors. The other very important problem is that they are personalized, as the user must enter with the same identity on different devices. So the distribution of the task is subject to the possibility of having access to these sources of data and sync the data and states between those devices. But our main focus is letting the user select devices which are in his proximity, even if it is his own device or a colleague's device, his wife's device or a shared screen in a museum, and to be able to communicate with that device(s) to benefit from its features on the execution of the task at hand to reach his desired (mainly better) results.

As Denis and Karsenty [12] pointed out, a framework's adaptability can have two different sides. On the one hand, it is the capability of the system to adapt itself to the environment that the user is in, as well the capabilities and features of the existing devices. On the other hand, it is the ability of the system to let the user adapt himself to the different use situations and apply his preferences in case of need. The ability of the system to overcome the adaptability challenge lies in how much the system is capable of recognizing the devices (resources) that exist in the user's proximity, and making the most of them by the integration of these multiple devices. As well as supporting the privacy concerns at the same time but instead of using the credential as mentioned-above, the system can manage it through two separated phases the first phase is related to the user which the device belongs to and needs to provide the required permission before the interaction starts. And the second phase is about the data that the application wants to use and needs to be handles between these devices properly.

1.2.4 Synergistic Specificity of the Applications' Parts

The other problem that can arise while using this seamless interaction between multiple devices is how different parts of applications can be distributed on the different devices and interact with each other to achieve the final user's goal. This problem arises from the necessity of having synergistic specificity in the system. The meaning of "synergistic specificity," as Schilling [66] defined well is "the degree to which a system achieves greater functionality by its components being specific to one another" in a specific configuration. Systems which are high in synergistic specificity are able to provide functionality and user experiences that isolated devices are not able to perform and propose to their users.

Currently, this integration proposed by different systems, are not fully aligned with the task requirements, what the devices are proposing and what the user expects. For example, consider that there is an architecture company where the user wants to take a good picture of the proposed building plan, edit it on his tablet and then show it to his boss on the big screen, and they want also to edit it together. The common trend currently to perform this workflow is that people use another application in between to send the data (here, the photo) to the other device using applications that have been specifically developed for sharing the data or using available solutions such as emailing or messaging applications.

Aside from the time that will be wasted to launch another application to share the data, it will put more complexities and degradation into the quality of the final result. Here just making the devices and the application installed on them be able to interact with each other is not enough. While we need a solution that lets these different devices and the required applications be able to become united and work together as a single unit. So based on this system the user can directly find and launch the other application on the tablet with the data it needs from his smartphone, edit it and receive the result back on his device, or send it directly to the big screen to review it with his boss. The proposed solution sounds better because it has enough knowledge about the task at hand, its different parts, and requirements. In addition, the system also has a good knowledge about the availability of the application's components on the proximal devices and is able to distribute the parts of the task between them, apply the changes and collect the results.

1.3 Research Questions

In order to fully understand the requirements and how the proposed solution can overcome the above-mentioned problems, the following research questions need to be answered:

Q.1 What are the currently available solutions (methods and technologies) capable of supporting direct interaction between multiple proximal devices? (Chapter 2)

One of the main obstacles that exists in the field of multi-device interaction is that you cannot find a playing field where all people can come along, find the existing works and improve them. Each researcher tries to provide his own solution without considering the other related works. We have reviewed the existing works to better understand the requirements, existing challenges, strengths and weaknesses of the works that have been done so far, especially those that function within the Android framework and align with what the users expect from these systems. These steps helped us to better formulate what the challenges are and what we need to do.

Q.2 What do we need to consider to enable multiple proximal devices to directly interact with each other? This required answering the following sub-questions: (Chapters 3 and 4)

- What are the requirements of the final users, and how do we manage them to allow the users to seamlessly distribute the execution of tasks?
- What are the requirements of the developers, and how do we manage them to allow the developers to seamlessly distribute the execution of the components of their applications?
- What would be the immediate outcomes of the proposed solution?

The proposed solution will help us to fill the gap that currently exists between the functionalities of the devices and what the resources available on them (hardware and software resources) currently offer and what could be achieved by integrating them in reality. This will at the same time address the limitations that exist in a single device while allowing us to exploit all the potential and resources that exist on the devices in our proximity, and greatly improve the usability of the applications which are installed on them. In this way, the user will need less time to benefit from another device(s) in the execution of a task at hand.

On the other hand, one of the main aspects to be considered for the users is that it should not only facilitate the work for them to achieve their desired goals but also should be exciting for them to use this solution. In

order to reach this goal, we first need to know what is currently available and accessible that we can benefit from in the architecture of our solution. We also need to determine what we need to propose to tackle the existing challenges and better fulfil user expectations. More specifically, the main focus goes to the Android framework as it will be the primary building block of our architecture, and the other proposed blocks to put the solution into practice will be in line with it.

Our solution aims to open a new perspective in the design of solutions for multi-device interactions. On one hand, we will try to provide a comprehensive infrastructure to facilitate the developers' work, as they will not be restricted anymore to the physical boundary and limitations that exist on a single device; they will also have access to a pool of resources and potential features to use when designing their applications. This will give them the required freedom to focus on developing more innovative and useful applications for their users.

Q.3 What are the technical aspects considered in the implementation of the middleware, and how have we addressed the possible barriers and inconsistencies that we were faced with in the process? (Chapter 5)

Working inside the Android framework, however, provides a good opportunity for us to focus more on the task distribution aspect and not on the execution of the task on the proximal devices. But in order to integrate these two steps, we need to be fully compatible and obey the Android framework rules. This question is related to how we have integrated and synced our middleware infrastructure with the Android framework to handle the distribution of the tasks and exploit the potential that already exists in the Android OS to provide the proposed paradigm.

Q.4 How will the proposed solution validate itself in the real case situation? This is more specifically answered by the following sub-questions: (Chapter 6)

- How can it improve the process of developing distributed applications in both aspects of novelty and required attempt?
- What is the opinion of final users about it while they interact with it?
- What is the opinion of developers about it in the sense of usability, and how much can it improve their attempts to develop distributed apps?
- What is the overhead of it in terms of performance and energy consumption at runtime?

To better understand how successful we were in achieving the desired results and how capable the proposed solution is of providing for the devel-

oper's and user's needs, several realistic use case scenarios should be defined and implemented to evaluate how the system will work on the real devices, how much it can help the developer, and how it can persuade the final user. To this end, we have conducted several experiments with different users and developers, have collected their opinion and idea during the experiments as well as we have measured the performance of LIQDROID at runtime during each experiment.

1.4 Research Objectives

The main research goal of this thesis can be defined as follows:

Distributing Android execution between a set of proximal Android devices to create a bigger Android ecosystem. This will enable the proximal Android devices be able to directly interact with each other. This ecosystem will enhance cooperation between the devices and will empower the user to start a task on an Android device and be able to seamlessly distribute its execution to the proximal Android device(s) which best suits its requirements.

Most of the current solutions focus on supporting the heterogeneity of the devices with different platforms and synchronizing their state through the cloud infrastructure by accessing the last version of the data. These solutions are not capable of providing an infrastructure for the user, based on the task requirements, to be able to interchangeably use different devices which have better capabilities for executing that task. As these solutions do not provide enough abstractions for users with regard to all the features and capabilities that are available in a device, along with the requirements of a task, and the user's needs we still face the situation that users most of the time are only interacting with a single device. Even if the user is aware of the availability of the other proximal devices, as he is not able to handle the interchangeable use of different devices for different parts of a task, sometimes he tries to find a solution by himself to overcome the communication obstacles between the devices, or he prefers to do the whole task on a single device because he is not able to make them cooperate with each other. In order for a task to seamlessly flow between different devices with as little user intervention as possible, we have eliminated the boundaries that already exist between proximal Android devices which force them to work in an isolated manner. This will enable the user to distribute the execution of a task between a set of proximal devices and reach his goal. While he has used several devices interchangeably, he still thinks that all the execution happened in the single

device in his hand.

Another important aspect in achieving higher user satisfaction is that although this multi-device cooperation can improve the task execution, it should not cause any problems in the normal execution of the other applications and the device (i.e. it should not decrease performance) or disturb the user while he is not interested in multi-device usage and is using his device normally. It should be invisible, and whenever the user needs it, it should do the job.

The proposed solution should be general enough to be applicable for supporting the needs of most domains of use, and easy to use for various users (expert or non expert). Moreover, as we mentioned, there exists a wide range of Android applications already on the market, so the proposed solution should be able to serve these applications or at least need minor changes to become compatible with the middleware and benefit from its proposed services and features.

1.5 Major Contributions

The major contributions of the thesis can be defined as follows:

- A comprehensive explanation of the available solutions provided by the researchers or the industries to integrate multiple devices and let the user distribute the execution of a task between them. Providing a comparison between these solutions and our proposed solution and finally, defining the actual requirements, possible issues, strong points and limitations that still exist in the field of multiple devices interaction.
- The proposed solution is a middleware infrastructure called "LIQDROID." [4] The LIQDROID has been developed inside the Android framework, which enables the set of Android devices in the user's proximity to become united and create a bigger Android ecosystem. LIQDROID will let a running task seamlessly flow between different devices in this ecosystem and benefit from their possible capabilities and features to enhance the task's execution and its final artifacts. LIQDROID will manage the discovery and communication between these devices and also take care of the data availability, which will be served as the input for the consecutive executions on the same device or other user selected devices to reach the final user-defined goal. This data can be the output of the previous execution (on the same or a different device) or user

[4] Because of the sponsorship, the source code of the stable prototype of LIQDROID can be only available upon direct request.

defined data from the device's storage, with the possibility to store the outcome of the executions in internal storage or in the cloud, to be accessible during the running of the task or in the user's new context.

– The proposed infrastructure will enable the user to discover the available resources and features (hardware and software) on the proximal Android devices, based on the running task's needs, and apply the user's preferences in distributing the execution of the task between the set of different capable proximal devices. The user will also be able to control the interaction between the devices at run time and apply his preferences.

– Proposed and implemented different concrete real-use case scenarios in different domains of use to test LIQDROID during its execution to demonstrate its strengths and remove the weaknesses. These LIQDROID-compatible applications better exploit the capabilities and features that LIQDROID offers to the end users by the emphasis on incorporating different contexts, interaction types, and different users, all of which impose different requirements and needs.

1.6 Thesis Structure

Chapter 2: In chapter two we will define the existing solutions and researches have done in the field of integration and interaction of multiple devices. Their current strengths and weaknesses, comparing them with the LIQDROID's desired approach and a short description of how we are going to solve them by LIQDROID. In this chapter, we have provided a general and classified overview of the aspects that play the key role in the challenges and complexities that exist in providing users' persuasive solutions.

Chapter 3: We will explain the main concepts about the Android framework that has used in the implementation phase of the LIQDROID, the technical words and the basic knowledge related to their usage which is required to make the chapter four and five more understandable.

Chapter 4: In this chapter, a high-level explanation about the architecture of LIQDROID has proposed. The abstraction and modules to support different requirements have defined follow by a general explanation about the necessity of having them and what will be the final feature proposed by the system to the final user.

Chapter 5: By benefiting from the technical word that explained in chapter three, and also the required modules explained in chapter four, we will give a more technical explanation about the implementation of each module. It includes the logic that the middleware uses to benefit from the available features in the Android framework to handle the distribution of the task execution. The challenges that we considered and those that we have encountered during the implementation and the solutions that we have provided to overcome them.

Chapter 6: In this section, we will discuss more the concrete use case scenarios that we have defined and implemented to evaluate different aspects of LIQDROID in the real situation and will highlight the features that it can propose to the developers and end users.

Chapter 7: In the end, we will provide an over view of the obstacles that exist in the way of distributing the execution of a task between the different set of proximal Android devices, and conclude it with the proposed solutions and the possible future directions and improvements.

CHAPTER 2

Related Work

In this Chapter, we will provide a general overview of the current solutions that exist in the field of multi-device interaction. As it is an extensive topic, and has attracted much attentions these days both from the academy and the industry, several different terms have been used by researchers to define the platforms and systems which are concerned with the interaction and cooperation between multiple devices, users, applications and platforms. In human computer interaction, the terms "distributed user interfaces" (DUI) [5, 16, 45, 48, 80], "multiple user interfaces" [12, 19, 54, 59, 71] or "multi-device systems" [23, 46, 70, 77] have been used interchangeably. In the following, we mention to this category generally by using the term: DUI. While the introduction of continuity of tasks in marketing and industry used the terms "multichanneling" and "crossmedia" [60, 69], which generally focus on cross-platform interactions. Besides the differences that exist in the naming schemes but their focus is on providing an infrastructure to facilitate the multi-device interactions.

Here we have gained the most benefit of the current solutions by prioritizing them based on their main scope of work, the challenges that they have concerned, which have highlighted the different design approaches to their strengths and limitations, as well as mentioning to some of the leading

solutions that are placed in those categories.

We have provided a general overview and comparison of these available technologies to LIQDROID in the table 2.1 while for more comprehensive explanation and comparisons you can refer to the relevant section which those that are more competitive to our solution are explained at the end of this chapter.

Available Technologies / Supported Features	LIQDROID (our proposed middleware)	Sip2share [7]	Alljoyn [1]	Samsung Flow [2]	Multi-Device Interaction with Dynamically Migrating Engines [21]	Remote Service Call [49]	Android Binder extension [36, 42, 54]	Google Play Services	Liquid Software [22, 48, 74] (Liquid.js for DOM [75], Liquid.js for Polymer [21])	Apple Continuity [34]	Conductor [27]	Panelrama [75]
Multiple Devices	✔	✔	✔	Only Samsung devices	✔	-	✔	✔	✔	Only Apple devices	✔	✔
Multiple Platforms	Only Android OS	Only Android OS	✔	Only Android OS	Only Android OS	Only Android OS	Only Android OS	Only Android OS	✔	Only Apple OS	✔	✔
Multiple Users	✔	✔	✔	✔	Static Credentials	✔	✔	Same Credentials	✔	Same Credentials	✔	✔
Multiple Contexts of Use	✔	✔	✔	✔	✔	-	-	✔	✔	-	-	✔
Multiple Modalities	✔	✔	✔	✔	✔	-	✔	✔	✔	✔	✔	✔
Data Synchronization	✔	-	✔	✔	-	✔ (only the data available in its apk)	-	✔	✔	✔	✔	-
State Synchronization	✔	-	✔	✔	✔	✔	-	✔	✔	✔	✔	✔
Task distribution (Task continuity)	✔	✔	✔	✔	-	✔ (only activities)	✔	✔	✔	✔		✔
Supported apps	Existing apps & compatible apps	Compatible apps	Compatible apps	Compatible apps	Compatible apps	Existing apps	Compatible apps	System apps	Web based apps	System apps	Web based apps	Web based apps
Match the task with the device's capabilities	✔	✔	-	-	✔ (only UI adaptation)	✔ (only UI adaptation)	-	-	✔ (only UI adaptation)	-	-	-
Integrating devices to perform a task	✔	✔	-	-	✔	-	✔	-	-	-	✔	✔
Managing devices' Integration challenges	✔	-	-	-	-	-	-	-	-	-	-	-
Managing Artifacts	✔	-	-	✔	-	-	-	✔	-	✔	-	-

Table 2.1: *Comparison between LIQDROID and the available technologies in the field of multi-device interaction*

Reviewing these approaches have enabled us to better understand the challenges that already exist in this field, understanding the different design

approaches and the challenges associated with them to better form the proposed architecture for LIQDROID. Then we will also provide more details regarding the available middleware technologies that are prominent in supporting multi-device interactions and the differences that LIQDROID will have compared to them.

2.1 Models and Technologies in Distributing User Interfaces

With the improvements in ubiquitous computing and the increasing trend of availability of computing devices, the user finds himself more and more in a situation where he needs to move between different devices to carry out his interactive tasks. In the field of HCI, distributed user interfaces received a lot of attention from researchers with regard to the benefits from the availability of multiple devices to provide innovative interaction solutions.

This trend is a move away from traditional centralized desktop computing environments, where all the tasks were performed by a single device and all the data were stored in the same device or another centralized place (a certain device), to environments where the tasks and data can flow between several available devices at any time. This new generation of interactive environments makes it possible for different users to work simultaneously and have access to several devices alongside each other. Various approaches have been devised to tackle existing problems by enhancing the integration of multiple devices, improving usability, decrease user's effort and increasing user satisfaction [16].

Based on the definition proposed for DUI, it is the concept of distributing different elements of an interface between a different set of devices, different platforms and a different set of users. In general, the following dimensions can be considered: input, output, platform, the context of use, time and user(s) [11, 16, 67]. Considering different combinations of these dimensions causes the problem space to become larger by an order of magnitude. Consequently, the effort required to interpret interactions and provide a novel paradigm may increase very rapidly. The vastness and diversity of existing methods in supporting multi-device interactions in the field of DUI made it hard for us to be able to directly compare our work with them. Meanwhile we have provided a proper categorization of these available solutions based on their important factors and common features that they include to support particular difficulties that exist in this field.

At the end of this section, based on what we have discussed through those categorization we will give a brief discussion of the current limitations in the field of DUI and the issues that were weighted more for us to consider

them in the LIQDROID, to make it capable of exploiting higher levels of potential that exist in these multi-device and application interactions and achieve further user satisfaction. These categories are as follows.

2.1.1 Supporting Multiple Devices

Different types of devices with different resources may be available in the user's proximity. Availability of devices is the main building block in the definition of DUI, which has an important effect on proposing the usage patterns between the devices. There can be different reasons why the user may prefer to hold several devices [58]; here we mention the most persuasive one. First, a single device may not have the capabilities, functions, and data needed for a task, aside from the fact that sometimes a device is not capable of performing two different roles simultaneously. For example, you are not able to write an email while you are watching a full screen video on your smart phone, so either you choose to do them one at a time and momentarily ignore the other one, or you are forced to use another device for watching the video simultaneously. Second, some users prefer to separate their devices based on their own interests such as devices for work and for personal use, to have more privacy and freedom of use. Several experiments have been conducted to perceive the different paradigms that exist in the way that users integrate multiple devices and interact with them. A variety of factors are considered in these experiments, such as which device is the best match for doing the current task at hand [25], considering the hardware features available [2, 62]. By interpreting the outcomes of these experiments, models and also guidelines for better understanding and designing new paradigms which are proper for multiple device usage scenarios were provided, along with introducing available gaps, open questions and limitations that still exist in this field [38, 58, 64]. So far, one of the main challenges from the user's point of view in this area is managing the information [31] that is the consequence of handling the interaction between multiple devices. We will discuss this aspect more in the section 2.1.5. From the results achieved, three main interaction patterns may happen while using multiple devices, which are as follows:

- Sequential Usage Patterns: where a task is transferred from one device to another that has more suitable features for the task and the context of use.

- Parallel Usage Patterns: where several devices (or screens) simultaneously are performing the same task and their state is synchronized.

- Complementary Usage Patterns: where several devices cooperate with each other to perform a single task. Each consequence step of the task will be handled by a device which has the required capability. For example user takes a photo by his smart phone and edits it by an application which is installed on his tablet.

Based on the patterns mentioned above, along with the requirements they impose, different capabilities and features (hardware and software) of the available devices in dynamic environments, different models have been proposed to help developers as well as users to overcome the complexities better and handle interactions between multiple devices. In the modeling phase, the considerations do not only concern the task at hand that will be transferred to the other device, but also the current state, predefined parameters, and underlying content to continue the execution.

One of the main reference models among these proposed models is called 4C dimensions [11], which provides a broad, comprehensive view of essential aspects that will help developers to better understand the features which are important in their target use case scenarios and design their frameworks by counting on them. The model mentioned is expressed as follows:

- Computation: What are the elements (UI portion) that can be distributed across different devices during the task execution?

- Communication: When the elements will be distributed means focusing on the time dimension of the distribution. There are two types of distribution: the static distribution refers to the execution of the methods which could happen during the design, development, compilation or load time, and the dynamic distribution is about the method's execution that occurs at runtime.

- Coordination: This is responsible for distributing the components and is referred to as *meta-user interface* . It controls the UI distribution in the integrated systems and the main building block, and the point that causes the difference exists in the distributed user interface framework. (We will explain this more in the following part.) This unit can be under the control of the system, the end user, mixed (the user and the system together will control the distribution) and in some rare cases, a third party (an external agent) will take responsibility for controlling the distribution.

- Configuration: how the UI will be distributed, which is related to the

representation of the UI on the destination platform, which could be the same, adapted or needs to be changed.

The proposed model will enable developers to consider different combinations of these dimensions to develop a framework capable of targeting a wide range of DUI related scenarios. While it lacks considering the user as another dimension in collaborating with the system or with other users across the system.

Besides these proposed models for handling the interactions in a multi-device environment, there are several novel frameworks and toolkits that have been designed and developed in the field of DUI to mitigate and handle the challenges that exist in building distributed user interfaces and that have also been exposed by the models mentioned above. Here we mention a few of them which are more powerful and interesting in the sense of proposed functionalities and their usage in real case scenarios, and are closer to the targets that LIQDROID is going to point out as well. While we have mentioned to some of them in more details and also a comparison with LIQDROID in the section 2.3. Panelrama [80] is a web-based framework that lets the user interact with a user interface through different devices. Their framework proposes three main features for developers: breaking their UI into some panels, automatically distributing these panels to the best-fit devices and synchronizing the state of these panels when distributed across several devices. Their proposed framework does not consider wearable devices and displays as well as sharing the content in use. Gradual Engagement [47] provides a pattern that represents the possibility of letting different proximal users with or without their devices gradually connect and interact with other device interfaces (such as a big screen) and exchange information. HydraScope [30] transforms current existing web-based applications to web-based meta-applications which they have defined it as: "a collection of application instances with coordinated contents and views that are accessible to multiple users via mobile interfaces." These meta-applications will make it possible to execute and synchronize several copies of these applications on multiple devices in parallel. Shared Substance [24] introduces a new data-oriented programming model for application developers to separate the data and functionality aspects. Their sharing mechanism is close to the application programming language, which will help the developer to better understand and apply the distribution features at implementation time. Frosini and Paterno [20] propose a flexible framework that supports the distribution of user interfaces between a dynamic set of devices based on their type and a dynamic set of users based on their roles. Instead of using a fixed server that can become a failure point, they proposed the idea of the migrating engine

which can be switched between the devices at any time.

2.1.2 Supporting Multiple Platforms

To support multiple platforms, the proposed solutions in the field of DUI should have enough flexibility to support the requirements and limitations that these devices introduce to the system. For overcoming this obstacle, they mainly benefit from web-based applications that give them the possibility of running the applications on any kinds of platforms. These web-based frameworks are designed and developed to support collaboration and teamwork between multiple users and cross platform devices such as the old or new generation of web applications supporting collaborative learning and working [3, 61], multi-player games, dividing a screen for different users and interacting with it simultaneously or distributing a task among several devices [55, 80]. These web-based approaches are strictly constrained by browser support [55]. This dependency will put a limitation in their path that restricts them in being capable of exploiting all the resources and capabilities of the devices. The vast improvements in mobile technologies, supporting mobility and being widely used by users, have made collaborative mobile applications a tough competitor for web-based applications, as they are capable of providing the services of web-based applications in much less time regarding aspects such as discovery, performance and response time [79].

2.1.3 Supporting Multiple Users

The other important aspect is the concept of having a single user or multiple users in the context. Which can be considered as the notion of social interaction coming up as the collaborative or independent interaction paradigm that will add many complexities to the final system. Kray and colleagues [39] discussed several issues that may arise in designing DUI by considering multiple users and multiple devices at the same time, which make it a difficult task, as users may want to achieve different goals that can conflict with each other, by considering the obvious fact that most of the current devices have been designed to be used individually and it is the responsibility of the underlying framework to cover the complexities that may occur by sharing the device with a wide range of users. One or several users may work separately or in parallel on the same task by using different types of devices. In section 2.1.4 we will further discuss the different contexts and types of social interactions that can happen with the presence of several users in an environment.

This aspect is more interesting while designing the frameworks to be used for multi player games or when several users simultaneously want to work on the same content. Notice that here our focus is not on the content and version control aspects but how multiple users can cooperate with each other or interact with the same device and maybe control it. Frameworks in this category [4,32] let different users simultaneously interact with the same device, for example, sharing a screen [47] to do different tasks simultaneously, and mostly have considered the position of the user as the main aspect of their design.

2.1.4 Supporting Multiple Contexts of Use

A wide range of research has been conducted in DUI to realize how different environments such as modern working, with home and public spaces equipped with various digital devices, ranging from desktop PCs and laptops to various types of mobile devices, can impose new requirements in the design and development of the DUI frameworks and toolkits. This aspect will directly influence the usage patterns, as the availability of the devices and users, and the adaptation of these devices by the users depends highly on the environment that they are involved in and needs to be preciously controlled. This is a move toward using holistic and ecological thinking for understanding and designing multi-device systems and interactions, which means that to provide a reliable and precise interaction paradigm, we need to consider it as a part of a larger ecology. More precisely, this ecology is what Nardi and O'Day defined as: "a system of people, practices, values, and technologies in a particular local environment" [52]. This whole ecology can be considered as the context of use which defines the complex relationships between the existing users, the devices and the goals they want to achieve. Designing a multi-device system based on the context of use is in contrast to the other solutions that try to serve a broad range of usage scenarios, not so perfectly fitted but as much as is needed and accessible to all. The importance of the context of usage is more tangible when considering the workflows that happen in most of today's modern offices, as different professions have their own requirements and needs in the sense of data flow between the tasks performed on the various sets of devices, input and output modalities, processing requirements, and the work flow. In order to better understand the importance of the location in designing the interaction between devices, we benefit from the example that Santosa and Wigdor [64] provided to show the serial usage of several devices to reach a profession dependent desired goal. This sample use case scenarios

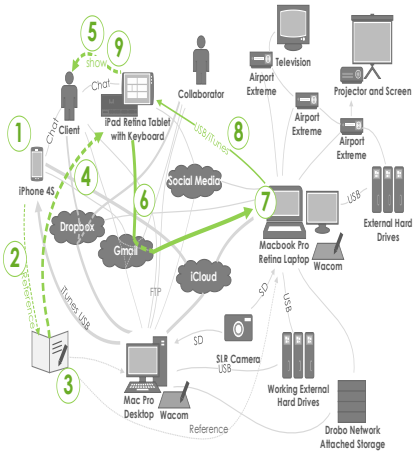


Figure 2.1: *A sample of serial multi-device workflow of an interior design profession [64].*

(Figure 2.1), showing that handling the execution of the subsequent tasks relies heavily on the outcome of the previous tasks. Figure 2.1 shows a serial execution of the tasks in a construction company among several devices. The work flow is as follows: (1) the user takes photos of the measurements and sites by using his smartphone, (2) he uses these photos to make sketches (3) of them, uses a tablet (4) to create 3D mockups using an application installed on it, as well as showing it to clients (5) in the upcoming meeting, and (6) they use an email application on the tablet to send the files to the laptop (7). They then apply some changes to the file using the software available on the laptop (8), sending the file back to the tablet via USB (9) to show to the client. The problem that arises here is that sometimes using another application, such as the email application for sending a file to another device, may change the extension of the file, and the other application won't be able to use that file as the input anymore. Also, it will be time-consuming to open another application, attach the file and send it to the other device, and on the other device open the email application again, save the file and then open the target application, load the file and continue the task. This example clearly shows the importance of multi-device interaction, in addition to the fact of how a different context of use can put complexities in the way of the interaction between the devices regarding the different expectations that they have and new requirements that they impose on the system. Certainly, one of the important aspects that needs to be effectively controlled is managing the information or the data that will flow between the users and their devices (which we will talk about in more detail in the section 2.1.5).

Another point of view in defining the context of use is considering public spaces, such as a museum, airports, hospitals, or exhibitions which are suited to a lot of users holding a broad range of devices and their availability changes continuously. Currently this high potential (availability of users and devices) that exists in these public places is not fully exploited and approached by the current solutions. For example in a museum, visitors may prefer to interact with the big screen that exists in the museum to explore more about the museum [40], while providing an interaction paradigm to support this wide range of short access requests in a consistent way would be a difficult challenge. Generally speaking, considering the context of use in the places where computing and sensing devices are ubiquitous could enhance the flexibility and completeness of the underlying infrastructure and highly motivate the user to benefit from them and increase his satisfaction.

2.1.5 Supporting Multiple Modalities (input / output)

Supporting multiple modalities on the one hand, is the concept of methods for entering the inputs into the available devices, and on the other hand, it is about managing the interactions between the devices, generally called input redirection. Input redirection is concerned with how the input entered on one device will be transferred to the other devices in the same environment [35]. The input redirection will enable the user to manipulate the content that has been shared and displayed on other devices with the device at hand. This is valuable more in situations where the user is not capable of reaching the other device or the changes that applied in a single device should be propagated among a large number of devices, otherwise it will take a long time for the user to apply the changes to each of them. For example, considering the example of a classroom where many students are using their tablets to follow the slides provided by the teacher, in this case, input redirection can propagate the changes that happen in the tablet of the teacher to all other students. This behavior will save a lot of time for the teacher that he can assign to more valuable teaching activities and can highly improve the learning procedure. Certainly, the importance of output (content, artifacts) redirection is another aspect which in most cases is much more valuable than the input redirection, as some of the current researchers have stated [78]. Managing and the availability of the data in general, which could be the output of a user performed task, or any content that the user needs to share with the other devices, is one of the main concerns in the field of DUI. This aspect will be extremely beneficial to provide a real seamless interaction between devices because, as we have already mentioned, while multiple

devices are integrated, the user needs to have access to the underlying file in the destination device to continue the work. Different approaches have been conducted to overcome existing limitations and obstacles to let the user share his files between his different devices, such as cloud-based infrastructure like Google Drive, Dropbox, and Apple iCloud. But the maximum benefit of these devices will be achieved when the user can benefit from them, not only from his personal devices, but also when the user uses a third-party device to perform a task and wants to be able to share the achieved artifact (output) on his own device. We will discuss this aspect in more detail and how LIQDROID is capable of handling it in the section 4.2.3.

2.1.6 Discussion and Comparison

Although existing models and toolkits have been developed based on the requirements that exist in the field of DUI, there is still no general design or toolkit that suits several scenarios [16]. In addition, little information is available regarding how these frameworks work in real case scenarios and comparisons about their features and capabilities [50]. This makes it hard for researchers to follow the existing works and improve other works that have been done so far. What actually happens is that, based on the use case scenarios, the requirements are defined and the frameworks have been developed. This trend will end up with several works that provide similar features, and fewer improvements will happen. As we have already mentioned, one of the building blocks of the works that have been done in the field of the DUI is providing solutions for cross platform environments, because personal computers are one of the key devices in these environments. Our work is different regarding this aspect because our main concern is providing an Android ecosystem which lets the Android applications (mobile applications) distribute to several proximal Android devices. The proposed ecosystem will enable the devices to not be restricted to only use web applications to support the cross-platform feature but to use different applications available on the devices and benefit from all the resources of the devices through them. But the other factors that we mentioned above are common between our approach and the DUI approaches. Therefore, reviewing the research that has been conducted in the field of DUI helped us to better understand user needs when located in and interacting with an environment where different kinds of devices, users, and work flows exist. In general, in the following, we mention the existing limitations in supporting the interaction between multiple proximal devices and how our proposed solution is capable of overcoming them.

- One of the main differences between our work and the above mentioned studies is that our main focus is on distributing the execution of a task on the proximal devices by benefiting from the infrastructure that application developers are already using. This solution is different from defining a new language or framework for developing applications but is empowering the current available infrastructure for developing applications (mainly Android applications) by adding the missing building block. This missed building block, in theory, is a unit that manages the existence of multiple devices, multiple users, and multiple contexts, as well as managing the underlying interaction's requirements coming in the form of input and output. In practice, this is a middleware that uses the existing features of the underlying infrastructure, the Android framework, and provides the services which are required to distribute the different parts of the running applications to the other available proximal devices and manage their execution.

- Most of the available works have been designed for earlier devices available at that time and are not capable of adopting newer technological enhancements that have happened in the computing capabilities of smartphones or the widespread adoption of new devices such as tablets, car tablets, and wearable devices, as well as being able to integrate with cloud services [64]. From the device point of view, although these new devices provide extra resources and better user experience, at the same time, they increase the level of complexity that the available solutions should be able to tackle. This means that there is an obvious need to advance the management of multi-device interactions [27].

  From the user's point of view, one of the obstacles that this new generation of devices will put in the way of the user is that they expect more professional and technical behaviour of the user. To support non-technical users or users with different levels of professions as well, the proposed system should benefit from these high-level improvements, manage the complexities that they may put in the way of the integration of multiple devices, and hide these complexities from the end user's view. The consequence of this way of designing the system is that we can distribute the functionalities that a device is capable of performing to the other proximal devices that may not be able to perform, and expand their capabilities. The above mentioned feature is one of the aspects that is undoubtedly missing in today's technologies and modern work spaces [27,64]. LIQDROID will be independent of the underlying physical device by providing an abstraction for the physical device so it

can be used by any kind of Android supported device. We will discuss this feature more in the section 4.1.4.

- As most of the currently conducted experiments show, one of the main obstacles that users still put in the way of widely acquiring the solutions proposed to support multi-device interactions is lack of comprehensive managing of the data [64]. It is desired to improve the proposed solutions to handle the data and the data flow between devices. In the section 4.2.3, we will discuss more about our proposed and implemented solution in LIQDROID for managing the final task's artifacts and sharing files between multiple devices.

- The other missing point in the current solution is the lack of handling and sharing of events that happen in different integrated devices. To fully support the synchronization between the devices, besides sharing the last state, we also need to take care of events that may happen in the devices during the distribution of the tasks, and that the user could be interested in being aware of. This aspect is more crucial to be handled when the collaboration of multiple wearable devices is required by the user to reach the desired goal [8, 9], for example, the different integrated wearable devices that a patient may use to control his health status or an athlete may use to analyze his level of activity in companion with his smartphone. We will discuss more about the necessity of propagating the events between the integrated devices in multi-device interaction's use case scenarios and how LIQDROID will handle this in the section 4.2.4.

2.2 Models and Technologies in Distributing Services

"LIQDROID computing" [26] is the best word to have used for the next technology revolution, which is the flow of user's tasks, not just data, between devices. It lets the user move the task at hand to the other devices and continue it from the last state that it has. Before we start talking about the solutions available in the industry to handle task continuity, we need to explain the concepts which are helping us to better understand the playing field and the challenges that these solutions are going to eliminate for the user.

We benefit from the categorization defined by the [14] for the service technologies which are presented to support the multi-device interactions. This categorization includes multi-channel and cross-media services. Below, we first provide a short explanation about each category and then the pro-

posed solutions. As it was difficult for some solutions to match exactly with one category, overlapping may have occurred.

2.2.1 Multi-channel Services

These are the services that enable the user to have access to the same functionalities across different devices which have different platforms (i.e., channels). These services have provided the user access to the same data synced through these different devices. So the user is able to simply move between the devices to benefit from the service. In multichanneled services, the repetition of a task is in focus as the primary requirement in multichanneling, in that the same functionality and its underlying content should be available on different platforms to ensure achieving the user's defined goals. Most of the web-based services and applications are good examples of multichannel services. As another example, we can mention the same applications that have been developed for different platforms, such as Spotify, Foodspotting, Endomondo, Airbnb, and Uber. These services provide a user profile for each specific user to save his important information and content and then share these profiles when the user is logged in to use the service, by using different applications installed on different platforms.

2.2.2 Crossmedia Services

The term "crossmedia" was defined for the first time by Dong et al. [14], according to the Filgueiras et al. [17] outline, which is "the collaborative support of multiple media to delivering a single story or theme, in which the storyline directs the receiver from one medium to the next, according to each medium's strength to the dialogue." His main focus is on the modularity of the components of a system and he considers two different categories of systems. The first category is those systems where the system's expected performance will be achieved only if all the components are fully combined. An example of these systems is the Apple iPod, which depends on desktop computers to obtain the content and the power. The other categories are those systems that provide better functionality when they benefit from higher levels of modularity. In this case, each component can work fine, even in isolation, but higher levels of the system's performance will be achieved if more components are combined and work together. He proposes a new definition for the crossmedia systems by considering the level of synergistic specificity (already defined in section 1.2.4). "Crossmedia systems are designed by establishing interoperable combinations of media around a distinct theme (space or activity) and developed with the intent of enabling

and supporting the synergistic use of their components." In contrast to multichannel services where the same functionality is needed to be available on different devices (task redundancy) in crossmedia services, the focus is on supporting a logical series of the task, although it would be even easier for crossmedia services to provide the same task on all the other devices.

The best examples of crossmedia services are the services that have been developed to support task continuity. Task continuity is the synchronization and migration of a task at any time and anywhere between multiple devices. Synchronization is the concept of keeping the data and its structure consistent by storing it in a certain place (mostly the cloud) among all the devices that have access to it. Moreover, the migration aspect lets the user continue the task from the last state that it had on the previous device. The best available commercial solutions for the data synchronizations are Apple iCloud, Google Drive, and Dropbox which all are "cloud-based" storage services. In the beginning, it was just the cloud infrastructure that made it possible to have seamless interaction between devices, but Apple's Handoff [34] has extended this possibility by also using Bluetooth and Wi-Fi Direct connections. One of the best examples of services that Apple has provided for its users is the possibility to answer a received call from an apple PC, although the PC is not potentially able to receive phone calls. In addition, the Samsung Flow [63] has also provided some level of task continuity by using the Android platform, but it is only available for certain versions of Samsung devices. Other developments such as Fluid computing middleware [6] have also developed frameworks for data replication and synchronization to let the data flow between different devices. The other commercial product that lets the user have task migration is Apple's AirPlay, which lets a user stream content (video and audio) from his iOS devices (iPhone, iPod Touch, iPad) on multiple AirPlay-compatible devices on the same network, including TVs and stereo systems using a Wi-Fi network. Furthermore, Google Cast lets the user stream video and audio to a TV or sound system through different platforms (Android, iOS and Chrome apps). So the user is able to use his mobile device as the controller of the content, which will be shown on the large screen. In this case, each device has been used because of the features and capabilities that they provide, and their integration will enhance the user's satisfaction.

In this chapter, we have designed and implemented some usage scenarios to better test and improve LIQDROID provided features on real devices and in real contexts. In general, we have proposed a comprehensive infrastructure that helps the user to better identify the resources (proximal devices) that are available in his proximity to easily benefit from them in the execution

of his current task. LIQDROID can bring for the user(s) a fully cooperative interaction between the multiple devices that can help him to perform his desired task in much less time and with higher productivity. This productivity will bring better-achieved results and also extra available time for him to assign to other tasks.

The proposed infrastructure can also conceivably improve the developer's efforts as they can easily benefit fromLIQDROID without the need to put forth extra effort in developing the requirements of these cooperative interactions inside their applications. Instead, they can put their focus on how they can acquire more from this new interaction paradigm to develop innovative use case scenarios to facilitate their users' work as well as provide a unique experience for them to attract more attention.

Besides the fact that implementing these steps separately inside of each particular application requires much more time and effort, it can also decrease the performance of the available resources and have an adverse effect on the final results. For example, consider that several users at the same time compete to distribute a task on the same proximal device. As each of them first needs to run a separate service on that destination device to be able to start the interaction, this will form two immediate problems. First is the interruption that this parallel usage can cause on the execution of the ongoing tasks. And second is that the execution of several services at the same time on that destination device to answer the request of their users, without the presence of a central unit to manage them, will highly decrease the performance of that destination device.

During the evaluation of LIQDROID on the real devices, we encountered some problems which made us review the architecture of LIQDROID and apply some changes on it. We mention just some of these problems here such as preventing the interruption that receiving the permission requests can cause while a user is doing critical tasks and is not interested in any collaboration. This made us think about providing a separate phase that will allow the user to be able to completely come out of the circle of these proximal devices by deactivating the advertisement phase. On the other hand, the Wi-Fi instability and instant disconnections while LIQDROID was collecting results sometimes caused an inconsistency in the system that we needed to handle better by monitoring the connection status.

These evaluations also reminded us that the developer may also need to have access to some of the information which is available on LIQDROID such as the list of connected devices and the occurrences of the events that happen in a device, for example when the screen goes off and on, or receiving an acknowledgement that shows the execution of the distributed

task(s) or storing the achieved results that have been handled successfully through LIQDROID, etc. So we put our focus on providing more accessible features for the developers to cover their needs.

The other concern which we always had in our mind during the evaluation phase as well as the architecture design and also the implementation was related to the changes that happened inside the Android framework and the updated versions of it that sometimes put some serious restrictions on the methods of accessing some of the previously available features. For example, accessing the Activity Manager in the previous versions was possible for us, while in the newly released updates because of some privacy concerns the Android framework has decided to restrict the access of third party applications to this feature. This had a serious effect on our work, but we solved it in another way as we have already mentioned in the chapter 5.

In general, we can say that working with LIQDROID will be very interesting and easy for the users and developers as a new paradigm of interaction between multiple devices. It introduces a beneficial way to explore the potential that already exists on the devices, whereas with the current solutions on the multi-device interactions we do not properly benefit from them. And by introducing LIQDROID, we have implemented our idea about the possibility of **direct** interaction between the multiple proximal Android devices to distribute the execution of a task and also to attract the attention of other researchers to this kind of cooperation between the devices.

Besides these works, more recently Mikkonen and Pautasso [22, 49, 75] have also used the word liquid software that has previously been used in a technical report [29] for synchronizing Java applications running on virtual machines on different computers. In their explanation, liquid software allows users to migrate tasks among devices while minimizing the configuration of and synchronization among devices. They have well-identified different dimensions that one should consider to design liquid software, and they have used them to evaluate two web-based frameworks. These two frameworks are: Liquid.js for DOM framework [76], which is designed to automatically synchronize the contents and the state of the DOM (Document Object Model), and Liquid.js for Polymer (LfP) framework [21], which facilitates the development of liquid Web applications developed within the Polymer framework. The Polymer framework has been developed by Google on top of Web Components [65]. Again, Mikkonen and Pautasso consider data and state synchronization to be the two pillars that enable a seamless flow, while LIQDROID exploits Android to materialize liquidity and distribution at a lower level of abstraction and to support more sophisticated interaction patterns. This will enable us to have the possibility to integrate the capabilities

and features of the available destination devices in the process of choosing the best match for distributing the execution of the task. As in our point of view, one of the issues that has not been covered appropriately yet, even in their work, is that most of the time users are not aware of the capabilities and features that are available on the other devices in their proximity, which can profoundly improve the time and attempt for doing a task. This reminds us that adopting the user interfaces onto the set of destination devices where the task has been distributed is just a single dimension of the potentials (software and hardware) that exist on those devices. So the proposed solutions and framework should also consider this aspect as one of the main building blocks in their design and architectures. The proposed design of LIQDROID will be a positive move toward better managing the devices' adaptation, privacy concerns and seamless interactions, as Mikkonen and Pautasso have also mentioned that these aspects need to be explored in more detail for the future direction of their work or generally for the future research in the area of liquid software solutions.

One of the important differences between the crossmedia systems and the multichannel (cross-platform) systems is that in the multichannel systems, the focus is only on making the applications run on multiple platforms or the content become accessible to them, but the crossmedia systems try to find a better resource match for the task at hand (functionality) in order to enhance the device interaction. Different researchers have focused on increasing the inter-usability of multi-device systems to increase seamless interaction and let the user easily switch between the devices.

2.2.3 Discussion and Comparison

Among these systems, Denis and Karsenty [12] have proposed a conceptual framework of inter-usability, which defines knowledge and task continuity as the two most important factors in approaching the higher levels of inter-device consistency and adaptability in multi-device systems. Knowledge continuity is the possibility of applying the experience and the knowledge that the user achieved while working with the service on the previous device to the new device, while the task continuity is transferring the task, content and its state to the new device and resuming it. This means that, besides the aspects which are important to the task at hand, the proposed system should also be able to take care of other dependencies between the device and the task. As LIQDROID has put its main focus on providing an Android ecosystem where knowledge continuity is properly supported, the user will feel that he has access to all the functionalities in a single device, though in

reality they are located and performed on multiple proximal devices.

As already mentioned, LIQDROID has been developed inside the Android framework and is only beneficial for Android-based devices. So our focus was more on comparing LIQDROID with the solutions developed for supporting cross media services. The primary difference between our work and the available solutions is that LIQDROID is not going to support the synchronization and migration of the task between multiple devices based on providing the latest version of the data in central storage, or use the approaches such as cloud-based services to share the resources (e.g., connectivity, storage, data, computation) between different devices. But it provides a mechanism to handle the challenges that may happen during the direct interaction and integration of multiple proximal devices, where handling the data consistency is just a single part of it.

More precisely, besides the task continuity that these solutions provide, there are other factors which are important and should be considered in the multi-device interaction, as Sorensen et al. [74] presented in the 4C Framework, which includes: communality, collaboration, continuity, and complementarity. This emphasizes that, along with focusing on the synchronization and migration aspects, it is necessary to also focus on devices being able to properly integrate and work with each other so that their full strength and potential are exploited. As we mentioned in the previous section 2.1, properly handling the interaction between devices to support task continuity is not just the coexistence of multiple devices in the proximity of each other, but we need also to pay attention to other factors that play key roles, such as the diversity of the available devices (in the sense of the functionalities and resources they have) and the interaction modalities.

2.3 Middleware Technologies

In this section we have provided a comparison between our proposed solution and some of the competitive middlewares that are supporting multi-device interaction, including defining the aspects that are different from their assumptions or provided features.

2.3.1 Conductor

The Conductor [27] is a prototype framework that enables cross-device interactions to distribute the user interfaces. In the following we briefly explain the services that this framework provides and also also a comparison with LIQDROID provided services.

– Inter-Device Communication through Cues (pull style):
The procedure that they are using to perform the cross device interaction is that opposite to the exiting scheme that while the user chooses the data and also the other device to apply the transition. Here they are using the concept of the cue which is the same as what exists in the musical notation, specifically for orchestral pieces, which is the signal to start the music. So in an ongoing task the user applies a long press on any item which is active in the Conductor's context. He will receive a menu that contains the "broadcast" feature. The broadcasted cue will be appeared on the other device and if the user does not provide an instant response they will be stored in a vertical list and will be hidden until the user accessing them. If the user apply tap on the cue the most appropriate action will be performed by the device which also depends on the current application which is running at that moment on that device.

– Targeting Transmissions to Devices(push style):
They also provided another mechanism that enables the user to select the target device and transfers directly the cue to that device. In this case the target device as soon as received the cue will perform its desired action. It will be useful for the cases that the target device is not accessible. In this case in order to differentiate the availability of similar devices beside providing a name for each device they also assign a unique color to it. To this end when the cue has been tapped on the new device it will duplicate the session that the source device had.

– Implicit Cues:
Beside those cues that are intended to perform a default defined action the user is also able to use implicit cues to transfer the current contextual information to another target device and resuming the work there.

– Functionality Bonding with Duets:
This is the possibility to bound the user desired task to the device that it should take place. So they need to apply a bond mechanism between the source device and the target device which is called duets. To this end the use can have access to the duets list while they have also included in each cue the identity of the source device, so by adding each cue to the duets list (through drag and drop) can perform the bound mechanism.

– Managing Sessions across Devices:
They have provided a task manager, that will include the updated screen shots of the devices which are running the Conductor. This will enable the user to be able to exchange and manage the sessions between the devices.

– Adding Devices to a Symphony:
The user is able to add a device to the set of devices available to interact (Called Symphony) through different ways such as: QR Codes, NFC, Bumping ad proxies.

The type of the interaction that they have provided has interesting features although the availability of the potential problems will be identifiable when it has been used by different users through different use case scenarios. But in the case of having a comparison with the features that they have considered to be able to manage the interaction between the devices and those that are also available in LIQDROID we can provide the following explanation of LIQDROID similar services:

The main important thing that we can mention is that one of our main focuses in LIQDROID is providing the direct interaction between the devices between needing of having any central server that manage the interaction. While in the Conductor there would be a central server that handles the devices connection and routing the messages.

On the other hand, the other important capability of LIQDROID is that the distribution of a task between the devices after that the user has selected the set of the devices that prefers and have connected to them is that the user is capable of having the list of the capable applications on the proximal connected devices that are able to perform the user desired task and is able to choose between them. This will prevent th user to distribute the task on a device that may does not have the proper application to perform that task.

Based on the study that we had we found that in general the interaction between the devices can be categorised in three main categories as: Device Shifting, Complementarity and Synced Devices that we will explain each of them with more detail in the section 4.2.2. In LIQDROID we have considered the requirements of each category in the sense of the state, data and interaction management and provide the required abstractions in the architecture. Which also include the capability of let the user control the execution of an ongoing task on the other involved devices i that interaction.

There are different ways that the user can end a session first is automatically when the devices are bot proximal to each other any more, second the user has the possibility to finish the execution of the ongoing tasks on the

other devices which are involved in the execution of the ongoing task and at last there is also the possibility to become disconnected from a device to prevent from the distributions by receiving the request from the connected devices any more.

LIQDROID has a separated module for taking care of connecting the devices which make it be able to use different connection protocols available on the devices to provide the required infrastructure to connect the devices and make them be able to communicate with each other.

2.3.2 Panelrama

Panelrama [80] is a web-based framework to distribute user interfaces and synchronize the state of them by applying minimum changes to the available languages. In general, Panelrama uses an object-oriented paradigm. In the following section, we discuss its main feature, and then we will compare its services with LIQDROID.

Panelrama introduces the concept of dividing the user interface into panels. A panel is a new XML element. If we consider that there exist different functionalities that are grouped together, the panel will surround each of these groups of related functionalities. To this end, the developer that provides the XML file of the user interface's view needs to consider for each of them a panel which is extended from the default panel object. Panelrama makes it possible to synchronize the state of these panels between the several devices, as well as (automatically) assign the panels to the device that best fits its needs. The state information, which is based on the application's logic, will be encapsulated and stored in the shared panel instead of the local device's storage. While the developer can assign different panels to general views (called *local panels*), there would also be a shared *global panel*. This *global panel* will not be assigned to any device, but the developer is able to use the sync flag to update a given state variable in the *local panels* with the general panel. So any changes that happen in the *local panels* will be synchronized with the state of that variable on the *global panel* and vice versa. These state synchronizations happen through a client server architecture. There are four different categories, defined in the Panelrama for device capabilities as: physical size, keyboard quality, touch quality, and mouse quality. There is also the possibility for the developer to add the capabilities that he needs to this list. Because of the limitations that they faced with the web standard, both the device capabilities that had been ranked, as well the panel's capabilities requirements, needed to be defined by the developer.

By benefiting from the Android framework available features, LIQDROID is able to provide the list for the devices which are capable of executing the distributed task both in the sense of the device's hardware capabilities and also the applications available on that device that can better suit the task requirements. The user is able to select the device and the proper application at run time without the need for the developer to perform any extra effort to define the required task capabilities or the device capabilities. Also, distributing the whole task will distribute the task's logic, and, along with the view to the other capable devices, will make it easier to be adopted on the existing applications and also for developing completely compatible applications. Benefiting from the Panelrama while there is strict consistency between the application's logic and the view would be difficult for the developer.

In order to be able to synchronize the state of the tasks running on different devices, we have considered the feedback and update messages. This will enable the target device to become synced with the source device while being able also to send back a response, which could be the result of that updated state in case of need. Although the devices will be connected in the peer-to-peer architecture, LIQDROID would be able to assign the controller and client role to the devices, the source device, as the controller, being able to manage the task's execution state on the other devices involved in the ongoing interaction. In addition to the state synchronization, the data availability also plays a key role in multi-device interaction, as all the devices should be able to have access to the required data, whereas in the Panelrama, the data replication has not been well described and supported.

2.3.3 Multi-Device Interaction with Dynamically Migrating Engines

The proposed solution [20] is a framework for dynamic user interface distribution in multi-device and multi-user environments without need of having a fixed server. As the figure 2.2 shows there are two different main roles in their system the Engine side and the Client side. The Engine side is in charge of handling the distribution request and the state changes. While the client side is in charge of sending the state updates to the engine and applying the state changes that it receives from the engine.

They have considered three general different states as: Invisible, Disabled and Enabled which means that a proposed part of the proposed UI will not visible in the other devices' UI or they are visible and not interactive or they are visible and interactive. They categorise devices base on their main capabilities such pc and tablet are in the same group and smartphone and

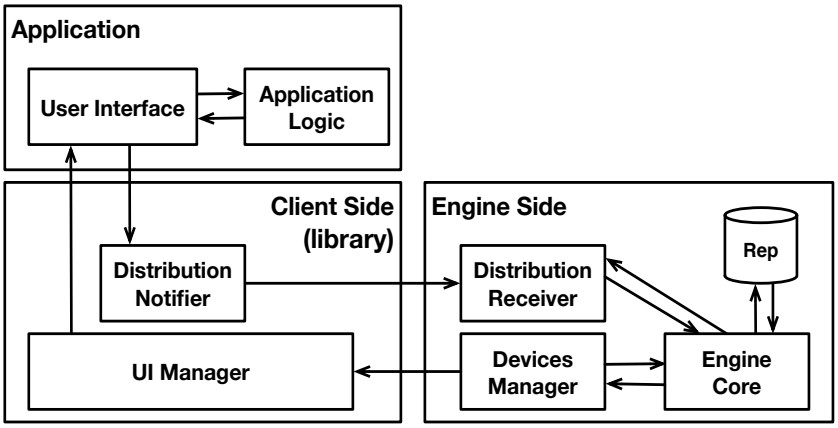


Figure 2.2: *Their proposed framework architecture.*

mobile devices categorized as another group. Each device needs to provide its credentials along with its capabilities to join a distribution to the engine. After the engine has been calculated the credentials positively will insert the device in the proper group and will send to it the distribution state. This distribution state will let the device know each element of the UI should be n which of those three general states. As long as the session is alive the device will be able to send the updated to the engine or receive the updates happened through the other device from the engine. Each session will have a time-out that if no interaction updates received by that device, the device will be excluded from that session. It is possible also to move the engine from one device to the other one while in this case all the state distributions happened so far will be sent to the new engine.

Here we mention to some of the important differences between LIQDROID and above work. LIQDROID will be able to support the execution of the proposed task by benefiting from another third parties applications on the destination devices. While in the proposed framework the same application on both sides will be executed while the different UI elements on them has been enable/disabled. This can restrict us from being able to fully benefit from the capabilities and resources which are available on the devices. Despite the device capabilities that has been considered here in LIQDROID beside the physical characteristics and features of the devices the applications which are available on them will be considered also in the process of the task assignment. So the user at run time will have the possibility to choose the device that he prefers along with the application on that device that he prefers.

On the other hand based on the context that the user belongs and the type of the task which is going to be distributed the involved user may decide to provide the credentials or not. And providing a static credentials may restricts the users' preferences. While in LIQDROID the user will be able to accept the interaction in the beginning while at any point of the interaction he is able to terminate it. Although the user will have the possibility to terminate an ongoing task on the other devices and release the resources, but the proposed solution is based on the availability of the devices in the proximity of each other so as soon as the devices go out of the range of the proximity of each other (which is based on the underlying connection protocol) the communication between them will be automatically terminated.

As well, LIQDROID is not only restricted to benefit from the UI of the other device while it is also able to manage the service distribution. Which means that multiple users at the same time are able to interact with a single device and benefit from its capabilities and resources beside the one who is using the device's UI.

2.3.4 AllJoyn

AllJoyn [1] is an open source platform and language independent framework which provides the possibility of deploying distributed applications on heterogeneous systems. To run an AllJoyn application, the AllJoyn daemon should be already installed on the device. This daemon provides the inter-process communication system, which is based on a service oriented architecture, so that the applications can register themselves to it as the service provider or the service consumer. AllJoyn uses a virtual bus to connect multiple AllJoyn daemons and also bus attachments. The bus attachment is an application that is connected to the AllJoyn bus. Each bus has an interface which defines a group of bus methods, bus signals, and bus properties, along with their associated type signatures. There are some standard interfaces that the AllJoyn framework implements, but in practice they are implemented by client, service, or peer processes. The companion applications can be written in different languages, such as C/C++ and Java, and through the AllJoyn daemon are able to communicate with each other. AllJoyn provides a mechanism to advertise and discover the available services when the devices are in proximity to each other. To this end, each bus attachment will have a unique name and a well-known name. The unique name is assigned by the AllJoyn as soon as a bus attachment is connected to the bus. The well-known name will be assigned to a service upon its request, which will

be used for the advertisement. The service consumers are able to apply the discovery and find the proper service through the well-known names that have been advertised. In order to prevent the inconsistency that may happen when there exists different implementation of an interface, to differentiate them, additional information will be added, which is provided by an object path. In order to have access to the services which are provided by the other bus attachments, a bus object is required that is the implementation of the given bus interface. If the developers have provided the bus, it would be easy for them to implement the interface, otherwise they need to provide an introspective request to have access to the definitions of the interface of that bus. After the bus has successfully been created between the bus attachments (service provider and consumer), the remote method call and receiving signals can be accessible through a proxy bus object.

One of the main points of our proposed middleware infrastructure in comparison to the AllJoyn is that working inside the Android framework makes us capable of benefiting from it as much as possible, to handle the execution of the tasks and the inter-process communication. This enabled us to provide a more consistent system in the shape of a connected ecosystem, and to be able to provide wider services for the developer and the end users. This consistency between the proximal devices plays a key role in providing a real seamless interaction between multiple devices. By using AllJoyn, the interaction between the applications is started when the two applications have been launched on both sides, attached to the created bus, and are able to communicate with each other. While benefiting from LIQDROID as soon as the two devices which have LIQDROID already installed on them are in proximity of each other, the user can launch a LIQDROID-compatible application on one side and initialize the interaction by distributing and controlling the execution of the task on the other device, without the need to follow extra steps in advance to integrate the applications.

Furthermore, developing LIQDROID inside the Android framework will have another advantage more valuable for the developers. Working with LIQDROID will be much easier for them, as they won't need to learn so much new information to provide the infrastructure to make applications able to communicate with each other and also with LIQDROID. They will be able to follow the usual procedure to develop an application, and by adding a few lines of code, they will be able to make their application compatible with LIQDROID and distribute the execution of their applications on the proximal devices.

The other difference is that we don't need to specifically call a LIQDROID-compatible application on the other side to execute the requested

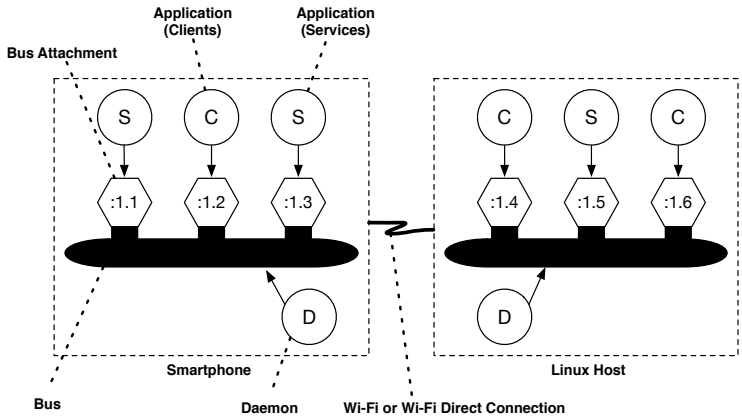


Figure 2.3: *Architecture of AllJoyn bus.*[1]

task. LIQDROID is able support the interaction between multiple devices through benefiting from the available applications in the Android market. Its maximum potential will be exploited if on both sides the applications are compatible.

In LIQDROID, there would be two different steps for the discovery. The first is advertisement and discovery, which happens for connecting the user and selected proximal devices together. In this step, the devices will transfer a message between each other, which includes the device, such as its name, its type, and the user defined name, while it will also include the other data which will be required to connect the devices together. We will discuss this more in the section 5.1.

The second discovery happens when the user wants to find the devices and the applications available to him through all the connected devices which are capable of performing the user's desired task. LIQDROID will retrieve all the required information from the Android OS and the applications which are going to be used. LIQDROID does not need to provide any extra information for the purposes of advertisement or discovery. Moreover, the user will always have access to the last updated list of the capable devices (both supporting the hardware and the software availability, which are the applications) to distribute the execution of the task on the device which better suits their needs. LIQDROID benefits from different connection protocols such as Bluetooth, Bluetooth Low Energy, Wi-Fi and an ultrasonic modem for the advertisement and discovery phase, while the second phase of the

[1] `https://allseenalliance.org/framework/documentation/learn/core/standard-core`

discovery and the task distribution will be supported by Wi-Fi. It should also be possible to add and use other connection protocols in the future.

Besides the solutions mentioned so far that are platform-independent, there are also middlewares that have developed inside the Android platform to support the collaboration between the devices. This category of services enables developers and users to easily adopt the features that these middlewares are providing, as they are already familiar with the Android platform and how it works. The main focus is providing the underlying features for distributing the applications among several devices in a way that resembles the local Android behavior for inter-application communication on a single device.

2.3.5 Sip2Share

Sip2Share [7] is a middleware working within Android Framework. It supports to create a peer to peer local network to provide and request services in Android. It provides a network of mobile devices to publish, discover and invoke the services which are available on them.

In order to publish and discover the services the peer needs to do it through the super-peer which is a class in charge of matching the sender request and the service provider. After the super-peer found the proper match, it will send the address of the device to the requester peer and the peer can start interacting with the service providers directly. The figure 2.4 shows a general view of the operations and the figure 2.5 an example of the interactions among the peers and the super-peer to publish and subscribe for a service.

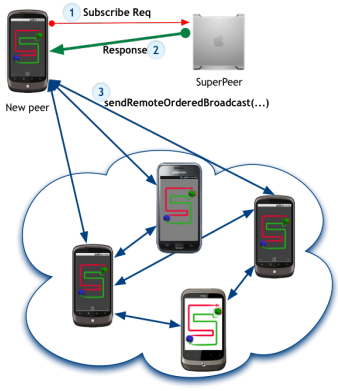


Figure 2.4: *High level operations [7]*

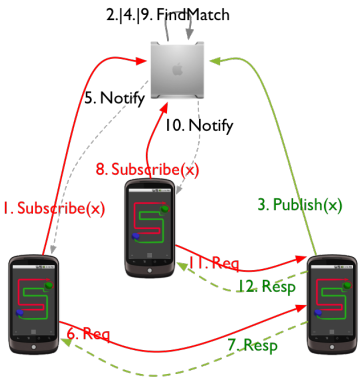


Figure 2.5: *Possible interactions [7]*

But the difference between the Sip2Shsare and our work is that LIQDROID uses the Android OS available features to provide the list of the capable installed applications that can handle the user's request. For this purpose LIQDROID benefits from the procedure that the Android OS follows to provide the Chooser list. The Chooser list is the list of applications that the Android OS shows to the user when there are several applications installed on the device that are capable of performing the same action for the user. Considering this example that you want to open a desired URL and you have already installed different browsers on your device such as Chrome and Firefox. As soon as you click on the URL link, the Android OS will show you a list which contains the Chrome and Firefox and lets you choose the one you prefer, the mentioned list is called Chooser list. LIQDROID will follow the same procedure to handle the user desired action, it will send the user's request to all the connected devices and receives back the list of the capable applications (the appropriate applications' components) installed on the other devices. After receiving back all the responses, LIQDROID will prepare a list from these responses and show this list to the user. The user is able to pick one (or several) of the application (s) available on the connected proximal device (s) that he prefers to perform his desired task. In this procedure all the connected devices have the same role and we don't need to have a super peer which is in charge of matching the requests to find the provider. We will provide more technical description of this feature in the section 5.5. The other difference between our work and sip2share is that it is not capable of sending a large size of data to the several devices as well managing the artifacts that are achieved from the execution of the service on the other devices. On the other hand LIQDROID lets the user be able to call any android activity or service at run time without the need for registering these components in advance. Besides LIQDROID is able to handle the synchronization between the applications' components distributed on the other devices (the state of the executed activities or services on the other devices) and accessibility to the results (artifacts) that the execution of these components may provide for the original user and independent of the device that the task execution has took place.

2.3.6 Remote Service Call

The Nakao and Nakamoto proposed a middleware [51] for handling the remote service call to provide cooperative work environments. By the proposed work they aimed to find out the feasibility of extending the inter-process communication mechanism in Android to do the remote service

invocations without applying any modifications to the Android applications. The proposed work provides the following features: launching a component of the application which is called Activity in Android (we will explain it in more details in the section 3), sharing screen system and sharing the data. In order to share the screen, they have provided a prototype which uses the Drawable class, Surfaceview class, and VNC (virtual network computing, deprecated from Android 1.5 because of security problem). Sharing the screen by using the VNC approachis based on sending only the differences between the image that is showing on the two devices but the other two approaches send the image along with an array of the coordination information and the position of that piece of the image on the screen. By the resource sharing they are focusing on those data that the Android apk file includes and are stored on the storage of the device at install time. Their approach to sharing these stored data between the devices is that instead of loading these data from the application file they download it from a server and use them while launching an application on the other device. The difference between their work and LIQDROID is that we are not only launching an application component on the several devices, more than just two devices that they used, but also we handle the synchronization and status changes that may happen during the execution and distribution of those components. At last, LIQDROID is also capable of sharing the resources not just those that already are included in the apk file of the application but also those data that are the results of the execution of the components between the connected devices. For example, the user may want to take a picture by using the camera of Android device B and continue the task by using device C for editing the picture received from device B (The editing application is not available on device B). At the end receives back the final result from device C on his own device. In this case the image is not already available in the apk file of the editing application on device C but is the result of the execution of the user desired task on device B and is fundamentally needed to be available on device C to let the user continue the task at hand to reach his desired goal.

2.3.7 Middlewares on Android Binder extension

Another category of the works that implement the multi device applications interactions are those that are based on extending the existing Android Binder [37, 44, 56]. The Android Binder lets the applications which are running in two different processes can interact with each other to call each others functionalities (remote method call) and pass arguments between

each other. We need to mention that in Android each application is running in its own process sandbox that is assigned to it at install time which is for supporting the security, stability, and memory management reasons. If two different applications want to benefit from each other's functionalities, the Android kernel is in charge of binding these applications together and let them to inter to each other processes. The communication in Binder framework is a client server model. For the security reasons, the client does not know exactly the address of the Binder, so each binder needs to register itself as a service in the Service Manager of the Android. So when the client wants the service, it will ask the Binder address of the requested service by only knowing the name of the service from the Service Manager. Extending the binder mechanism to enable the execution of the client and the server processes on two different devices requires to handle the added complexity overhead related to the application registration mechanism and also application permissions to access to the device resources [56].

Besides the solution as mentioned above to handle the interprocess communication, there is another mechanism which we will explain it in the following. Another way of enabling the applications which are running in different processes in Android frame work is using intents (we will explain it more in details in section 3) which provide a point-to-point as well as publish/subscribe messaging mechanism. LIQDROID uses this mechanism of providing the interaction between the applications' components that are distributed and running on different devices. Because first, we don't want to change the current behavior of the Android operating system through extending the binder which will put extra power consumption and performance overhead. The second reason is that the binder mechanism requires providing an AIDL file in the client and the server side (remote service app) during the apk creation and must be shared between them. The AIDL (Android interface definition language) file includes an interface with the methods signatures of the remote service which is defined by the developer and is used for two purposes [67]. The first reason is that it creates a proxy class that lets the client app has access to the service and its methods, and second it creates a proxy class which will be used by the service to redirect the method calls to their implementations. While in the intent mechanism the other component that we want to call and use only needs to provide the required intent filter (we will explain it more in details in section 3) to be accessible from the other components that may require its provided functionalities and it is not needed to provide any files or application registration in advance. On the other hand, LIQDROID takes care of the artifacts that the execution of the application's components will create on the destination

devices as well as it manages the synchronization required between the distributed application's components are running on different devices.

2.3.8 Google Play Services

The Google has provided different services that enables the devices and applications installed on them be able to interact with each other. But still these services are not capable of exploiting the maximum capabilities that the multi-device interaction can propose to the user. These services such as the Google Nearby Message API [2] that we will explain it in more detail in the section 5.1, will enable the proximal devices be able to discover each other and interact with each other through sending messages. Or the Google Cast Android API [3] provides the experience of the multi screen for the user, so the user can select a content on his smart phone and view it through the big screen. While the user needs to have the Chromecast device attached to the big screen. Or the possibility to share the content provided by the applications installed on different devices through the Cloud Storage [4] that we will explain it in more detail in the section 5.6.

But what is lacking among the available services is that we need to have an comprehensive infrastructure that multiple-users and multiple-devices (any type of Android devices) be able to **directly** become integrated and interact with each other (without need to have any device in between). To this end, LIQDROID is the proper infrastructure that will facilitate the multi-device integration and also solve the complexity that may occur to synchronize and manage the state of these devices during the interactions.

LIQDROID will provide the possibility to add these services in case of need to improve the handling of the required features such discovery of the proximal devices through using the Google Nearby Message or handling the data management by using the Google Cloud Storage while at the same time it will provide the building blocks to distribute the Android OS between a set of proximal Android devices to let the user distribute the execution of a task on them.

2.4 Conclusion

We found several research contributions in the field of multi-device interactions, which shows the increasing interest in this topic. However, despite this interest, still, there is a big concern from the developer's and designer's point

[2] https://developers.google.com/nearby/messages/overview
[3] http://www.androiddocs.com/google/play-services/cast.html
[4] https://firebase.google.com/docs/storage/

of view, as they want to distribute a task (may be handled by different parts of an application) to other devices and let users have access to them [59]. To facilitate this concern, we proposed LIQDROID, which let them identify the different possibilities and distribute them by using the features that LIQDROID proposed to them. LIQDROID will facilitate the discovery and communication of several proximal Android devices, enable interactions between multiple users, cross-device applications and manage their interactions at runtime. Although the other works put their main focus on the cross platform aspect, our main focus is on supporting the heterogeneous types of only Android supported devices, as we believe that this will enable us to provide more comprehensive solutions by benefiting from the powerful infrastructure that Android OS has as the building block, and put our attempt on improving and managing the interactions, instead of extending the current functionalities or redesigning them. Furthermore, providing distributed applications should not need extra effort from developers. Also, it should be able to support the already available applications (with or without minor changes required) with fewer changes.

In the end, what is more important to notice is that the users do not really care if the application is a mobile application, web application, or hybrid. What they do care about is being able to reach their desired goal in a fast, smooth, efficient, and delightful user experience that helps them in their activities. So our first attempt in developing LIQDROID was not just proposing a new way of interacting between multiple devices but offering the user the ability to easily move between the nearby devices to perform a task and reach his desired goal with more quality in much less time and attempt, which, in the end, would make him have a better experience handling the tasks. To the best of our knowledge, LIQDROID is the first middleware that had specifically been designed for proximal Android devices and is able to support the distribution of the task execution between devices with a close focus on concepts such as managing interaction and synchronization at run time, and design a more robust, seamless flow between devices. In the end, this will enable the user to consider multiple devices as a single unit (an Android ecosystem). Contrary to the existing models (such as cloud-based services), the devices involved in the interaction can have roles both to offer a service and use a service [15], without the need for registration in advance. In this way, LIQDROID goes further and lets the user, instead of just requesting a service (executing part of an application on the proximal devices), integrate them, and manage their artifacts and execution, which means that they are able to use the output of a service as the input of another service. LIQDROID will be running in the background and all the

installed applications will be able to benefit from the services that it provides, simultaneously, without blocking each other's or another user's ongoing task.

CHAPTER 3

Background About Android Framework

As LIQDROID has been implemented inside the Android framework and is going to use the features and functions that are already available in it, we need t first provide some definition about them. By considering these definition first it would be easier to get what of the Android features have been playing the key roles in our design and implementation phase. Second it would be easier to understand and follow the technical details explanation which is provided in the next chapters 4 and 5.

3.1 Android Application Components

In Android, an application is a package of components, and components are the entry point of the application. There are four different categories of components, and each category has its own scope and lifecycle.

3.1.1 Activity

Almost all activities are responsible for creating the user interface or window to interact with the user and receive the inputs from him.

Activity's lifecycle

Each activity has its own lifecycle and the activity's state changes by the user's interactions. In general among the available states in the lifecycle of an activity that the figure 3.1 shows, there exists four different states which is in our interest as follows:

- Start: When the activity enters to this state the Android OS will invoke the onStart callback. This is the state of the activity when it is in the foreground and has the user's focus. So the user is able to interact with the device and provides inputs for the activity.
- Pause: If the user launches another activity, or use the home button or high priority events happens on a device will cause that an activity goes to the background and loose the user's focus. At this state the activity is still able to receive the broadcasts but not the inputs of the user. But in the cases that there is not the necessity that the activity becomes responsive it is better to release all the resources that it acquires i=during his execution as well the broadcast receivers.
- Resume: This is the state that the activity that was previously in the pause state acquires when it comes back to the foreground and takes the user's focus. So instead of launching a new instance of that activity the Android OS will bring back the previous instance for the user.
- Destroy: The activity will goes to this state when it has been directly through the user or the activity component itself(finish()) is killed or the Android OS has been killed it to release the resources and to save space.

LIQDROID as a third part application is not able to have access to this state changes of the application that it has been launched on the other devices directly through the activity but we will explain more in the section 5.7.2 how LIQDROID is able to be informed and use these state changes in his architecture.

Android Task and Back Stack

Each application can have none (a service) or several activities and the user can move between these activities by using intents. The sequence of activities that the user launches and move between them to achieve a demanding goal is considered as a general unit which is called task. The sequence of the activities that have been launched by the user will be stored in a stack which is called back stack. In this stack the current activity which

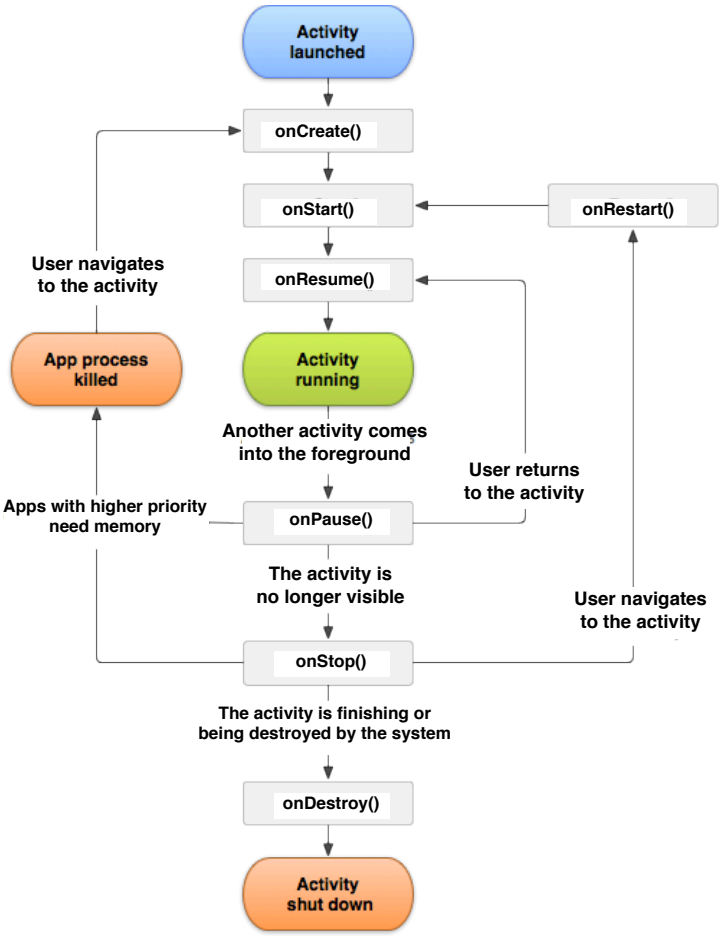


Figure 3.1: *A simplified illustration of the activity lifecycle.* [1]

is in the foreground will be the first item. This order helps you to be able to use the return hard key on your device or the one that is available inside the device to go back to the previous activity. when the user or the Android Os kills an activity that activity will be removed from the stack. On the other hand, if you launch an activity through the home screen, which is currently in the background so in this stack, depending on the component's configuration a the instance that already exists in the background will be resumed or that instance will be killed and a new instance of it will be created again.

Considering that you are writing a message to your friend through the

[1] `https://developer.android.com/guide/components/activities/activity-lifecycle.html`

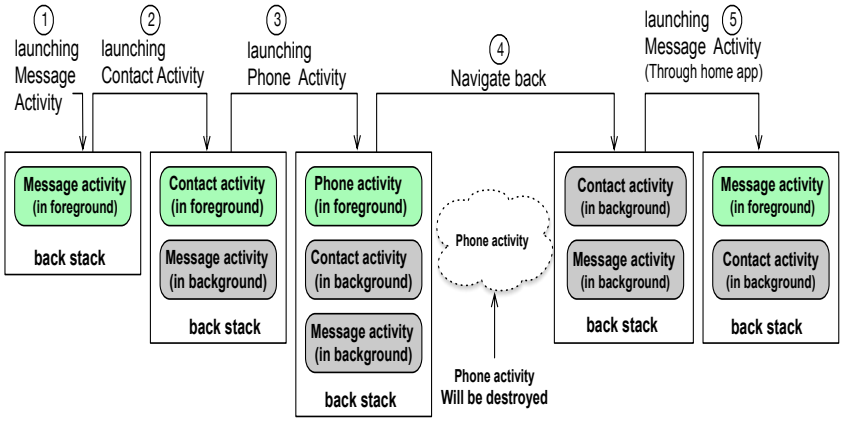


Figure 3.2: *A sample view of how the activity task stack works in a device while the components belong to the same package or different.* [2]

message activity. Then you decide to call him instead, so you launch the phone application, which first shows you the list of the contacts through the Contact activity. Then you select your friend's contact and by clicking on that the Phone activity is launched which shows you the contact info of your friend. As it includes the old number of your friend's you decide to go back to the contact activity and choose another contact name of her. As you can not find her new number so you go back to the message activity to take her number from there. As the figure 3.2 shows these different steps through the changes that happens in the back stack.

3.1.2 Service

The component which is running in the background and remain active even if the application is not under the focus of the user. It may also offer its functionalities for third parties applications. Service does not have any user interface, and in the case of need, an activity will be bounded to it to let the user interact with it.

Service's lifecycle

Although the lifecycle of the service is simpler than the activity but it is much more important because it is not in the focus of the user and if it does not handled properly will decrease the performance of the device.

[2] `https://developer.android.com/guide/components/activities/tasks-and-back-stack.html`

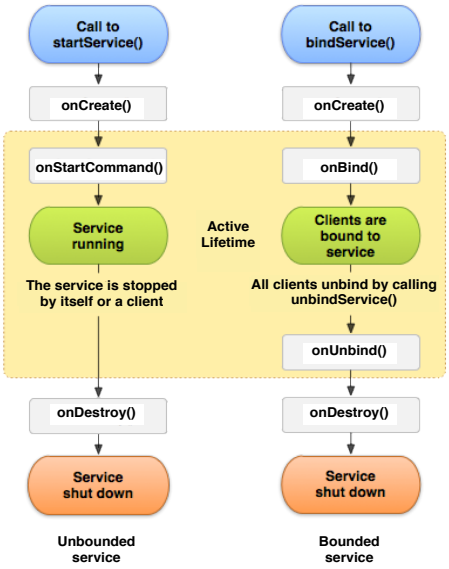


Figure 3.3: *The service lifecycle. The diagram on the left shows the lifecycle when the service is created with startService() and the diagram on the right shows the lifecycle when the service is created with bindService().* [3]

The figure 3.3 shows the life cycle of the service in two different situations:
- Unbound Service: While an application component has been started the service through using the startService() method. In this case the service would continue to run in the background, even if the original component that has been started it is destroyed.
- Bound Service: While an application component bind itself to a Service through using the bindService() method to perform interactivity and inter process communication. In this case, the service will be stopped as soon as the component binding to the bound service is destroyed. When a component binds to a Service it retrieves a communication interface that can use it for sending requests and receiving responses within the process or even across processes.

The difference between the bound and unbound services is that: An application component starts the service, and it would continue to run in the background, even if the original component that started it is destroyed. You will use the startService() method to start an unbound service.

[3] https://developer.android.com/guide/components/services.html

3.1.3 Content Provider

Each application is able to store its data in the file system through Android OS or create an SQLite database, on the web, or any other persistent storage location. If the owner of the application wants to share its data with other applications, will provide a content provider and let other applications to use it to query or even modify its data. If the data is stored on the storage of the device (public access), the application does not need to create a content provider to load the data in the application but the general content provider inside Android framework will do it on behalf of the application.

3.1.4 Broadcast Receivers

This component is responsible to receive the broadcast announcements that the application is interested in them. There are two different types of broadcasts: first, called system-wide broadcasts, which are general purpose broadcasts such as announcing that the screen has turned off, the battery is low, system boots up, the device starts charging, etc. Second, those broadcasts that are created by other applications, in this case, Android OS will capture and will again broadcast them, those applications that have been registered for them in advance will catch them.

3.2 Android Inter Process Communication

3.2.1 Intent

Each application can be launched directly or instantiated through another application. Whenever a new application is launched or its component is started a new process will be assigned to that application. Because of privacy concerns, Android runs each application in a separate sandbox to protect its process and data. If an application wants to use a functionality which is provided by the other application's components, it cannot directly activate that component and need to acquire permissions that have been defined by the other application in advance. These permissions let the Android operating system enter to the application's process and activates the component which is requested by this application.

For better managing these inter-process communications among applications, Android introduces the INTENT, which is a passive data structure holding an abstract description of an action to be performed. Intents are asynchronous messages and a universal mechanism for calling another component inside an application or outside, along with the required data. Each

intent has a field which is called "Action" and make intents distinct from each other. Intents are used for starting the activities and services and also delivering a broadcast.

There are two types of intents: Explicit and Implicit intents. Explicit intents are used when you exactly know the name of the component (the fully-qualified class name) that you want to send the intent. But in the Implicit intent instead of defining the name of the component, you define the generic action to be performed for two reasons: first, you don't know the exact class name of that component (as third parties may have developed the component that you require). Second, when you are interested in calling all the components which are able to do a certain action, for example, to view an image and you want to choose one among the applications that are able to do it. Which in this case Android OS will show a dialog under the name "Chooser", includes a list of installed applications capable of performing that user desired action and let the user pick the application that prefers.

3.2.2 Intent Filter

Is a short description of the capabilities that an application is able to perform. On the other word, what an activity or service can perform and which are the type of the broadcasts that the application is interested in receiving. When an application creates an implicit intent, the Android OS will compare the action of the intent to the intent filters which are declared (inside a component or in the manifest file) in each application, if there is only one application that supports that action, the OS will launch it, otherwise will pop up the Chooser dialogue.

3.3 Conclusion

We need to mention also that from now on we will use the term component or directly the name of the components such as activity or service to mention to the application's part as currently we are completely familiar what exactly they mean. The component includes also the content provider and the broad cast receiver as we mentioned above but in the concept of distributing an ongoing task is mainly concerned about execution of the activity and the service which the user is able to interact with them and execute them to reach his desired goal.

CHAPTER 4

Proposed Middleware Architecture

This chapter, we present the idea of the multi-device **direct** interaction through presenting the architecture of LIQDROID, which is a middleware infrastructure that will enable the user to seamlessly distribute the execution of a task on a set of proximal Android devices. In order to better understand how this direct multi-device interactions can enhance the users' achievements and satisfaction in performing their daily tasks, we have provided a concrete sample use case scenario of distributing the execution of tasks while there exists various sets of devices, users, contexts and interaction types that we will explain each part of it in the relevant sections (4.1.1, 4.1.4, 4.2.3 and 4.2.4.)

The fully cooperative multi-device interaction proposed by LIQDROID will capable the user to seamlessly distribute the execution of a task based on the capabilities that this task requires to the capable devices while he will be able also to manage its execution. To this end, the figure 4.1 shows the architecture of LIQDROID to propose the above-mentioned features and the procedure of handling the execution of a distributed task. We proposed a layer-based architecture to enhance the procedure of managing the requests and the interaction between the multiple devices and solving the challenges that may occur during the task distribution. So by benefiting LIQDROID

the user will have the possibility to discover devices in his proximity, put them in different groups, and distribute the execution of his desired task on the members of his desired group. While during the interaction still he will be informed about the occurrences of the events, manage them and at the end of the execution have access to the achieved results. LIQDROID is implemented as an Android service which will continuously run in the background and its execution will not cause any interruption in the normal execution of the other applications' components on the device that are not going to use LIQDROID or, generally speaking, are not compatible with LIQDROID. Therefore, the user can let LIQDROID run in the background without the need of quitting it while he wants to use other applications. On the other hand, several LIQDROID-compatible applications are able to send requests in parallel to the LIQDROID, and LIQDROID is capable of serving all of them while this multi-tasking will not cause any interference in delivering the responses back to the components which have sent the requests. We will discuss this aspect and how LIQDROID can handle this situation more in depth in the section 5.5.

LIQDROID will be a service which is running in the background and will benefit from the Android framework to provide some services for the running application. LIQDROID will receive the requests for handling the distribution of the execution of a task through a LIQDROID-compatible components. To this end, whenever a user wants to execute a new task (such as launching a new component) through a running LIQDROID-compatible component, this task execution request will be received by the LIQDROID instead of the Android operating system of that device. The LIQDROID will first ask the user that in which device he prefers to execute or resume the task, through providing him a list of capable proximal connected devices. After the user has selected the proper device, the LIQDROID will ask the Android OS on the selected device to take the responsibility of executing the user's desired task or handling the rest of the execution of the task that has been handled previously by the Android OS of the previous device. If we can point to it more precisely, from the step number 8 the Android OS of the destination device will become responsible to take care of the execution of the distributed task. In this way, LIQDROID, by distributing the Android OS between a set of proximal connected devices, will form a bigger Android ecosystem that a task can easily move among its elements. This ecosystem will change the current behaviour of single-user single-device interaction to fully cooperative environments in the sense that in the proposed environment a single user is able to seamlessly distribute the execution of a task to the proximal devices, or different users can actively participate in the execution

of a task on a single device through using their own personal devices or the device at their hands.

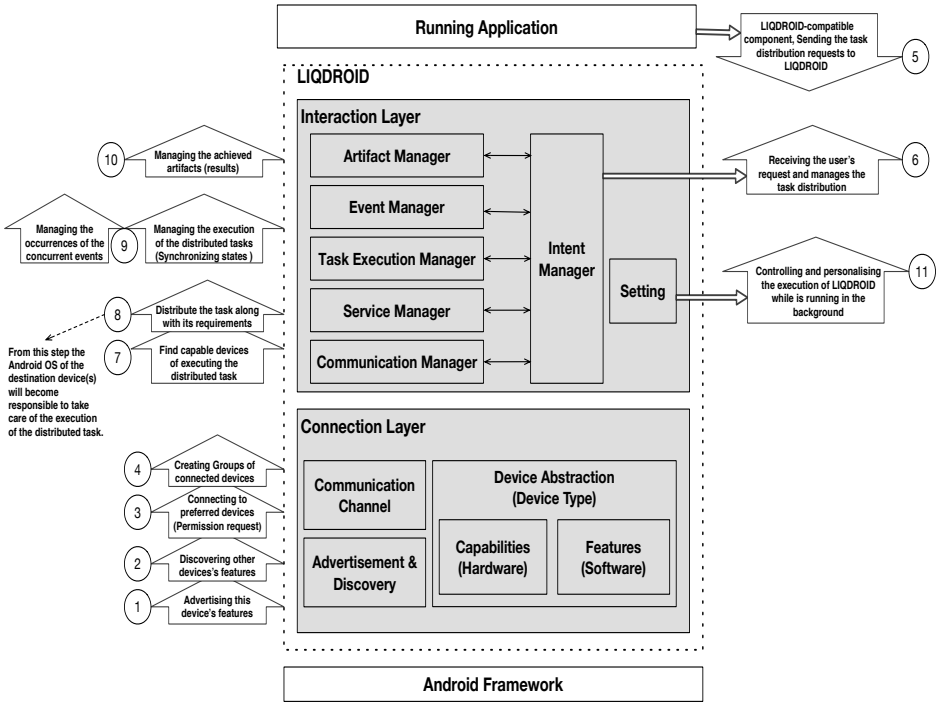


Figure 4.1: *A General Overview of LIQDROID Proposed Features*

4.1 Connection Layer

This layer is in charge of providing the primary infrastructure to connect the proximal devices to each other in order to enable them to communicate with each other later. This layer aims to provide some preliminary steps before connecting the devices and starting the communication in order to, also, engage the user's preferences in the connection procedure. In the following, we will discuss more these different steps and the importance of considering them.

4.1.1 Advertisement and Discovery

Considering that you arranged to meet your friend after working time in a certain location that you received through a message. Before leaving the office, you launch the map on your smartphone with the received location. While you are checking the path, you reach your car, and you transfer this

map to the tablet on the dashboard. You navigate through the path using your car, and you reach your friend. While you were up to say goodbye, suddenly you realized that you did not show him the video related to your last demo of the current project. So you start the video player on your smartphone, but he says that he needed to leave but before that, he asks you to send the video to him so that he can watch it in his path to home. Therefore you start to discovery his device through your smartphone and while you found it you select it and resume the video there.

As soon as the user enters a new environment such as his home, office or a public place, he needs to discover the available proximal Android devices. This will let the user have a chance to be informed about the device availability and select those devices that he prefers to connect to. By the user's preference we mean that, based on the task at hand and considering his assumption about the appropriate device (the resources which are available on the device) and the context where he is located in, he may prefer or trust on a different set of devices to connect to. For example, in a public place, the user who wants to receive information regarding the place may prefer to connect to a big screen provided by the place, that is more probable to have the information that he requires, instead of a foreigner's smart phone. The LIQDROID is able to discover the Android devices ranging between smart watches, smart phones, tablets, Android Car or TV which LIQDROID have been already installed on them. As long as LIQDROID is running in the background, the device features will be advertised to the proximal devices through an advertising message. The content of this message is the information about the device capabilities (Software), resources (Hardware), and information which will be used later for the connection purpose. When a user joins this context and starts discovering the proximal devices, he will receive the list of all the proximal Android devices which are advertising themselves; each item of this list includes the name of the device as well some information about it such as the type of the device, its battery state, and the brand (manufacturer company) of the device. This information helps the user to select the device(s) that he prefers to establish a connection with. When the user has selected the device(s) to connect to, an authentication message will be sent to those devices; upon receiving the acceptance response from them, devices will become connected and the user is able to start the communication and use them to distribute the execution of the task at hand. From now on, we will call the device that has sent the connection request as the *source device* and the other devices that the user is connected to them as the *destination device(s)*. The communication between the *source* and the *destination* devices is one-way which means that, although devices

are connected, the destination device is only capable of answering to the requests received from the source device and is not able to send any request to use the source device for executing any task on it. This will enhance the privacy to prevent the execution of unwanted tasks on the source device and, also, the inconsistency that may happen if the source and the destination device want to distribute a task simultaneously and, more precisely, when the task is regarding launching activities. Because if at the same time the destination device attempts to launch an activity on the source device, the activity that the user on the source device was interacting will go to the back ground. On the other hand, if the user does not want to receive a connection request which sometimes in the public places would be annoying instead of quitting the service which could stop the execution of the ongoing tasks, he is able to stop the advertising feature that we will discuss more about it in the section 4.2.7. Accordingly, the device will not appear in the list of the discovered devices.

4.1.2 Group Formation

Up to here, the user is able to send a connection request to the nearby devices and connect to them. But to have a better overview of the connected devices and also the possibility to select the proper set of devices to distribute a task on them, it is possible to filter out the list of the connected devices in the source device as a group. Thus, the user is able to create a new group or add members to his previously created groups. This will be useful when the user wants to have a different set of connected devices based on his own interest points such as a group of his personal devices, in-home available devices and another group for the office devices.

For example, as the next phase of the scenario mentioned above (in section 4.1.1) considering that you have arrived home and you also decided to show the proposed video to your family. So you discover the set of devices available at your home including your family member's devices to find the best set of devices to use to play the video. So you select the devices that you prefer, and you create a group between them. So you start the video player on your device while you decide to transfer the view part of it to the Big Android screen. Also, you send the controller part to your wife's smartphone so she can pause it through her device when she needs that you provide some explanation to understand it better. Suddenly you receive a phone call, and you leave to answer it while your family continues to watch the video.

The user created groups, by default, are private only for the user who

has created them but if he prefers he can also advertise them. Advertising a group will let the newly joined members find the device that they want to connect to by searching among the members of an available group instead of going through the list of all the available devices. This approach will be very practical if there exists high numbers of Android devices available or if the user has familiarity with the place that he just joined and the devices or people available there. So, instead of searching long lists and selecting the desired devices the user can select the proper group(s) and it is more probable for him to find the devices which are in his interest through the members of that group instead of checking the list of available devices one by one to find the proper devices.

These are not the only usage of the group formation, as in the next steps while he wants to distribute the execution of a task, he has the chance to only execute it on the members of an existing group instead of all connected devices. In this way, all the ongoing propagation of the requests that the source device provides and LIQDROID will be only limited to those connected devices which are members of that group. As a result, if you want to execute a task on your personal devices along with some data, you are able to ask LIQDROID to only execute the ongoing tasks on that subset of the connected devices by selecting the personal devices' group. Which could also provide some level of privacy. While still the user has the possibility to regret the group selection and execute the task on all the connected devices.

4.1.3 Communication Channel

Plainly, connecting the devices is one of the main steps in enabling the multiple proximal devices to be able to interact with each other but the main responsibility of LIQDROID as a middleware infrastructure to handle the execution of the distributed tasks will start after, when the devices become connected. To this end, LIQDROID is entirely independent of the way that devices are going to be connected to each other as a consequence it is possible to consider different connection protocols to distribute the task at hand. Currently, due to time constraints we have only used Wi-Fi to connect the devices and to test LIQDROID but it is possible to add and use different connection protocols for pursuing the communication purposes such as Bluetooth, BLE, Wi-Fi Direct, etc. in the future.

Regarding the official documentation of the Android for implementing the Bluetooth connection [1] we need to perform the Bluetooth discovery and then apply the connection mechanism. As we have already implemented

[1] https://developer.android.com/guide/topics/connectivity/Bluetooth.html

our own mechanism for advertising and discovery, there is not the necessity to implement the Bluetooth discovery. To this end, the only thing which is required to be added to the proposed message we will explain it in more details in the section 5.1 is the device's address (MAC address). Based on the Official Android document, to connecting devices we will need to set up a server socket and accept a connection which needs around 26 lines of code and then for initiating a connection with a remote device as a client we need around 27 lines of code. The rest of the process of sending and receiving the message from the socket will be similar to the one that has been already implemented in LIQDROID. For the Wi-Fi Direct just changing a few lines of the code will be enough to support it. Supporting different connection protocols can empower the user to the benefit of them in different contexts based on his preferences to save energy consumption, preserve his security or remove constraints that may exist in the availability of them on the proximal devices. As it would sometimes become challenging for the user to find different possible connection protocols and select among those which better suits the needs of the undergoing interactions, it would also be nice to consider a mechanism that can help the user to offer the existing possibilities and let him choose the one he prefers.

The connection step happens after the advertising and discovery to pursue the communication between the devices. After the user has selected the devices, a connection request will be sent to them and if they accept they will become connected and the user is able to communicate and then distribute the execution of the tasks among them. To send and receive the advertising message and the connection request/response we have used the Google Nearby Message API [2] that we will discuss about it in the section 5.1.

4.1.4 Device Abstraction

To handle a wide range of possible scenarios instead of considering the Android device type, we have provided an abstraction of the devices including the definition of the devices based on the capabilities (hardware) and features (software) that are available in each of the devices. By the capabilities we mean the availability of the hardware resources on the device such as the camera, different available sensors, the screen size, etc. and by the features we are targeting the installed applications' components on these devices. When the user asks to distribute a task to the proximal devices LIQDROID will find the list of the application's components on the connected devices which are *capable* of performing the user desired task so the user can choose

[2] `https://developers.google.com/nearby/messages/overview`

them to resume the task's execution there. Comparing the task requirements with the device capabilities and features will have two benefits for the system and the user: First, there could be several components on a device which are able to handle the user desired request and all of them will appear in the list beside the name of that device so the user has the chance also select the component that he prefers more on a device. Second, there could be a device which is already connected but does not have any component to support the execution of the distributed task so it will not appear in the list.

The other advantage of this design approach is that it also enables to tackle the existing issue in connecting several types of Android devices at the same time together as there is already the limitation between connecting the wearable devices at the same time to both a smart phone and a tablet. However, by using LIQDROID we do not consider a wearable device as a separated category of the Android devices while it will be a part of a bigger Android ecosystem that it has created with the other connected devices and is able to interact through LIQDROID to the tablet and the smart phone at the same time. For supporting this aspect, it is only needed that the two devices have the possibility to connect to each other through Wi-Fi and LIQDROID will handle the rest of the communication process and the distribution of the task between them. More generally speaking, it will also enable LIQDROID to be effective for the new incoming Android devices in the future as they will not be considered as a new device type while the only thing that matters about devices in LIQDROID is the device's capabilities and features.

4.2 Interaction Layer

As soon as devices get connected the user can start the interaction between the devices and distribute the execution of a task. This layer oversees handling the interaction between the connected devices to be able to distribute the execution of the tasks and manage them during the execution. In the following sections, we will explain the modules that are considered in this layer and their usage more in detail.

4.2.1 Intent Manager

As we explained in the section 3.2.1 one of the ways to interact with the other Android components as a third-party application and enter to their processes execution is through sending intents. And this would be the way that LIQDROID as a third-party application is able to interact with other components. So, all the interactions between the installed components and LIQDROID will be handled just through exchanging intents. The intent

that the LIQDROID-compatible components send to LIQDROID should include the data that which is required to handle the task distribution. The provided data should obey the rules and format provided by LIQDROID. The LIQDROID will benefit from this information to interpret the requested features, properly distribute the task based on them and finally be able to manage it during the execution. As soon as LIQDROID receives an intent, it will interpret it to find out what is the user's request and will call the module which oversees handling the request or providing the results.

The execution of the intents will be still under the control of the Android OS and LIQDROID will only transfer the intent and assign the execution task to the Android OS of the device that the user has chosen to distribute the execution of the task.

4.2.2 Task Execution Manager

This section makes the actual difference between LIQDROID and the works that we have mentioned them in the sections 2.2.2. The user is able to send a distributed task execution request to the connected devices and receive the list of capable components available on them, then he can choose those that he prefers and execute the task on them. This step is a kind of runtime service registration that happens also on the other solutions as they first register all the services and then, when the user sends a service request, they handle it if the service exists. While in LIQDROID the user will have always access to the last updated list of the available services based on the changes that may happened on the devices' availability or the applications installed on them. Nevertheless, after executing the task on the destination device(s), we need also to take care of the execution during the interaction and it is done.

More precisely, an important part of our work is concerning about the mobile devices (except Android TV), which causes that they join and leave the system at any point of the time. The underlying system should be prepared to end the execution of the task when the requester device is not available any more or vice versa, if the destination device has left. The proposed issue was only a brief mention to the contents of what we will talk comprehensively in this section. In order to better understand how the interaction between the devices can go on, we have considered three main different categories; the interaction between the devices enter to three different phases: Activation phase, during the interaction which we will consider it as synchronization phase, and the termination phase.

Activation Phase: The user is connected to some proximal devices and

want to distribute a task to them. The LIQDROID will first ask the user if he prefers a certain group or he wants to have the list of the components on all the available devices. As soon as LIQDROID receives the request for launching an activity or starting a service, it will provide a list of the capable components available on the connected proximal devices as well as those components available on the user's device. The user selects one or several of the components and LIQDROID will execute the task on them. From now on based on the application logic the user is able to use the device at hand to manage the execution of the ongoing task on the destination device(s) or is able to directly interact with task on the destination device(s). The LIQDROID has provided the infrastructure required for the both use cases and we will explain these possibilities in the following sections.

Synchronisation Phase: Based on the user's desired goal and availability of the proximal Android devices, interaction between the user and these devices can have different forms. The user may want to distribute a task to several devices and works in parallel or may use the devices one after the other to complete the task and reach his desired goal. So applying the synchronization between the devices mostly depends on the type of the interaction which is going on and is application's scenario dependent. The LIQDROID can support the following categories of the interactions between multiple-devices:

- Device Shifting:
  Occurs when the user decides to resume the current activity on the other available proximal device for any reason such as it has some required resources which are not available on the device at hand, the battery of the current device is dying, or wants to leave the current context. As the user currently is holding different types of devices with himself the device shifting is one of the main concerns from the solutions propose by the industries as we mentioned in the section 2.2.1.

- Complementarity:
  Occurs when the user prefers to share a component between all the other devices in the proximity. The type of the interaction between devices will be changed from the default one which is peer to peer to the Controller (master) and Client (slave) nodes; it means that the device that has been started the interaction will become the Controller and is responsible for sending different parts of the activity to the other devices (Clients) and controlling them.

  Complementarity is mostly important in doing the tasks that are not

possible to be performed by using a single device or at least the performance of the user will decrease. For example, consider that you have attended to a Skype call meeting through your smart phone and you receive an email, so you should whether discard the email to check it later or check the email while the other attendance will lose your video or you will lose the presentation totally for the time that you put the call in the background.

The other example can consider the fact about the performance degradation; consider that you receive an email with an attachment and you want to reply back to the email by considering the information available in that file, so every time you need to open the attachment and memorize piece of the information, then open the email and write them down.

Yet, by using multiple devices you can have the Skype call or the attachment of the email in one device and through distributing and resuming the task to the other device you can check your email or reply the email more easily.

- Synced Devices:
  Occurs when the user prefers to open the same or the consequences activities on different devices and update the state of all the activities right after it happens on the source device (the one which user is directly interacting). The difference between the Synced devices and Device Shifting lies more on the use case scenarios. In fact, in the case of Device Shifting the same activity would be shifted to the other device so more probably the user is not interested in keeping the activity alive in the source device while in the synced devices, the same/different activity is running on all devices at the same time they will be updated together.

  One of the best examples to exploit the good opportunities and benefit from multiple devices is that considering that your friends and you have gathered together and want to organize a trip, instead of opening the desired places in different tabs of the browser you can launch them in your friends' devices and let them check different things related to that place such as hotels, best places to visit and the prices and in a few minutes you can synchronously check all the interested places, collect the result and make your choice. The other example is useful when there are several connected devices that the user needs to synch their states; for example in the shops or the public places that they advertise the same product so the user is able to control the execution of the

ongoing task on the other devices through the one which is accessible or is in his proximity.

Managing the execution of the activities on the several devices lets the user be able to interact with different activities at the same time (in parallel). This feature is not currently supported in the stand-alone usage of Android devices as the activity that the user wants to interact should be in the foreground and no more than one activity can be in the foreground of a device at any moment.

Termination Phase: Introducing the mobile technology completely changed the way that the user interacts with the devices. Because he is not forced any more to stay in a static place to follow the execution of a task and he is able to move between the environments and use different sets of devices to continue the ongoing task and reach his desired goal. However, this feature will introduce other challenges to the system and solving them requires the continuous monitoring of the user's status and more precisely the availability of the devices.

Consider that you enter a museum and you want to interact with the big screen that is available there. The big screen already supports the multi-devices interaction (LIQDROID has already installed there) so you are able to control it through your smart phone and find the place of art work that you want to see. You find its location and you leave the screen without going back to the home page. The other visitor comes to the museum and wants to interact with the big screen; at this point, the big screen should be capable of accepting the new interaction and the new visitor should feel that he has the possibility to interact with the big screen by reading the instructions provided in the home page. The LIQDROID is capable of handling these kinds of situations which is needed to release the resources first, by updating the list of the connected devices or second, by providing the alert to the user to terminate the task. In the second case, LIQDROID provides the required abstractions that as soon as the user puts the current activity in the background and starts interacting with another activity, is possible to notify the user about the ongoing activity execution on the destination devices and receives his preferences to either terminate the activity there or let it continue.

Because of the limitations that exist in the Android framework for the third parties' applications about the lifecycle of the components, this feature is only available for LIQDROID-compatible applications. As the privacy of the Android does not let LIQDROID finish the execution of any other components which do not originally belong to its application package. The termination phase will help the connected devices release the resources

which are not used by the user any more. This will improve their battery and performance as well as preventing any interference that executions of different components may cause to each other. Also, will make the device be ready and fully operative for the next users.

4.2.3 Artifact Manager

We continue the scenario mentioned above from the point reached in the section 4.1.2, considering that you suddenly receive a phone call and you leave to answer it while your family continues to watch the video. When you came back your son says that you made a mistake in spelling your name at the end of the video. So he asks you to send the video to him so he can edit it immediately through one of the powerful edit applications that he has on his tablet. So you select the previously created group to have the list of the available applications to edit video on the connected devices which are members of this group. You select the one that your son prefers and launch the video editor on his device with the current video through your device. He starts editing it and applies the required changes. Tomorrow at the office you remember that you did not check the corrected version of the video last night. So you open the video player, download the version that your son has provided, and you see everything is perfectly fine.

We have provided the Artifact Manager Module in LIQDROID to facilitate two obstacles that exist in the multi-devices interaction. The first is providing the possibility to share a storage between all the integrated devices to facilitate the task distribution. This storage will be used for providing the underlying data to be used by the distributed task as well the results that will be achieved by the execution of the task. On the other hand, it will remove an obstacle that exists in distributing the Android OS on a set of proximal devices. In the following parts, we will discuss more these two obstacles and our proposed solution.

Most of the time the storage limitation may cause the user not be able to execute his desired task although everything else is working perfectly fine. Considering the following use case scenario that you are going to a vacation with your friends and in the middle of the day, you notice that your smart phone's storage is full and you are not capable of storing any picture or video anymore. This situation motivated us to develop a mechanism that let the user exchange data between mobile devices and external storage systems in a transparent way. This mechanism will enable the user to be able to share the storage between the devices which means that if there is not enough free storage space available on the user's device and he is not able to capture any

image anymore as he already received a notification which says the storage is full. Thanks to LIQDROID the user can use his tablet to distribute the capture image task while LIQDROID will store the result of the task and the user can receive it whenever he arrived at the hotel and could manage to have free space on his device. This will be useful when the user is travelling and does not have access to any nearby external storage, but there exist other proximal Android devices.

LIQDROID will provide the possibilities for the user to either receive the result back on the source device instantly or later or use it as the input data in the next request. The big obstacle that exists in the available solutions which let the user share the data between several devices is that the user needs to use the same credentials on all the involved devices. But as we mentioned currently, LIQDROID will enable the user to also benefit from the devices which do not belong to him to execute a task with the required data. Thus, instead of letting each LIQDROID-compatible component to handle its data, LIQDROID is responsible to manage the data involved in the execution of the distributed tasks. So, when the updated data mentioned recently is needed by the third device, LIQDROID on the third device is in charge of loading the data that has already been uploaded to the central storage by LIQDROID on the second device. We will discuss more about the privacy concerns and the concurrency handling in the section 5.6. The proposed mechanism will also support the homogeneity of the data provided by one component which will be used by the other one(s), and LIQDROID will not make any changes in the file extension or the content while storing or loading the data. The following example will better demonstrate the advantage of sharing the storage between the connected devices; assume that you want to take an image and send it to your friend, but your device does not have enough memory, thanks to LIQDROID you are able to use the other proximal Android device to take the image, send it to the other device which has better application to edit the images and then send it to your friend.

The other obstacle that we needed to solve in distributing the Android OS was providing the required data for the execution of the distributed task on the destination device. As we mentioned in the section 3.2.1 it is possible to send the required data along with the intent to the other components. The current mechanism that the Android OS uses to enhance the execution of the components is that instead of sending directly the whole data from the source component to the destination component, it uses the address of that data acquired in the storage. Therefore, the destination component is in charge of, firstly, loading the data from the storage and then executing the

intent. Loading the data happens through using the content providers that we have explained in the chapter 3. To better understand the procedure consider this example that you want to open an image through a newly installed application to show images which support enhanced features; the gallery application sends the URI of the image along with an intent to the mentioned component and that component will load the image first and then shows it inside the activity to you. The LIQDROID follows the same procedure to handle the execution of the data on the destination devices as we will explain in the following. When the user wants to launch a component on the destination device with the required data which will be loaded in the component from the source device's storage, LIQDROID will take care of transferring this data to the destination device and will launch the component along with this data. For this purpose, we need to provide a mechanism in LIQDROID to load the required data before the execution of the intent. On the other hand, when the execution on the destination device is happened, it may have changed the state of the data or have produced a new version of that data. This updated data by default will be stored on the destination device's storage while the user may need it in another place (the source device or another connected device).

Since the context that we are considering is highly volatile and the devices can join or leave at any time the data storage should not depend on the existence of any particular device. Otherwise, it will cause that user loses his interest because, in one hand, before he wants to leave a context, he needs to calculate the time required by a task execution to provide his desired artifact and then leaves the context. On the other hand, it may happen that the user is forced to leave the context before the task provides the artifact while he does not want to interrupt the task execution and wants to receive the final results later or maybe from another place. As an example consider that you were in a meeting with your colleagues to edit a map, shared on the big Android screen, at the end of the meeting you needed to leave the office immediately to pick up your child from the school; so you did not have enough time to wait until the changes were applied successfully and downloaded the last version of the data on your device. Thanks to LIQDROID at night you are able to run the LIQDROID-compatible component that you were using in the meeting, the last version of the underlying data that has been stored by LIQDROID will be loaded inside the component, and you can continue working on it.

The other advantage of managing the data through LIQDROID is related to the time that multiple devices need to have access to the same data

simultaneously. In this case the device that owns the data, transfers it to the other devices (sequentially); the source device will ask LIQDROID to execute the task using the required data. The LIQDROID will store the underlying data in the cloud and the other involved devices will load the data for that device (in parallel) and execute the task simultaneously. This mechanism will significantly decrease the time required for preparation of the multiple devices to perform a synchronous task. So all the devices have the possibility to have access to the underlying data at the same time and the execution of the tasks on them can happen simultaneously.

4.2.4 Event Manager

In the concrete use case scenario that we have proposed so far, we mentioned that you suddenly receive a phone call and you leave to answer it. The question that may appear here is that is there the possibility to use the proximal devices to answer the phone call? And how the user can manage these types of concurrent events that may happen during a task distribution? In this section, we provide more detail about how LIQDROID is capable of managing these events concurrencies.

The proposed Android ecosystem will let the user experience a consistent interaction between the multiple proximal devices. One of the requirements of this consistent interaction is that the system should be able to support the occurrence of the concurrent events that happen in each of the involved devices that have been shaped the custom Android ecosystem for a user. Based on the effect of the ongoing event LIQDROID sometimes needs to enter to the execution of the task or apply some changes to the provided features to better control the system. In LIQDROID we have considered two different categories for the events, the first is device related events which are those that are related to the hardware features of the device such as the Android input events or system events which we will call them as "Device Level Events"; the other category is related to the changes that may happen during the execution of the components installed on the devices such as the state changes that happen in the life cycle of a component which we will call them as "Interaction Level Events".

Device Level Events: The LIQDROID enables the propagation of the events between the devices involved in an interaction to better manage the execution of the ongoing distributed tasks. Sharing these events can make the definition of the bigger Android ecosystem, that we explained in the beginning, more sensible. Because as the occurrence of an event in a single device will enable the user to perform proper actions to prevent

unwanted interruptions that may happen on the execution of the ongoing tasks, knowing these events that happen on the integrated devices to perform an already distributed task will have an important effect on the coherence and seamless integration of the ultimate Android ecosystem and will cause valuable contributions. In one hand LIQDROID will benefit from these events in the process of making decisions or executing the task such as a low battery event.

To better clarify the usage of event propagation we provide a usage scenario already implemented in the LIQDROID. Considering that a low battery event happens in one of the user connected devices, in this case when the user wants to launch an activity on one of the connected proximal devices, for sure he will not choose the device that has low battery because it is more probable that in the middle of the interaction the device will be turned off. Thus, when LIQDROID receives this event from a device, although devices are still connected, LIQDROID will exclude this device from the list of the available proximal devices to distribute the execution of the next task. On the other hand, the device level events are important for the application developers to activate or deactivate some functionalities. For example, events such as the screen goes OFF or ON, may cause some changes in the behaviour" of the application.

The other aspect we may consider in this category is the input event redirection where the input events will be received in the source device can be transferred and handled by the destination device(s). So when you are interacting with your device the consequence effect of this interaction will be appeared on the connected devices. Considering that you are controlling a video which is played in an Android big screen through your smart phone. When you click on the play button on your device, this touch event will be transferred to the big screen and will be interpreted by the video player activity there, and the video will start playing.

Interaction Level Events: Devices can have different states at starting or during an interaction with the other proximal devices in the sense that they are already busy by executing an activity or they are free. We emphasize on the activity component because at each moment it is not possible to have more than one activity in the foreground while there can be several services running on the device at the same time. It may happen while a task is executed on the proximal device, a concurrent event happens inside one of the source or the destination devices, same as receiving a message, a call, a notification, etc., that LIQDROID needs to enter to the task execution and manage the situation to prevent the conflicts that may arise. In this case, to better understand the user's needs and apply his preferences, when a new

event arrives LIQDROID will notify the user about the current execution state of the task on the connected devices and provides some suggestions for him to handle the situation. After receiving the user's preferences, LIQDROID will apply the required changes based on them.

There could be some situations that the user is not able to provide any choice or the device is not accessible by the user, so we have defined a time threshold in which LIQDROID will wait for the user's choice and the suggestions will be accessible for the user. If the user does not provide any response at the proper time, LIQDROID will discard the event as it may not cause any conflict and leave the execution of the task to be continued as it was.

To better clarify LIQDROID proposes procedure, consider the use case scenario that you are using an activity on an Android TV to play a video while using your smart phone to control it. Meanwhile, you receive a phone call. The following situation may happen based on your preferences:

- You may want to pause the video, answer the call (through your device or another proximal device) and resume it again when the call has been finished.
- You may want to pause the video because you prefer to answer the call through using the big screen, If there exists the possibility to distribute the call activity to the Android TV.
- Or you may want to let the video continue without interrupting it while you are answering the phone through your smart phone.

To handle the possible situations mentioned above, LIQDROID will provide a suggestion menu on the source device and will ask the user to choose the one that he prefers. After receiving the user's choice, LIQDROID will treat with the distributed tasks based on it.

4.2.5 Service Manager

Here, by the service we mean the service component that is available in the Android framework and we have talked about it in the section 3. Although in the previous parts we have more focused on distributing the execution of the activities between the proximal devices, in this section we have put more our focus on considering the distribution of the execution of the services to the proximal devices. The importance of the services is that they are capable of running for a long time in the background without interrupting the user or occupying the user interface which will cause their execution have much

lower power consumption. This will enable the user to run multiple services in the background together using different sources and providing different features for the user.

As we mentioned before, in order the user to be able to interact with a service an activity should be bound to it. The LIQDROID makes it possible for the user to use an activity on his device to interact with a service which is running on another proximal connected device. This means that the user of the source device can launch an activity on his device and bind it to a service existing on a connected proximal device to start the service, execute its methods and receive the execution results. As an example, consider a music player service which is available on a proximal Android TV, the user is able to discover it through LIQDROID, connect to it, send a music from the storage of his smart phone and start playing it on the TV while he is able to control it through the controller activity launched on his smart phone.

So, different devices at the same time can bind activities to different services which are running on a single device and benefit from the results that the execution of the services provides. This is opposite to the way which works for the distributing of the activities because when the user distributes the execution of an activity to a destination device, other proximal devices are not able to run another activity on that destination device simultaneously unless the previous activity has been terminated or gone to the background.

4.2.6 Communication Manager

The architecture that we have used for handling the connection between the devices is peer-to-peer which means that all the devices are able to directly communicate with each other. As the context is highly volatile, this will help the system to prevent from depending on a single device that would become a failure point for the system. While, by the roles that we can assign to the devices during the interaction, we can make a device be capable of controlling the execution of a task on the other connected devices involved in an interaction in case of need. We will discuss more how LIQDROID can handle this aspect in the section 5.5.1.

This module will also oversee applying the authentication step and check the connection status of the devices and update the list of the connected devices in the source device. Thus, when the user applies the discovery he will have access to the last updated list of the available devices, groups lists and groups' members list excluding those that are not in the proximity of the user any more. And the user will have the possibility to have the devices that recently joined to the context in the list of available members and be

able to connect to them.

4.2.7 Settings

As LIQDROID is an Android service which is running in the background, it may happen that the user wants to interact with it to change something while he participates in different situations or contexts of use. The proposed setting will enable the user to quickly have access to the important features in LIQDROID and be able to control them instead of requiring to totally stopping the LIQDROID's execution. These accessible features will be as follow:

- Stop Advertising:
  Because of the dynamic nature of the mobile devices and the possibility that they offer to the user to be able to interact with the ongoing task while he is moving between different places, a user may join different contexts at any moment. As we said the advertising mechanism that LIQDROID is supporting lets the devices advertise themselves continuously and become discoverable by the proximal devices. But there could be situations that this mechanism will cause some inconsistency and inappropriate experiences for the user. For example, in a public place with the existence of a wide range of Android devices the user may become bombard by connection requests which may disturb the user and prevent him from using his device appropriately. So the user is able to stop the advertising mechanism whenever he prefers by accessing the setting menu of LIQDROID and activate it again later instead of completely terminating the LIQDROID's execution on the device.

- Start Discovery:
  The user can start discovering the proximal Android devices whenever he wants to become connected to them to distribute the execution of a task or to connect to the newly joined device(s). Whenever the user starts the discovery, an updated list of the available devices will be shown to the user. Besides this list, the user also will have access to the groups' list that he has created them previously or the list of the groups that are advertised by the other proximal devices. More precisely devices are advertising themselves continuously, so the user is able to discover them whenever he desires. Manually starting the discovery will have several advantages, first, to trigger the list of the proximal devices and show it to the user only when the user is interested in, this will cause in less user distraction. Second, as we are working in a dynamic situation that many devices can join and leave frequently,

performing continuously discovering should happen in short time slots and this behaviour will put a lot of unnecessary overhead on the device and consume a lot of energy.

- Change Device Name:
  As the user may be placed in different contexts and acquires different roles based on them so he is able to change his name. This will enable the users of the other proximal devices to better recognize him and connect to him in case of need. For example, you have invited one of your colleges to join a meeting in your office; as soon as he enters your office he starts the discovery and finds the proximal devices through LIQDROID. As soon as he finds your family name in the list of the available devices, he sends a connection request to you; upon your acceptance he is able to connect to your device and have access to the service which is running on your device to download the presentation's resource. While in the evening you may join a family party and you change your device name to your first name to be easily discoverable by your family and friends. Later, you receive a connection request from your cousin who wants to receive the images that you have recently taken. You accept his connection request and let him run the activity to view the images and select those that he wants to have them. The proposed name will come along with the other information of the device that we have mentioned before.

These settings options will be accessible by the user through the notification bar (which based on the device could be on the top or bottom of the device's screen) and the user is able to interact with the LIQDROID even while the other components are running in the user's device and occupied its screen.

4.3 Conclusion

In this section we have explained the reasons for the different modules that we have considered in the architecture of the LIQDROID. These modules will enhance the creation of the multi-devices interaction by solving some of the challenges that already exists in this field. Working inside the Android framework also had some effect on the proposed architecture and the considered modules that we will focus more on those aspects in the next chapter.

CHAPTER 5

Implementation Details and Technical Descriptions

In this chapter, we will talk more about the technical details of LIQDROID and how different modules that we have discussed in the section 4 have been implemented to propose the desired features and how they are working with each other. LIQDROID is implemented in Java, which is the language that is widely used to implement Android applications. The first expectation in implementing the LIQDROID through the proposed modules was to benefit as much as possible from the features that are already available in the Android framework to distribute the execution of a component, more precisely the activity or service, on the connected proximal devices. Working inside the Android platform was beneficial for us from two points of view: first, the LIQDROID is able to utilize the components of the currently available Android applications in the market in the process of distributing the execution of the user desired task; and second, the applications will only need very simple changes to become fully compatible with the LIQDROID-to be able to both send requests to the LIQDROID and accept the execution of the user desired task. So, the developers will not need to gain much additional knowledge to make their old or newly developed applications compatible with the

LIQDROID. They only need to obey the rules in providing the required data for the LIQDROID to manage the task execution. In the following section, we will discuss more precisely these LIQDROID-defined rules.

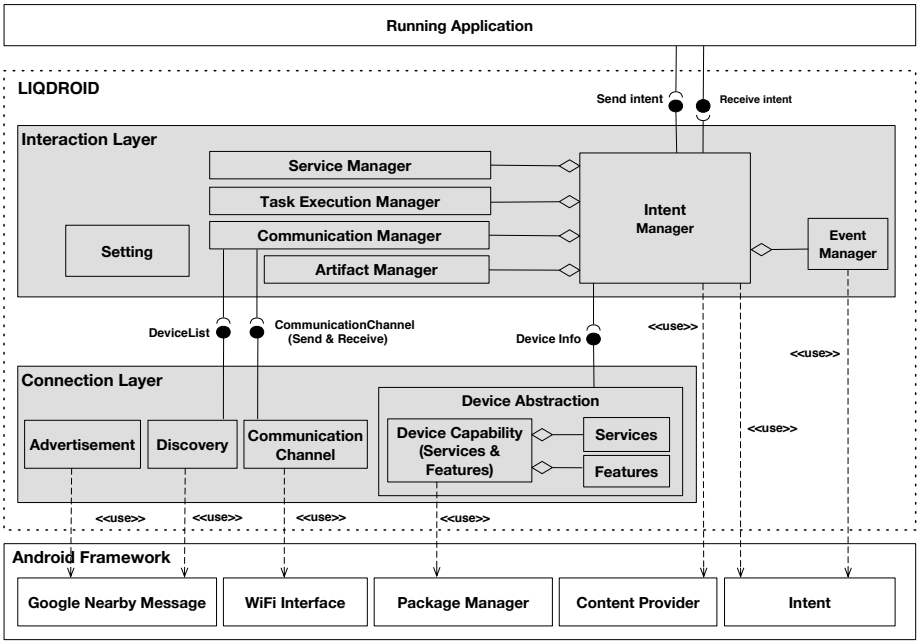


Figure 5.1: *Architecture of LIQDROID (Modules' dependencies)*

Undoubtedly, one of the important aspects of LIQDROID is its ability to properly support the connection step between the devices. After properly connecting the devices, LIQDROID is capable of offering a consistent Android ecosystem to the user, and the user will be able to exploit the features brought to him by this ecosystem to have a better experience in executing a task.

5.1 Advertisement and Discovery

The LIQDROID will be entirely independent from the connection protocol for sending the intent to the other proximal devices; therefore, we need to provide our own mechanism to support the discovery and advertisement of the device capabilities and the information which is required to connect the devices. In the advertisement phase, devices will advertise a message that includes the device name, device type, the Wi-Fi network name under which the device is currently connected, the IP address and the socket port number (we will explain its usage in the next section, 5.3). The device name and

type will be used to inform the user about the existence of available devices in his proximity so that he can connect to those that he prefers and deems as safe. For transferring this message to the proximal devices, we used the Google Nearby Message API. The Android devices will typically advertise their features, and the other devices are able to discover them upon applying discovery.

In order to connect to the devices that are available and preferred in the user's proximity, he needs to apply the discovery. When a user starts the discovery by accessing the setting menu of the LIQDROID, a list of the nearby devices will be available to the user. This list contains some of the information extracted from the message that has been sent to the other devices by using Google Nearby. Currently each element of the list contains the name of the device, the type of device and the battery level. Therefore, the user is able to choose whichever device that he prefers that suits his needs based on this information. When the user has selected the preferred devices to connect to, a connection request message will be sent to those devices through Google Nearby. This will enable the users of those devices to decide whether they agree to let the other proximal devices use their devices or not. Sometimes the user needs his device, or he is in a public place and does not trust foreign devices, or the device is already interacting with other devices. In such cases, by rejecting the connection request the user is able to prevent these interruptions. The source device will receive the connection responses from the selected devices through Google Nearby and the following scenarios may happen:

- Reject: If the other device does not accept the connection request, upon receiving the reject response, the LIQDROID will notify the user that they will need to select another device(s) to connect to.
- if the destination device has accepted the connection request, the two devices will become connected through the Wi-Fi that we will discuss in the section 5.3.

The figure 5.3 shows how the Google Nearby Messages work, which involves five steps. Take, for example, that two smart phones are in close proximity of each other. The process would be as follows: (Step 1) Device A registers online with the Google Cloud and stores the message that it wants to advertise there; then it will receive a token (key) to advertise with a predefined tag, which here is "LIQDROID". We have put this message in the above-mentioned information that will be advertised by the Android devices.

(Step 2) In order to receive the advertised data, the other device needs to subscribe to that data by providing its tag as "LIQDROID". (Step 3) In order to recognize that the devices are in proximity to each other, Google Nearby will use a wide variety of connection protocols which may be available on the devices, such as Bluetooth, Bluetooth Low Energy, Wi-Fi and an ultrasonic modem to transfer the token from Device A to Device B. (Step 4) Device B will use this token as a key to get access to the match data, which is the original message stored in the cloud. (Step 5) The Google Cloud will deliver the message sent by Device A to Device B. So, the exchange of all the data transferred between the proximal devices will go through the cloud and the data will not actually be sent peer-to-peer. For this reason, the Google developers have strongly suggested that to be able to transfer the message faster and to be sure that the other device will receive it, you should keep the size of the message less than 3KB; whereas the maximum message size that it can currently support is up to 100KB. So, in order to properly manage the synchronization as well as manage the data exchange between the proximal devices, we were forced to implement another layer to support the communication and the data management between the devices, which we will explain more in the section 5.3.

Through LIQDROID, all the devices will be continuously connected to Google and will advertise themselves so that the newly joined devices can be easily discoverable by the other devices without the user having to activate the advertisement procedure manually, unless they have disabled it previously.

5.2 Group Formation

After that LIQDROID on the source device has received the acceptance response of the connection request, it will provide the possibility for the user to put them in a group. The user will have a dialog box which will provide the following options for him to choose from:

- Create a new group with this newly connected device and define a group name.
- Or add it as a new member to available groups. In this case the group will be advertised with the updated information (the new member).
- Or to continue the communication without creating any groups.

The user created groups are defined inside each device, which will provide following advantages:

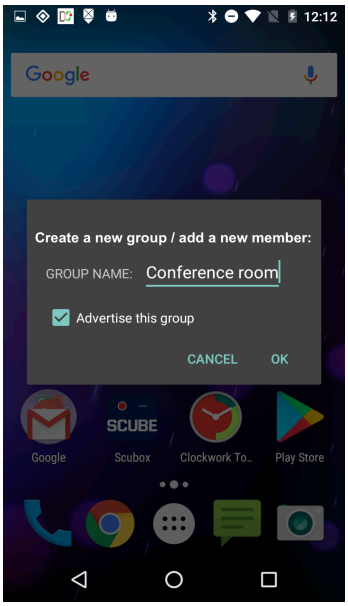


Figure 5.2: *LIQDROID provided alert dialogue for group formation upon receiving connection acceptance.*

- Categorizing the devices that the user has decided to connect to. Then LIQDROID will use this categorization to give the user the possibility to perform the distribution of the task on the set of devices that he prefers but not all of the connected devices. This will improve the performance of LIQDROID by decreasing the required time to send the distributed task execution requirements and receive the responses that we will explain more in the section 5.5.
- The possibility to remove a device involved in an ongoing interaction or remove it if he does not prefer to be connected with that device anymore. In this case, he will be able to reach the members of the group that the device belongs to, and after applying a long click on the device's name, a dialogue will be opened by LIQDROID which lets him remove that device. Then he will become disconnected from those devices although they are still in his proximity. However, if the user does not create any group, the devices will only become disconnected when they are out of the proximity range of each other. The proximity range of the devices depends on the connection protocol that the devices use to become connected (in the current version of LIQDROID, this is

[1] http://blog.p2pkit.io/how-google-nearby-really-works-and-what-else-it-does

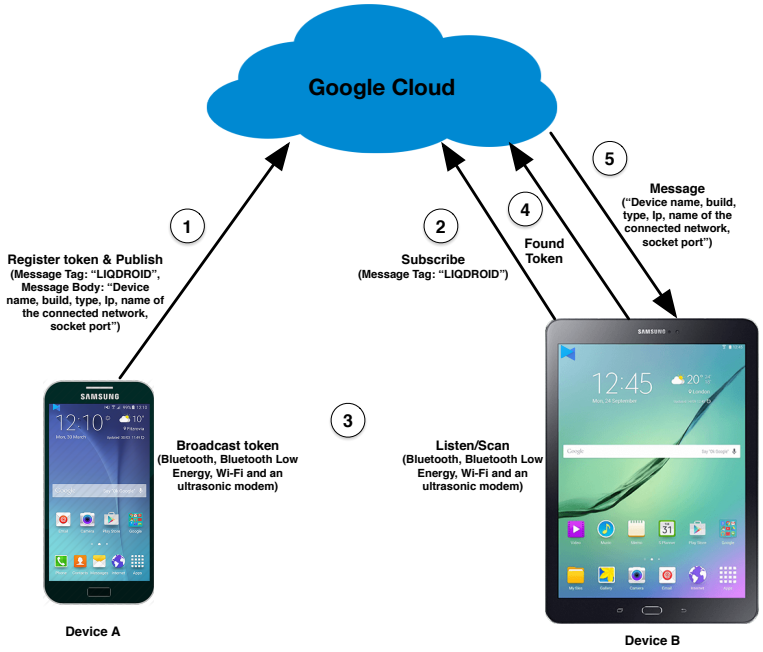


Figure 5.3: *Advertising and Discovering proximal devices through Google Nearby Messages* [1]

the Wi-Fi protocol).

- After creating a group the user is able to decide whether he prefers to advertise this group to the other members or not. If he decides to advertise the group, a new message will be created that includes the group's name and its members along with other information about the device (as mentioned above).Instead of the previous message, this new message will be advertised through Google Nearby.

  When the other device receives a message that includes the group name, it will add this group to the list of groups that are available on the device. To differentiate these two categories of groups, the groups that belong to that device will appear as the group's name along with the keyword "My Group." This list will be used as a filtering mechanism and will help the user to only recognize the groups that he has already created along with their members. He can also see the other groups and members that are available to connect. In the case of distributing the task, only the user-created groups will be shown to him so that he can select from among them.

Whenever a new user joins or the next time that the existing users perform the discovery, they are able to select a group and find the devices that they prefer to connect to among its members while there still exists the possibility to select the devices to connect to through the list of all the available proximal devices.

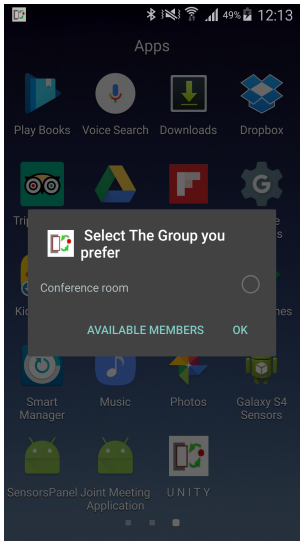


Figure 5.4: *Finding the desired devices through available groups' members or list of all the available proximal devices.*

5.3 Communication Channel

Based on the above mentioned limitation that exists regarding the size of the message that can be transferred by Google Nearby, we are obliged to use another connection mechanism to handle the exchange of larger sized data between the connected devices in case of need. In the recent version of LIQDROID, we have used the local Wi-Fi network to connect the devices together, which will enhance the speed of transferring data (specifically larger amounts) between the devices. However, it is possible to add and use other connection protocols in the future for transferring the data based on the availability of the connection protocols on the devices.

The Wi-Fi Network Name, IP address and socket port number that are included in the discovery message will be used to connect the devices through the Wi-Fi network. If several networks are available and devices are on two different networks, LIQDROID will notify the user about the

destination devices that can be used to connect to the other possible Wi-Fi networks so that the devices will be able to start the communication. The source device which has initialized the discovery will create several client sockets and connect to each device using a separate client-server channel. We assume that each device has already created a server socket and is ready to accept the client requests for connection.

Although the devices are connected and are able to both send and receive the data through the channel, the communication related to executing the tasks will happen in one way. This means that only the source device which has sent the connection request is able to ask to distribute a task on the destination device(s), and the destination device(s) is only able to answer the requests sent by the source device through the channel. If the destination device also wants to distribute a task to the source device, it should first send a connection request. This one-way communication to request to distribute a task will have two benefits. First, it will prevent the destination device(s) from executing tasks on the source device unless it has acquired its user's permission. Second, it will prevent the concurrency that may happen on the execution of the tasks on the connected devices, because the source device has a higher priority to execute a task and the destination device is able to execute a task on the source device if the user gives the permission. However, the user is able to reject the connection if he still wants to benefit from all the resources available on the destination device or if he thinks that the connection will cause an interruption in the execution of the current distributed task by him.

After the devices become connected, for sending the user's task distribution request (the intent), we have used the Android Message API. We have created an abstraction for the message that is being exchanged between the proximal devices; this class is called "unitMessage". The LIQDROID will use this class to perform the serialization and deserialization mechanism on the message, more precisely to the intent and its fields which may contain different types of the data, to be able to transfer it through the Wi-Fi channel and recreate it on the other side (destination devices).

The figure 5.5 shows the procedure of transferring the "unitMessage" to distribute the user-defined task on the destination device(s). The user provides a task execution request through a LIQDROID-compatible component(1); LIQDROID receives this request as an intent delivered by the Android framework (2). Then, LIQDROID will put this request as a data object along with all the other data objects that it may require on the destination device(s) to handle the task execution, inside a new "unitMessage", serialize it (3) and send it to the destination device(s) (4). When LIQDROID on the

destination device (device 2) receives this message through the network channel it will deserialize the message, store the data part that it includes in the storage, and recreate the intent. The new address of the stored data (which belongs to this destination device) will be replaced inside the recreated intent instead of the previous address (which belongs to the source device), and the intent will become ready to be executed. This will provide all the requirements to execute the distributed task on the destination device. The data objects which are to be transferred along with the intent and the execution of the intent on the new device will be handled by the Intent Manager module that we will explain in the section 5.5.

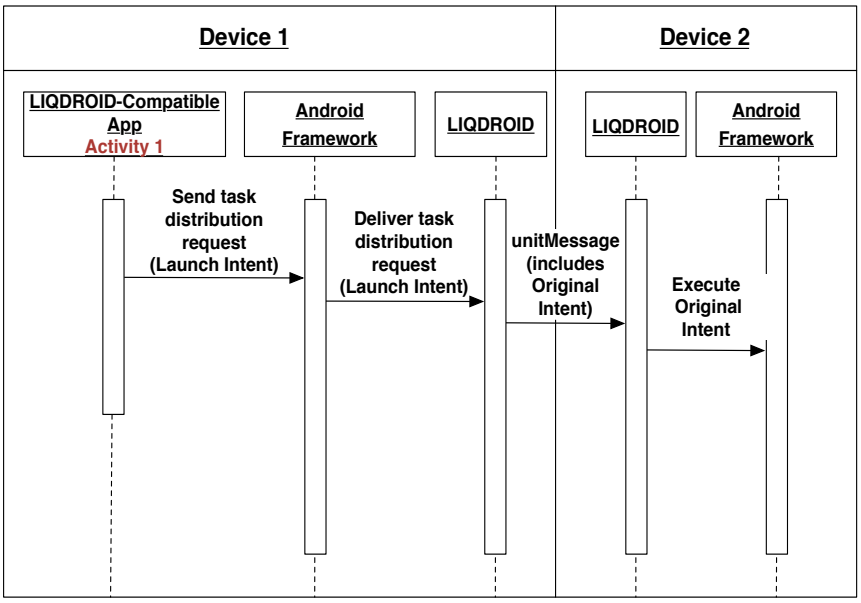


Figure 5.5: *Transferring the execution request(intent and its required data) through the unitMessage.*

5.4 Device abstraction

There are two different phases that LIQDROID may need in order to do the discovery for the user for the proximal devices that we explain them here separately.

5.4.1 Device Level Discovery

This phase of discovery happens when the user wants to connect to the proximal devices to distribute a task. In this phase LIQDROID will use

the physical characteristic of the devices to give an overview about the availability of the devices in his proximity, such as smartphone, tablet, etc., which will also remind the user about their physical characteristics such as the screen size or their functional capabilities.

5.4.2 Application (Components) Level Discovery

This phase of discovery happens when the user wants to select among the connected devices to find the best match or his desired device to distribute the task. On this phase LIQDROID will benefit from the features and capabilities that the device is supporting such as by filtering those devices that have a camera and can take photos. The mechanism that LIQDROID uses is beyond the physical device, but it will search among the components of the installed applications on the proximal devices to find those components that are able to perform the user's desired task. The idea of this mechanism has been adopted from the Android framework. So there is not any relation between the whole device and the requirements of the task that is going to be distributed except the features and the capabilities of that device. This will allow the user to not be restricted to a single device but to benefit from several devices in combination to perform different parts of a single task.

5.5 Intent Manager

The communication between the installed applications' components and LIQDROID happens through exchanging intents. As we have explained in the section 3, in order to call other components in the Android, the developer needs to create an intent and execute it. So in order to send the request for distributing the execution of a task, the developer needs to put the request inside the intent which he will use to call LIQDROID-service. The structure of the task execution request is in the form of an intent. This means that the developer should provide an intent in the same way that he creates when he wants to call other components inside the same device. However, instead of executing it through the Android OS, he will give it to LIQDROID to execute it. So, the developer does not need to apply major changes to benefit from the features which are available in LIQDROID to distribute the task; he just needs to provide additional data objects in the form of extra fields to this intent for LIQDROID. The developer should also remove the part which is in charge of starting the intent, as LIQDROID will handle this part properly (by considering the provided extra fields) on the destination device on behalf of the source component. The provided data in the extra fields will make LIQDROID capable of distributing the execution

of the task. These LIQDROID-defined extra fields include the name of the feature that the developer needs from LIQDROID to perform along with the extra data which is needed by LIQDROID such as the requester's component name, its package name, etc. As we mentioned earlier, different components in parallel can send requests to LIQDROID. LIQDROID will use this extra data as an ID of the requester to deliver the services to it and keep the communication between the source and the destination components consistent for the ongoing interactions such as sending updates, feedback or the task results. In fact, it will be a mechanism to be sure that the component which is receiving the responses is the one who asked for it.

In the following, you can find a sample code snippet of providing an intent to execute an activity to make a call. From now on we will call the intent that the developer created to launch a component the **Original Intent** and the intent that he used to communicate with LIQDROID-service the **Launch Intent**. The user is using the Sample LIQDROID App, which is a LIQDROID-compatible application, and sends the "callIntent" to LIQDROID to find the list of the components on the proximal devices that are capable of performing the call. Along with this request, he also puts the extra information requested by LIQDROID such as the "SourceAppComponentName" and the "SourceAppPackageName". The name of the action which is defined by LIQDROID and has been filled by the developer will enable LIQDROID to know which feature has been requested by the developer.

```
//The Original Intent for launching the contact activity to dial the
    mentioned number.
1- Intent callIntent = new Intent(Intent.ACTION_DIAL);
2- callIntent.setData(Uri.parse("tel:" + 391234567));

//The Intent which will be used to communicate with \textsc{LiqDroid}-
    service and to deliver the Original Intent
3- Intent startLiqDroid = new Intent();
4- startLiqDroid.setComponent(new ComponentName(("com.android.imani.
    liqdroid", " com.android.imani.liqdroid.MyService"));
5- startLiqDroid.putExtra("originalIntent", callIntent);
6- startLiqDroid.putExtra("SourceAppComponentName", "com.example.
    sampleLiqDroidApp.MainActivity");
7- startLiqDroid.putExtra("SourceAppPackageName", "com.example.
    sampleLiqDroidApp");
8- startLiqDroid.setAction("ChooserList");
9- startService(startLiqDroid);
```

We will discuss these LIQDROID-defined features defined in the form of intent-actions in the upcoming sections. Here the ChooserList means that the developer, by using LIQDROID, enables the user to select one among the available capable components on the proximal devices and make a phone call by using that component on that device.

The Action that has been defined in the *Original Intent* will be used by LIQDROID to find the list of capable components on the proximal devices. After receiving this list from all the devices that are connected to the source device, LIQDROID will provide a list of the available components on the proximal devices. This list includes the name of the device followed by the component name on it that is available to perform the user's desired task along with an icon of the application on which the component belongs. At the end of this list, LIQDROID also adds the list of the capable components that are available on the source device. Because it may happen that, although the possibility to execute the task exists, the user, based on a certain situation, prefers to use his own device to follow the task, these components will appear on the list by the keyword "My device" followed by the name of the component. The name of the component has been extracted from the application package that it includes; therefore, the user can better decide on the component of the application he prefers to use. As you see the *Original Intent* should not have the execution method because the developer will give this intent to LIQDROID to execute it. And the *Original Intent* should be executed on the device that the user prefers to distribute the task execution to. After the user has selected his preferred device from the chooser list, LIQDROID will send the *Original Intent* along with the data that it needs to the destination device using the unitMessage.

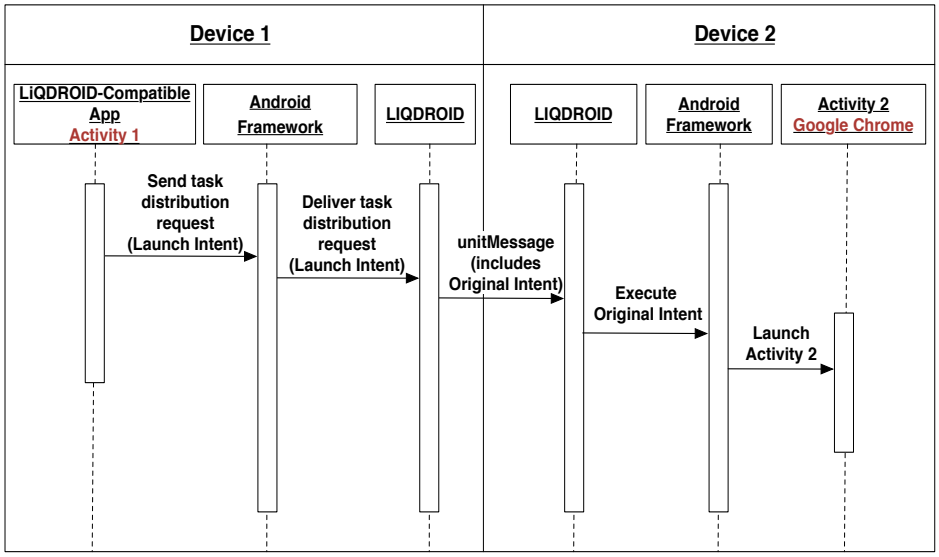


Figure 5.6: *Sequence diagram of distributing the task's execution through the Intent Manager Module*

Generally, the Intent Manager will interpret the intents at two phases; first, when it receives the intents through a Launch Intent, and the second when the intent has been received through the unitMessages. As you can see in the figure 5.6, the user sends the *Original Intent* through LIQDROID-compatible Activity 1 to LIQDROID (1). As soon as LIQDROID receives the *Launch Intent*, (2) has extracted the *Original Intent*, and the Intent Manager module is in charge of interpreting it and performing the task distribution by considering the desired features (3). Then LIQDROID will send this *Original Intent* through a unitMessage to the destination device (4). The Intent Manager module in LIQDROID on the destination device (device 2) is in charge of interpreting the *Original Intent*, providing all the requirements including the data, and executing it on behalf of the Activity 1 (5). The Android OS captures this intent and launches the Activity 2.

In the following sections, we will define the features which are proposed by LIQDROID and the extra fields required by it, and we also provide a short and simple document that will better guide the developers with samples for creating LIQDROID-compatible intents to request LIQDROID's available features.

5.5.1 Task Execution Manager

To provide a seamless interaction between the source and destination devices, to control the execution of the distributed task through receiving the user's inputs and preferences and applying them, LIQDROID needs to know in advance which type of interaction is going to happen between the devices and what the requirements are to handle this type of seamless interaction. We have provided a general overview of all these LIQDROID-provided features to support the seamless interactions between the devices in the table 5.1 while for the more technical and comprehensive explanations you can refer to the relevant section.

To better understand the necessity of these features, considering this example that if the user prefers to resume a task on the other device, instead of simply launching a new activity, we will need to transfer the current state and resume the activity on the destination device from that state instead of LIQDROID just launching it. This is a simple situation, but other more complicated interaction types exist where the user needs to control the execution of a distributed task through using the source device without reaching the destination device. This means that LIQDROID is in charge of distributing the input or the user's command is applied on the source device to the destination devices. For this purpose, LIQDROID represents a new

field inside the *Launch Intent*, which is called *"Device Role"*, to define the role of each connected device involved in an interaction.

Name of Feature	Category	Description	App To LIQDROID	LIQDROID To App	Module
Launch Action	Distribute a task (Launch a component)	• Is used for requesting to Launch a component on the destination device(s)	✔ On the source device	✔ On the destination device(s) • LIQDROID will execute the Original intent on behalf of the source component. • The target app can be the already existing apps in the Android market or a LIQDROID-Compatible app.	Task Execution Manager
Device Role	Managing devices' integration	• Is used for distributing the tasks of the categories Complementarity and Synched Devices.	✔ On the source device	–	Task Execution Manager
Bundle	State Synchronization	• Is used for distributing the tasks of category Device Shifting • The data which is required to resume a component on the deestination device(s) from the state that it has on the source device.	✔ On the source device	✔ On the destination device(s) • LIQDROID will execute the Original intent including the Bundle on behalf of source component • The target app should be LIQDROID-Compatible.	Task Execution Manager
Update Action	State Synchronization	• Is used for sending regular updates to the device(s) regarding to the state changes that happen on the running source component.	✔ On the source device	✔ On the destination device(s) • LIQDROID will broadcast the Update intent in the system • The target app should be LIQDROID-Compatible.	Task Execution Manager
Feedback Action	State Synchronization	• Is used on the destination device(s) to provide responses to the received updates in case of need.	✔ On the destination device(s)	✔ On the source device(s) • LIQDROID will broadcast the Feedback intent in the system • The target app should be LIQDROID-Compatible.	Task Execution Manager
Termination Action	Managing devices' integration	• Is used to kill the activity or services on the destination device(s) that the user is not interested in using it (them) any more. • Will make the destination device(s) free and will release their resources. • Will remove the activity from the Activity back stack of Android.	✔ On the source device	✔ On the destination device(s) • LIQDROID will broadcast the Termination intent in the system. • The target app should be LIQDROID-Compatible.	Task Execution Manager

Table 5.1: *Provided features by LIQDROID for managing the seamless interactions between the devices through the Task Execution Manager*

This extra field will enable the source device to acquire the role of the controller of the destination devices and manage them properly to collaborate with each other to achieve the user's desired goal. For example, consider that you have a video player activity running on your smartphone consisting of a panel that shows the video, a panel to control the video, and a panel for the set of existing reviews. You transfer the panel of video view to the big Android TV and the panel of the review to your tablet. Then, you use your smartphone to control the video or scroll down/up the reviews. In this scenario, the source device is playing the role of the controller of the two other devices, and LIQDROID needs to forward its provided commands to the right destination device.

There are two different types of roles in LIQDROID:

- Controller : which will be defined by the developer,
- Client: which will be assigned to the connected devices by LIQDROID which will also keep track of them.

LIQDROID handles the coordination between the devices through checking the "Device Role" field in the received Launch Intent. More precisely, the Device Role will be useful for the usage scenarios that go under the category of complementary or synched tasks.

5.5.2 Categories of devices' interactions

The synchronization between the devices not only happens during the interactions but also it considers the initialization and termination states, which means launching and terminating an activity or service through LIQDROID. Here we will further explain the requirements that need to be handled by LIQDROID to support the synchronization in each type of interaction between the devices:

1. **Device Shifting:**
   To handle this situation, LIQDROID checks if there is the possibility and interest in resuming a component on the other proximal devices by receiving the user's choice and interpreting it. After showing the list of the capable components to perform the task on the proximal devices, LIQDROID will receive the user's choice and handle it as follows:

   - Resume: If the user has selected the same component to be executed on the other proximal device(s), LIQDROID will transfer the current state of the component along with the intent to the other

device, and the component will be resumed with the same state it had on the source device.

- Other: If the user selects a different component, the state will not be required and transferred.

In device shifting, the data that we need to handle for the synchronization purpose is the current state of the component. The state which we are going to use to resume an activity on the other device(s) is the same bundle that the Android OS is using to keep the state of an activity when it goes into pause mode. To this end, if the developer wants to relaunch the activity with its last state, he needs to save this bundle. The LIQDROID will use this same bundle to resume the activity with the last state that it has on the source device while it is launching it on the destination device.

In order to improve LIQDROID's performance and to prevent unnecessary interactions with the source component, LIQDROID will ask the bundle from the source activity only when the user wants to resume the same activity on the destination device. By the same activity, we mean that the activity on the source device and the activity on the destination device(s) all belong to the same application package.

Although through the inter-process communication in one device we can send the bundle along with the intent to another component, this does not work properly when it is going to call a component on the other devices, so LIQDROID needs to apply another approach. Because the current mechanism that the Android framework is using to serialize the intent (Android Parcelable mechanism) does not support its bundle as well, LIQDROID needs to take care of serializing the bundle and sending it along with the *Original Intent* to the destination devices. It will deserialize it there and use it to resume the activity from the last state that it had on the source device.

2. **Complementarity:**
If there is an activity that will be executed in parts on different proximal devices and will be controlled by the source device, the activity needs to also set the device role field in the *Launch Intent*. Assigning a role, besides letting the "Controller" device control the other proximal devices, will give LIQDROID the possibility to manage the execution states of the assigned tasks on the destinations devices (Clients) or the ability to terminate them in case of need to release the resources. We will further explain the possibility to terminate a task in the section 5.7.

3. **Synched Devices:**
   To keep the state of the distributed tasks synced in all the devices which are running LIQDROID-launched components, LIQDROID needs to send intents on a regular basis as soon as a state change happens in the source component. State changes can be applied by considering different types of data received by LIQDROID on the destination devices. The LIQDROID will make it possible to support the exchange of various types of data (serialize/deserialize). The state changes may be required through the Wi-Fi channel, but the interpretation of these data and updating the component's state on the destination devices will be under the responsibility of the developer along with the frequency of sending the updates. For example, a user may need to open different pages of a book on several connected devices, and when he goes to the five next pages, the page number of all the connected devices will be shifted to the next five pages based on the current page on each device. Here, the data is as simple as a page number that should be transferred by LIQDROID to the destination devices, while it could be a more complex case including several types of data; for example, when the user wants to fill a form by entering the text and image on his device and at the same time update the state of the form also on the other connected devices.

5.5.3 Managing the devices during the interaction (State Synchronization)

LIQDROID enables the developer to be able to not only launch components on the destination devices but also send updates and receive feedback (acknowledgements) related to them regularly. It is capable of either perfectly supporting the usage scenarios that different components are receiving services from LIQDROID in parallel without interrupting each other or having access to the data of each other (prevents data leaking among the various components). In the following section, we will explain in detail these update and feedback actions defined by LIQDROID and how LIQDROID will manage their execution. To request different LIQDROID-supported features, the developer should create *Launch Intent* with the following action names along with the required data by LIQDROID for that action.

- **Launch Action:**
  This is the initial point for the process of distributing the execution of a task. So, whenever LIQDROID receives an intent with this action name, it will provide a list of the available components capable of performing

the action defined in the *Original Intent*, on the destination devices as well as the user's device at hand. If the user chooses his own device, the intent will only be executed there, but if he decides to (also) choose the other device(s), the intent will be sent to the other devices as well. In case of the scenario that the user wants to resume an activity on the other device, the developer needs to also put the bundle as an extra field in the *Launch Intent*.

To provide the list of the capable components (activities or services), LIQDROID follows the same procedure that Android OS already uses to provide the chooser list when it receives an implicit intent. The chooser list is the one that shows all the installed applications capable of performing the user's desired action. For example, when you want to open a URL, if you did not define the default browser yet, Android will show you the list of the installed browsers on your device and you can choose one to open the URL.

Here LIQDROID of the source device will extract the action name of the *Original Intent* and send it to LIQDROID of all the other connected devices. They will create a list of all the components which have the proper intent filter by using the *PackageManager* and the *ResolveInfo* classes inside the Android framework and return this list to LIQDROID on the source device. As soon as the source device has collected all the responses from the connected devices, it will put them in a list along with its own list of capable components (received from the Android OS in the same way) and show the final list to the user to choose the ones that he prefers.

Figure 5.7 shows how this procedure occurs inside two Android devices. The "Take Image Activity" executes a *Launch Intent* to start LIQDROID-service (1). Android OS catches this Launch Intent and delivers it to LIQDROID as it is the only component that has the proper intent filter (2). LIQDROID extracts the action name of the *Original Intent*, puts it inside a "unitMessage" and sends it to the other device through the network channel(3). On the destination device (device 2), LIQDROID's Intent Manager module will receive the request and send it to the task execution manager module (4). It creates the chooser list by resolving the intent's action against the available components (5).The chooser list, which includes the components "Activity2" and "Activity3", will be transferred back to the Intent Manager Module (6) and then through the "unitMessage" to LIQDROID on device 1 (7).

The Intent Manager Module on device 1 will receive the component list

of destination devices (8) and will find and add the component "Activity4" that is available on its own device (9) and will show the final list to the user and let him pick the one preferred. As we explained before, after the user has selected the preferred components(10), LIQDROID will handle the execution of the selected components on the desired devices(11).

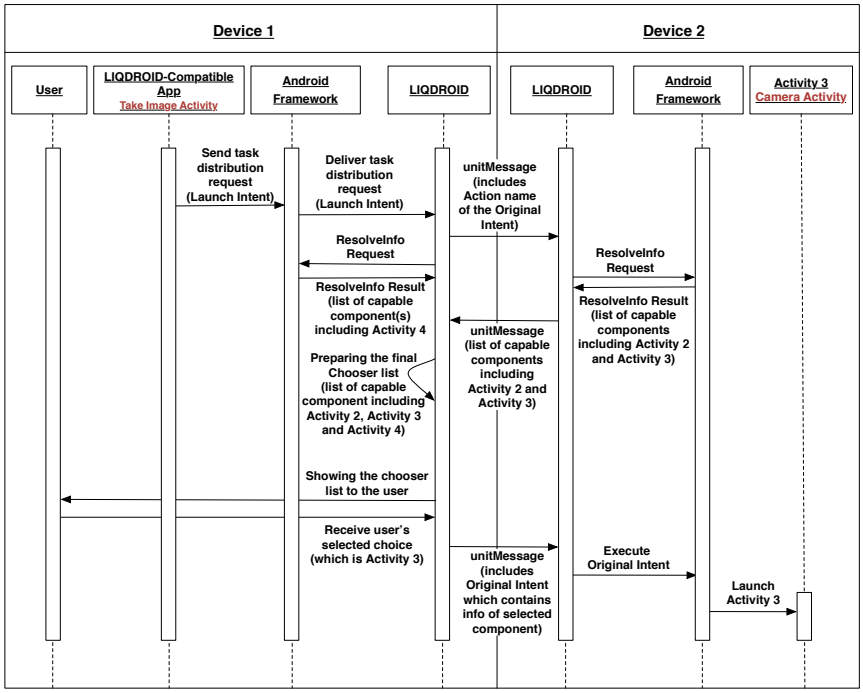


Figure 5.7: *Showing the Chooser list (of capable components) and launching the desired component(s) on the destination device(s)*

- **Update Action:**
  For those interactions that go under the categories of complementary or synched devices, it is more probable that the developer wants to send several updates to change the state of the components which are running on the destination devices. The Update intent can include the data related to the new received user's input or the state changes that happened during the execution of the component. As soon as LIQDROID receives the *Launch Intent* with the action name "Update," it will broadcast it inside the device, and the running component in charge (by comparing its package name and component name with those provided in the intent) can catch it. To overcome the privacy concerns

and prevent information leakage, it is better that developers define unique names for the fields inside this intent, encrypt the sensitive data, put it in the Update Intent and then send it through LIQDROID. So, in this way even if other components capture these Update intents, they are not able to steal the data.

As the interaction between LIQDROID and other components happens through exchanging the intents, this is also the case for updating the state of the other components. If the developer provides data inside the Update Intent which has a large size, there could be a delay in the execution of the intents in the destination devices and also in applying the updates on them. LIQDROID will use the same procedure as the Android framework to handle the execution of the intents, which is through considering a queue and processing them one by one. If while LIQDROID is sending an Update Intent, the user of the source device changes his mind and prefers to launch another component on that destination device, then LIQDROID receives a Launch Intent to launch a new component on that destination device, which has a higher priority than the Update Intent. However, if it is still busy with sending the Update Intents that it previously received with a large size of data, this new Launch intent will wait until its turn arrives.

In order to avoid this inconsistency when LIQDROID receives an Update Intent, it checks the data size. If the size of the data is higher than a certain threshold, it will transfer the data between the source device and the destination devices through the cloud infrastructure, which will highly decrease the time required to send updates with a large data size included. We will talk more about handling the data through the cloud infrastructure in the section 5.6. In this way, LIQDROID is able to better manage the execution of these queues of the intents and improve its performance.

- **Feedback Action:**
  In some use case scenarios, it is necessary to receive an acknowledgement or a response back in order to be able to continue the execution of the distributed task and go on to the next steps. In this way, the developer is able to provide milestones, control the task execution at runtime and be sure that the previous part of the task has been handled successfully by receiving the acknowledgements from the destination devices and then moving forward. For this purpose, LIQDROID has been considered an action name which is called "Feedback."

  As we currently mentioned, multi-device context is highly dynamic,

and devices may leave the context at any point in time, so acknowledgements are highly necessary for saving the time to know how much of the task has been done correctly before a specific event has occurred, such as the connected device dying or leaving the context.

The important point that we need to mention here is that, as we said, the communication between the devices for distributing the execution of a task is one-way. But this restriction is only concerned about distributing the execution of a task on the source device through the destination device, while the destination device is capable of benefiting from the Wi-Fi connection to send the Feedback intent to the source device without the need to ask for any permission in advance.

- **Termination Action:**
  As we are launching a component on the destination devices, we are using their resources, and this can cause some issues on the execution of the next tasks, such as decreasing the performance because there would be other executed tasks left in the background, or decreasing the battery, etc. To this end, LIQDROID has considered a mechanism to terminate the execution of the distributed task by finishing the running component on the destination device(s) to free up the resources.

  In general, LIQDROID as a third-party application is not able to kill any activity which does not belong to its own application package, so we need to call the kill method which has been implemented for this purpose by the developer inside that component to finish its execution. This is more important when the running component is an activity as it will acquire the screen of the device and will stay on the activity's back stack if a new activity is launched on the device. Therefore, in order to manage the execution of the components on the other devices, more precisely managing the activity back stack on the destination devices, LIQDROID reserved an action name which is called "Termination." The Termination intent will enable LIQDROID to manage the back stack of the Android framework existing on the destination device(s) and will remove the activity that has been launched by it there while the user is not interested in it anymore.

  To better understand the importance of this feature, assume that the user has selected device B to play a video. While it is playing the video, it receives a new launch activity intent and launches the new activity for taking a picture. As soon as the current task has been finished, the activity should be destroyed and the previous activity that was responsible for playing the video should come up to the foreground

again. So, to handle this scenario, we consider that when LIQDROID realizes that the taking picture task has been handled successfully (by receiving a Feedback intent on the source device), it will broadcast the terminate intent received by the source device and the camera activity will be killed on the destination device. This results in the previous activity being resumed, and the user can continue to watch the video without the need to reach that device to relaunch the activity. In the following section 5.7, we will further discuss this feature, especially considering cases in which different users may compete to distribute the execution of a task on the same destination device.

For sure, managing the execution of a service is much easier than managing that of an activity. The first reason is because several services at the same time can be running in the background, while there could only be one activity actively running on the device (if the activity goes to the background, all its ongoing tasks will be paused). The second reason is that the lifecycle of an activity is much more complex than the service, as we explained in the chapter 3. And this is the main reason why here we focus more on managing the execution of the activities, although in the section 5.8 we will also mention how LIQDROID will manage the execution of the services.

As we discussed more in detail here, handling the interaction between the devices and distributing the execution of the tasks is more complex than just finding the best capable device to launch a component there, which is provided by the service-based solutions currently available. In a way, it is not just a one-time communication to make a request to the connected device but the user can continuously send the required changes as an update to the other device(s), synchronize the state of all the connected devices, control their execution by receiving feedbacks, and be able to terminate an interaction.

5.6 Artifact Manager

LIQDROID can transfer the data between the connected devices through two different mechanisms: through the cloud infrastructure and directly through the local Wi-Fi network. By the data, we mean the files (images, audio, video, .txt files, etc.) and values which will be used as an input by a component.

5.6.1 Transferring data through the cloud infrastructure

As the devices may join or leave the context at any point in time, storing the final results of the distributed task can enable the user to have access to it on the move or later in time in other contexts with new sets of proximal devices.

Currently, we are using the Firebase for storing the data and loading it into the other device(s). It gives the user the possibility to load the results provided by the other components of the destination devices to be accessible even if those devices are not in the proximity of the source device or in general are not connected to the source device anymore.

The other way around, if the source device has been sharing content which is being used by the other users on the destination devices, even if the source device leaves the context, the other devices are able to continue the execution of the components until the task is finished, as well as being able to store the final results in such a way that they will be accessible to the source device from anywhere and at any time.

Another advantage of using this approach is the possibility to support the asynchronous execution of the consequence tasks. LIQDROID lets the user store the data (which was achieved from the execution of another part of a distributed task) and postpone the execution of the rest of the distributed task to the time that the proper device joins or is available in the context.

As the figure 5.8 shows, Firebase storage allows an application installed on a different set of Android devices to share their data with each other to have access to them to change and update them. As LIQDROID is a service which is running on each of the Android devices, the Firebase storage enables all LIQDROID instances installed on different devices to have access to a shared storage to upload or download the data of/for a component. In this way, even if there is not any storage available on the source device to store the results of a distributed task, the user is still able to distribute a task and have access to its result. This mechanism will also provide the possibility to directly move the data from the current destination device to the next destination devices selected by the user; so the data can easily flow between the devices without the source device being involved.

There are two types of data that are going to be transferred to the other device: structured and unstructured. Unstructured data includes various file types such as video, images, text files, etc., and Structured data is the values that will be kept in a database. For the unstructured data, LIQDROID will follow the current policies of the Android framework. This policy says that the files of an application will only be accessible through the components

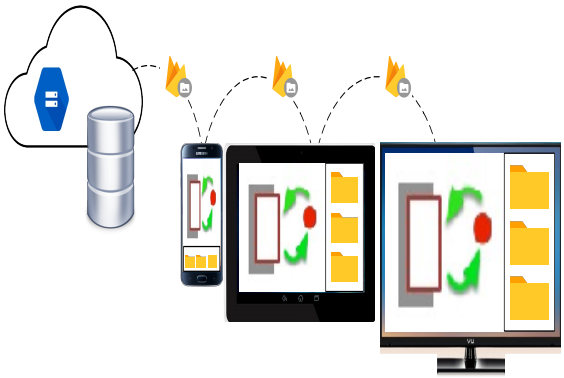


Figure 5.8: *LIQDROID will share artifacts between different devices through Firebase Storage.* [2]

which are in the application's package, unless it provides a content provider that lets the components of the other applications also have access to these files. LIQDROID will have a general content provider to share the data if the original owner of the file lets the other components have access to it; otherwise, it will keep the files private and will share them only upon the owner's request. The owner of the data will be the component on the source device that the task distribution has been requested through or the one that is controlling the execution of the distributed tasks on the connected destination device(s).

The mechanism of managing the results related to each distributed component through LIQDROID is as follows: The component on the source device should provide the extra information in the Original Intent such as the type of data and the defined name for the data that will be achieved as the result of the distributed task. LIQDROID will create a folder in the Firebase storage under the name of the source device followed by the package name of that component. Inside this folder, there will be a file under the name of that component which includes the achieved results. As we mentioned previously, the developer is obligated to provide the package name and the component name in each of the *Original Intent*. Consequently, in order to use the results of an already distributed task in another selected destination device, the developer just needs to add the file name. Then, LIQDROID on that selected destination device is in charge of downloading the data with the provided file name.

[2] `https://firebase.google.com/products/storage/`

A concurrency may happen if several destination devices attempt to store their achieved results through LIQDROID at the same time. To this end, LIQDROID will use a version control mechanism to support this concurrency as follows; in the first step when the developer makes a request for storing the data in this way, it is also in charge of providing the name of the file and is highly recommended to provide a unique name for each request that the user may make. In the second phase at runtime, LIQDROID will check the available files inside the storage that are assigned to that component; if the proposed file name already exists inside it, LIQDROID will incrementally add a number to that file name which shows the availability of different versions of that file. Whenever it receives a request from the user for a file that has the same name as the file name that already exists, it will show the list of the available versions of that file and will ask the user to choose the version that he prefers to download and work on. The most recent one will have the higher number.

To manage the storage of the files in the Firebase and to remove those files that are not required by the user anymore, after a predefined threshold, LIQDROID will check the availability of the components on the devices and the file will be removed when the original component is not available on the source device anymore (which means it has been uninstalled).

Beside the Structured data, LIQDROID is also able to support the storage of unstructured data. For the unstructured data, LIQDROID is capable of transferring the values and inserting them into the database on the source devices on behalf of the destination components (or vice versa). By using LIQDROID, the user has the possibility to have access to an ad-hoc database on a single device and will be able to insert data and make queries on its data through other proximal devices.

To implement this feature inside the components, the developer needs to make the request through creating the Original Intent by mentioning the name of the function (includes Insert and Query) and providing the values (through adding extra fields to the Original Intent). LIQDROID stores and retrieves the data by keeping the values in a text file, uploading it to the Firebase storage, downloading it in the source device that has the required database, and inserting the values. The mechanism for making a query on the source device and inserting the data into the database on the destination device is the same as the mechanism used to make a query and insert the data in a single Android device.

5.6.2 Transferring data Directly through the local Wi-Fi network

In general, using the cloud based infrastructure to store the results is the main data management mechanism of LIQDROID. If the user, for any reason, prefers to transfer the data without using the internet data or keeping it in the cloud, a direct data transfer mechanism can be used to send the data to the destination devices.

Although sending a file directly to the other device is more secure than storing it in the cloud, it would be time-consuming, especially when it is necessary send a large amount of data to several devices at the same time. So, it should only be used if there is a necessity. For sending the data directly, LIQDROID will combine it with the Original Intent and send it to the other device(s) through the "unitMessage."

As we already mentioned, one of the available policies of the Android framework is that a component is not able to send a large file directly through an intent to the other component. Alternatively, for sending the data, the intent will only contain the reference address of the content (URI), and the content provider is responsible for providing that content for that component. To extend this feature among involved devices, as it is not applicable to use this reference address on the destination device(s), LIQDROID will take care of transferring the required data through one of the above-mentioned ways and will store the data in a folder that is only accessible by LIQDROID. Then, it will execute the intent by using its new URI on the destination device(s). If the original component prefers to let other components have access to this data, it will be accessible by the other components through LIQDROID's general content provider.

5.7 Event Manager

We have provided a general overview of how LIQDROID is handling different categories of events that may happen in a device or during the seamless interactions between the devices in the table 5.2 while for the more technical and comprehensive explanations you can refer to the relevant section.

5.7.1 Device Level Events

Inside the Android framework, there is a general broadcast receiver that notifies the user about the different system level events happening inside a device to perform proper actions in case of need. For example, if the battery status goes under a certain threshold (generally this is defined as under 20%), it will notify the user to connect the device to the power source.

In order to provide this mechanism while we are proposing the bigger Android ecosystem concept, these events should be shared between the connected devices involved in an interaction to be able to better control the situation for the user and prevent the unwanted interruptions. For this purpose, LIQDROID has a general broadcast receiver that captures a subset of the system events that happen in a device, and it uses them while making decisions or lets the developer have access to them. This subset of the considered events in LIQDROID are events related to the battery status, storage status, screen goes off and on, and the wireless state.

Sharing these events between the connected devices will empower LIQDROID to make better decisions while they can interrupt the normal execution of the distributed tasks. LIQDROID benefits from the events related to the battery status of the devices to organize the list of the capable components. To this end, LIQDROID will remove those devices from this list while it will also notify the user if he already is interacting with them.

LIQDROID also lets the developer define his own event class and transfer it to the other devices. This feature will be more useful for transferring the input events such as touch events. In this case, the developer is able to define his own class to capture the input events, transfer it through LIQDROID on the destination device, receive it, and apply the event on the destination device. Take, for example, there is a PDF reader application installed on the source device and also on the destination devices and you are in the classroom and you open a PDF file. Of course you want to teach on your tablet and also the students' tablets. You scroll up and go to the next 4 pages to show a figure; at this time, this scroll event on your device will be captured and transferred to the other devices and will be applied there as well. So, the students will see the page that you are on currently without needing to perform the scroll up action on their tablets.

5.7.2 Interaction Level Events

This category of events is more important when the distributed task is about executing a component that is an activity because at each point in time it is only possible to have one activity in the foreground on a device. As we mentioned already, it may happen that while an activity is using LIQDROID, another activity sends a new request to it or the foreground activity that has been launched by LIQDROID changes (a higher priority event happens such as receiving a call or the user launches a new activity). LIQDROID will benefit from the task stack that Android is using to keep the order of the launched activities that took the user interface's focus in order to manage

these state changes.

Name of Feature	Category	Description	App To LIQDROID	LIQDROID To User	LIQDROID To App
System events	Device Level Events	• Devices are able to register to receive certain system events that happen on the other connected devices.	✔ On the source device	–	✔ On the destination device(s) • LIQDROID will use these events to manage the devices interactions. • LIQDROID will broadcast the event inside the device(s) • The target apps should be LIQDROID-Compatible to be able to receive the events.
Components events	Device Level Events	• The developer can define his own event class such as input (touch events) and transfer it to the other devices.	✔ On the source device	–	✔ On the destination device(s) • LIQDROID will broadcast the event inside the device(s). • The target app should be LIQDROID-Compatible to be able to receive the events.
Activity in the Launch State (onStart)	Interaction Level Events	• Lets LIQDROID to be informed about the status of the running component(s) on the destination device(s).	✔ On the destination device(s)	✔ On the source device • LIQDROID will notify the user if the currently selected destination device is busy to give him the possibility to choose another device • LIQDROID will prioritize the Chooser list accordingly	–
Activityon the Pause State (onPause)	Interaction Level Events	• Lets LIQDROID to be informed about the status of the running component(s) on the source and destination device(s).	✔ On the source device	✔ On the source device • LIQDROID will receive the user's preferences about how to proceed with the currently distributed tasks on the destination device(s) including following choices: ▷ Terminating the distributed task ▷ Putting it on pause ▷ Optional methods ▷ Accept (no changes will be aplied)	✔ On the destination device(s) • LIQDROID will behave according the user's choice. • The target app should be LIQDROID-Compatible to be able to terminate it or apply the optional method. • Putting a component on the pause state can be applied both to the existing apps in the Android market or LIQDROID-Compatible apps.
			✔ On the destination device(s)	✔ On the source device • LIQDROID will teceive the user's preferences about how to proceed with the currently paused tasks on the destination devices that could be: ▷ Terminaing the distributede task ▷ Discarding the notification	✔ On the destination device(s) • LIQDROID will behave according the user's choice. • The target app should be LIQDROID-Compatible to be able to terminate it by LIQDROID.
Activity on the Resume State (onResume)	Interaction Level Events	• Lets LIQDROID to be informed about the status of the running component(s) on the source and destination device(s).	✔ On the source device	–	✔ On the destination device(s) • LIQDROID will behave according the user's provided choice on the pause state which could be: ▷ Will resume the paused task ▷ Or will apply the developer defined method to resume the task.
			✔ On the destination device(s)	✔ On the source device • LIQDROID will notify the user that the distributed task has been resumed.	–
Activity on the Destroy State (onDestroy)	Interaction Level Events	• Lets LIQDROID to be informed about the status of the running component(s) on the source and destination device(s).	✔ On the source device	–	✔ On the destination device(s) • LIQDROID will broadcast the Termination intent in the system. • The target app should be LIQDROID-Compatible.
			✔ On the destination device(s)	✔ On the source device • LIQDROID will notify the user that the distributed task has been terminated.	–

Table 5.2: *Supporting the occurrences of different events in the devices and also during the seamless interactions between the devices through the Event Manager Module of LIQDROID*

As we do not have full access to this stack to extract the information we require, to handle these concurrent events that may happen in a device, LIQDROID needs to be directly informed when the activity's lifecycle has been changed. For this purpose, the activity needs to inform LIQDROID by sending an intent while the following state changes happen: "Start", "Pause", "Resume" and "Destroy."

In order to separate the normal execution of an activity when the user has directly launched it from the instances that the execution of the activity is under the control of LIQDROID and LIQDROID has launched it, the developer needs to set a flag inside the activity. In this way, the activity's lifecycle changes will only be reported to LIQDROID when the underlying activity has been launched by LIQDROID, so it will be able to manage the interaction between the devices and control the execution of the distributed tasks.

In this Android ecosystem, several Android devices are involved, and each of them has its own task stack that LIQDROID should also keep a close watch on to manage the activities that have been launched on them. The importance of managing the task stack of the other devices is more detectable when two different users compete to distribute two activities on the same device. This is important to be managed; to clarify, consider the example of an Android big screen in a museum that many users in a day will visit. These visitors may use the screen to extract different information such as the location of the art pieces, and they also may use it to view images or videos of the art pieces, etc., and each of these tasks may launch a different activity on the big Android screen. So at the end of the day, there would be several launched activities on the big screen that have occupied its memory and will decrease its performance. Nevertheless, by using the event handling mechanism implemented in LIQDROID, which we will discuss in more detail in the following, the unwanted activities will be removed from the task stack of the big screen whenever the new activities are going to be launched.

One of the main benefits of defining the "Device Role" is being able to handle these concurrent events that would happen in each of the participated devices. In this case, as soon as LIQDROID receives a Launch Intent, it will change the current value of the "Device Role" field on the destination device(s) and set it to "Client". It will then notify the other user to select another device, and if the user selects the same device again, LIQDROID will launch the new activity and notify the "Controller" of the previous activity that his activity went to the "onPause" state. If it does not receive any response in a predefined threshold, it will search for that device in the list of the connected devices, and if the device does not appear in that list,

this means that the device has left the context or is not in the proximity of the big screen anymore, so LIQDROID will call the "onDestroy". If the developer provided the proper method to be called by LIQDROID to finish that activity, LIQDROID will be able to finish the execution of the activity, so the activity will be removed from the task stack of the Android big screen. This proposed scenario is one of the existing potentials that may cause conflicts in the execution of the distributed task inside the proposed Android ecosystem. In the following, we will explain other possible conflicts that may happen during the execution of a distributed task on the proximal devices and how LIQDROID will manage them.

- **Activity in the Launch State:**
  In the Figure 5.9, we have defined the different states that may happen to a Launch Intent based on the user preferences at the runtime. In the first part that the LIQDROID-compatible application is not involved, the concern lies in discovering the proximal Android devices and connecting to them as we explained in the sections 5.1 and 5.3. After the user has connected to the devices that he preferred, he is able to distribute the task on them. So, the user launches the Video Controller application, which is a LIQDROID-compatible application, to play a video on one of the proximal devices while he wants to control it through his cell phone. In the first step, the user clicks on a button inside the Video Controller's activity, which sends a Launch Intent to LIQDROID. LIQDROID receives the intent and provides the list of available components on the proximal devices that are able to play the video. The user selects a component that he prefers.

  We need to mention that here the user has selected again a LIQDROID-compatible component that belongs to the same application package that the component on the source device also belongs to; in this way the source component has more access to manage the execution of the destination components as it knows their structure. This will enable the user in the next step to push the play button that is proposed by the Video Controller activity already running on his device and make the video player component that has launched on the destination device start playing the video. This procedure will be handled by sending an Update Intent that the developer has defined under the play button to the launched component on the destination device through using LIQDROID. As the role of the Video Controller activity has been defined as the Controller on the source device by the developer, the video control options such as play, pause and stop will be sent to the

destination device(s) through the Update Intent by using LIQDROID.

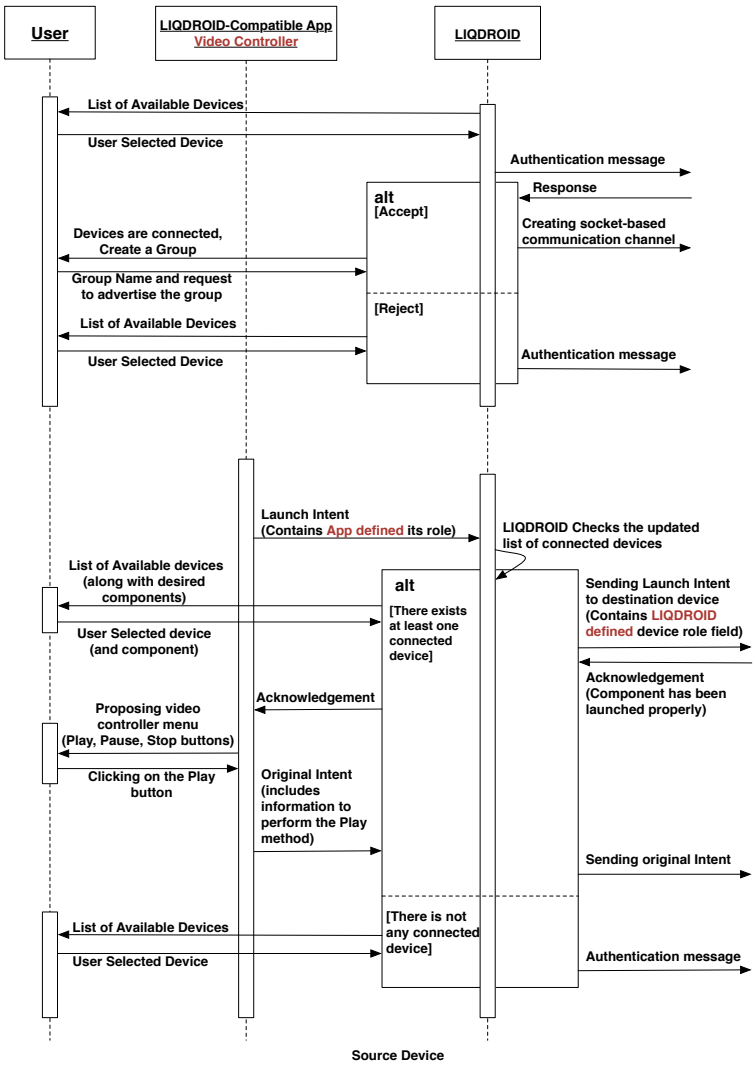


Figure 5.9: *Sequence Diagram of launching a component on the destination device while the source device's role defined as Controller.*

This is a normal behaviour of launching a component on the destination device when the destination device is not currently busy due to its own user or the other users which are able to interact with it through LIQDROID to distribute a task. If the destination device is busy, LIQDROID will ask the user of the source device to choose another device to distribute the task, but if the user still prefers that destination device,

LIQDROID will launch the component on the destination device. In the next item, we will explain what will happen to the previous running activity on the destination device that goes to the Pause state.

We need to mention also that the interaction between the user and LIQDROID happens through a pop-up dialog on top of the running components on the device. The pop-up will be removed automatically if the user does not provide any choice in a predefined threshold. This will be more useful when a device is not accessible by a user, or when the device is busy and cannot interact with the device to provide any choice.

- **Activity on the Pause State:**
  When the activity goes to the background and loses the focus, the Android system calls the onPause method on the activity, which stops all its ongoing actions. As soon as LIQDROID receives an intent which notifies this state, it will check the "Device role" field available in LIQDROID. If the current device has the "Controller" role, LIQDROID will show a pop-up to its user to ask for his preferences. This pop-up gives different possibilities to the user to manage the possible situations, which can be as follows:

  – Termination: terminate the activities that have been launched by LIQDROID on the destination devices (Clients).

  – Pause: LIQDROID will also put on the pause state the activity (or activities) which has been launched on the destination devices (Clients) by LIQDROID. For this purpose, as LIQDROID is not able to call the onPause method of the third party's activities, it will launch a simple activity which only includes a line of text notifying the Client device's user about the current status of the device. This simple activity will force the previous LIQDROID-launched activity to go to the background, and its onPause method will be automatically called by the Android framework.

  – Optional methods: The developer is also able to provide a method which is available in the activity that he prefers to be called by LIQDROID on the destination devices instead of putting the activities in the pause state; for example, these methods can be: decrease the voice or to pause the video which is playing on the other device. The difference between applying a method and putting an activity on the pause state is that in the first case the activity will remain in the foreground and has the focus, but in the second case putting

the activity on the pause state will cause that activity to go to the background.

– Accept: Which means that the user did not provide any input and no change will be applied. For example, if the video is playing on the client device, it will continue to play without any interruption.

Figure 5.10 shows the order of the actions that LIQDROID takes as soon as the launched activity goes to the pause state.

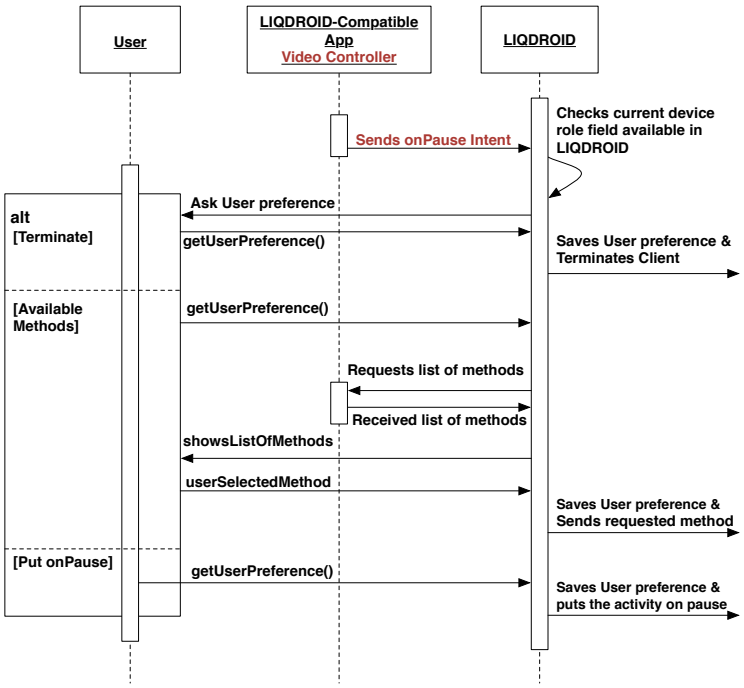


Figure 5.10: *Sequence Diagram describing when the activity on the source device goes to the pause state while the device has defined its role as Controller.*

On the other hand, if an activity running on the "Client" goes to the pause state, LIQDROID will notify its "Controller" device, and he can terminate the activity if he prefers or discard the notification. This situation mostly happens when the destination device is currently busy running an activity and LIQDROID receives a new launch request for a new activity on that destination device. Figure 5.11 shows the order of the actions that LIQDROID takes as soon as the launched activity on the destination device, which has the role Client, goes to the pause state.

- **Activity on the Resume State:**

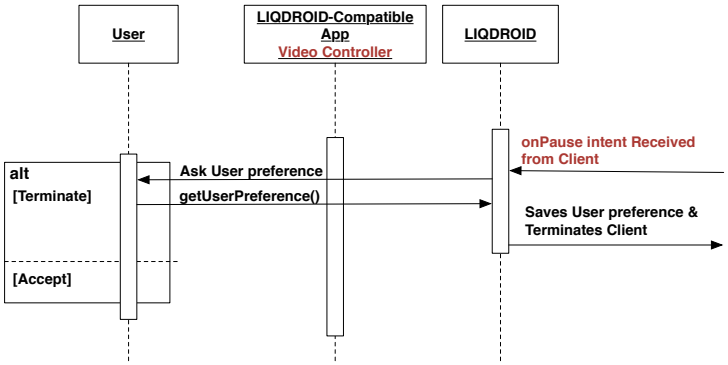


Figure 5.11: *Sequence Diagram describing an example of when an activity goes to the pause state on the Client device.*

When the user resumes an activity and it comes into the foreground, the onResume method is called by the Android system. If LIQDROID receives the onResume intent, based on what the user has preferred in the pause state and what has been saved by LIQDROID, it will make the proper decisions as follows:

- For the case that the source device has the role of "Controller": if the developer has also considered a method to be applied on the resume state, LIQDROID will send it to the destination device that has the role of client and it will be executed there. Otherwise, the activity will be resumed by LIQDROID. To resume the activity that LIQDROID has put in the pause state, it will simply finish the activity's execution that has been launched previously to make the activity go to the onPause state.
- In the case that the destination device has the role of "Client": LIQDROID will notify the user of its source device which has the role of "Controller."

The Figure 5.12 shows these two different possibilities.

- **Activity on the Destroy State:**
  Will be called when an activity is completely covered by another activity and is no longer visible to the user, so its window is hidden. This activity will be killed by the system when it wants more space and wants to release the memory or when the user manually closes the activity. Destroying an activity will cause it to be removed from the task stack of the Android operating system. When LIQDROID receives

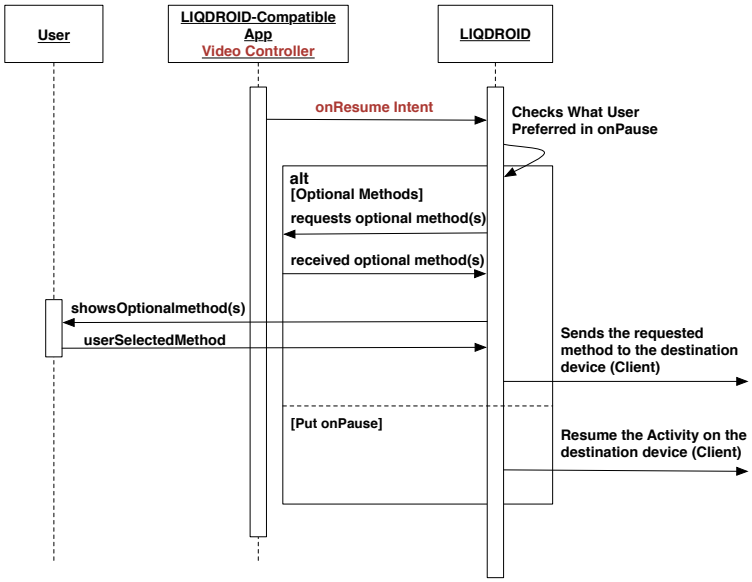


Figure 5.12: *Sequence Diagram describing when an activity has launched by* LIQDROID *is resumed.*

an intent which notified that the onDestroy method of the activity has been called if the activity belongs to the:

- Source device that has the role of Controller: this means that the user is not interested in executing the distributed tasks anymore, so it will call the termination methods implemented by the developers on the activities that have been launched on the destination devices having the role Clients.
- Destination device that has the role of Client: LIQDROID will notify its Controller device.

5.8 Service Manager

The current binding mechanism that exists in the Android framework is for the services and the activities which belong to the same device, so we are not able to use it while the activity and the service are placed into two different devices. For this purpose, LIQDROID has provided its own mechanism to handle this situation. So, this module will enable the activity that exists on the source device to start a service that is on the destination device to perform its desired task, and if there are any results, it receives them back.

For this purpose, the user first needs to have a list of the connected proximal devices that have the user desired service already installed. Second, LIQDROID needs to enable the activity and the service to communicate with each other by sending messages forward and backward. In the end, the service should be stopped when the LIQDROID-bound activity has been closed. This last step is very important because the service is a component that is running in the background, so the user is not able to easily close it except by accessing it through the settings menu of the device.

In the same way that the user is able to send a request to get the list of the capable activities on the proximal devices, he is also able to ask for the list of services and use the Update and Feedback Intents to interact with the launched service on the destination device(s). One of the main advantages of having the service manager is that, as the service does not have any interface, it is possible that several users can use the same device for running different services in parallel unless these services do not use the same hardware resource (microphone, camera, etc.) on the device.

In order to let the user know that a service is going to be started on his device, LIQDROID will provide a permission pop-up dialog before it starts the service on the user's device (users of the destination devices). If the user gives the permission, the service will be started. Otherwise, LIQDROID on the source device will receive the reject response and notify its user, and then it will show him the list of the other proximal devices that have the service in order to let the user choose another device.

In order to better explain how LIQDROID manages the binding procedure between the activity and the service belonging to two different devices, consider the following figure 5.13. The proposed sample scenarios used in this figure is one of the concrete use case scenarios that we have developed to evaluate the service management module in LIQDROID, and we will completely explain it in the chapter 6. Consider that the user wants to play a song using a music player service which exists on device 2; he already has discovered device 2 and is connected to it. The user launches the music player activity on device 1 and asks LIQDROID to get the list of the services which are capable of playing a song (1). LIQDROID prepares the list of the capable components by sending an intent including the user's desired action to the connected devices and collects the results, which here is only device 2 (2,3). LIQDROID on device 2 provides the list by benefiting from the Android framework (4,5) and sends the result back to device 1 (6,7). LIQDROID prepares the list of the available components on the proximal connected devices and adds the list of the capable components existing on device 1 and shows it to the user. The user selects the music player service

existing on device 2, and LIQDROID sends the Launch Intent along with the developer's provided *Original Intent* that has been received previously from the music player activity to device 2 (8). When LIQDROID on device 2 receives this intent (9), it will execute it, which will start the music player service. From now on, the activity is able to interact with the service by sending the Update Intents and receiving back the results from the service through the Feedback Intents. Whenever the user kills the music player activity on device 1, the OnDestroy intent provided by its developer will be sent to LIQDROID (11), and LIQDROID on device 1 will ask from LIQDROID on device 2 to stop the service (12,13). So, the service will be stopped if the bounded activity is not running anymore.

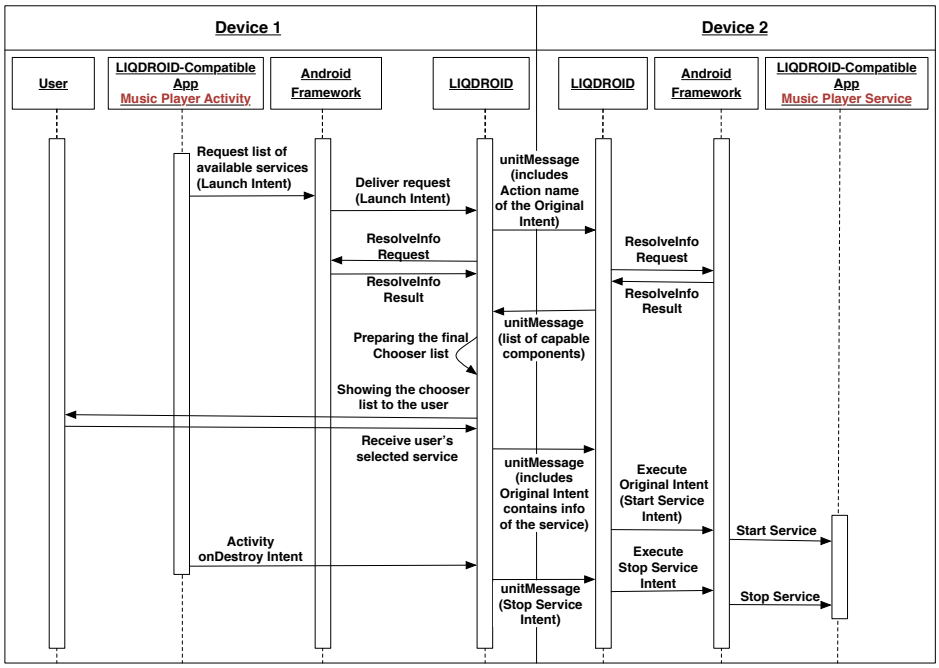


Figure 5.13: *Binding an activity belongs to the Device 1 to a Service that belongs to the Device 2 (an example).*

One of the main features that LIQDROID proposes by the service manager module to the user is the simplicity of transferring the data between the Activity and the bounded Service. For example, in the above sample scenario, the user can transfer the song by using LIQDROID through Google Storage or by directly sending it through the unitMessage. One of the other interesting use case scenarios which LIQDROID can support is about using the sensor measurements that are available on one Android device so that users on the

other devices can benefit from its measurements inside different activities existing in their devices to perform different kinds of analysis or usage of that data.

In the currently available solutions, in order to bind the activity to the service, the activity and the service need to provide the required interfaces at the implementation time, and then the two components are available to interact with each other. Yet, in the above-mentioned solution, the activity is able to bound itself to any third party services available on the current device or on the other connected proximal devices at runtime.

5.9 Communication Manager

When the user wants to connect to the available proximal devices, this module is in charge of sending the permission requests to the devices and also collecting the responses. In case of acceptance, it will connect the devices through the Wi-Fi. If it receives a reject response, it will first notify the user and then provide him again with the updated list of devices excluding the device(s) that has(have) rejected the connection request.

The communication manager module is also in charge of providing the updated list of the connected devices for LIQDROID. There could be two reasons that the already connected devices may become disconnected. The first one comes from the volatile nature of the mobile devices as the device may leave the context at any point in time; the second reason is concerning the Wi-Fi connection interruptions that may cause the devices to become disconnected.

As the devices may leave the interaction without notifying LIQDROID and moreover because the Wi-Fi connection is not always stable, LIQDROID will let the developer send a request to LIQDROID to get the list of all the connected devices at any point in time. This list will be updated by LIQDROID and will only include those devices that are already connected to the source device. So, it is necessary to monitor, at the time of the request, the last status of the Wi-Fi connectivity between the devices.

5.10 Settings

As we previously mentioned, LIQDROID is a service which continuously keeps running in the background and lets the user's device advertise itself. Although the execution of this service (LIQDROID) will not cause any interference with the normal execution of the other Android components, it can become annoying for the user if he is not interested in receiving any

connection requests from the other proximal devices. To solve this issue, the user is able to disable the advertising feature temporarily and reactivate it when he wants.

Contrary to the advertising mechanism which is continuous as long as it is active, the discovery mechanism is under the control of the user and needs to be completed when he wants to connect to the other devices and send a task distribution request. Performing the discovery right before executing a task will enable the user to have access to the latest updated list of the available proximal devices (which may have newly joined the context) as well as the updated groups list that will include the list of those currently available and exclude those that are disconnected or left the context.

The other feature that is accessible through the settings is the possibility to change the device name. As in some of the Android devices the user is not able to change the name of his device or provide more information for the other user which performs the discovery, we have provided the possibility to assign a name to his device so that the user is able to access the setting menu and the change name option. LIQDROID will show a dialog box to the user to enter the name that he prefers to advertise. After receiving this name, LIQDROID will add it to the advertising message and the other users will be able to see this name along with the other information about the device.

These three methods are widely accessible to the user through the notification bar as long as LIQDROID is running in the background. These settings include features that are useful for the end user and are not accessible by the developers to make any changes.

5.11 Conclusion

In this chapter, our aim was to provide a technical overview of the available features in LIQDROID and the challenges that we have faced in making LIQDROID work properly inside the Android framework. Our first goal was to benefit as much as we can from the Android framework without causing any inconsistency in the normal behaviour of the Android OS or decreasing its performance. In fact, this also enabled us to provide an easy-to-use solution for the developers to develop innovative applications and an infrastructure for the users to interact that is easy to understand and use. In the next chapter, we will discuss how LIQDROID will behave and improve the interaction between the users and the devices by providing some sample realistic use case scenarios.

CHAPTER 6

Evaluation

In this chapter, we will talk about our experience of using LIQDROID in real case scenarios. Although it is possible to use LIQDROID to call the components of the applications that are already available on the Android market, in order to fully evaluate different features of LIQDROID and its strengths, we have developed and implemented some LIQDROID-compatible applications from scratch [1] As we already mentioned, in the designing phase of this middleware we had two simultaneous perspectives. The first was from the point of view of a developer; we saw that he could easily benefit from the provided features of LIQDROID to distribute the execution of his developed application's components without needing to learn new concepts and without experiencing difficulty in implementing the required features of LIQDROID. The second view, which is even more important, was that of the final user that will benefit from LIQDROID. The user should be satisfied, which can be achieved based on two factors derived from the existing experiences in the field of multi-device interaction as follows:

- First, the execution of the provided software on the device should not be annoying for the user while he is not interacting with it, and the

[1] Because of the sponsorship, the source codes of the stable prototype of LIQDROID and its companion applications are only available upon direct request.

normal execution of the tasks on the device must be guaranteed. And while he is interacting with it, the provided features should not degrade device performance or distract the user's focus.

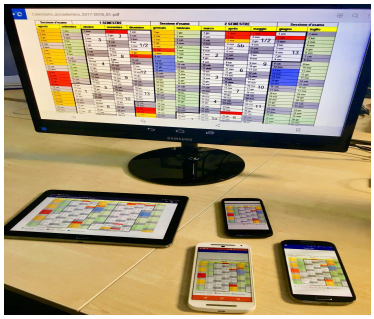
- Second, it should be easy to use and interact with so that the user does not need to spend much time learning how to use it or configure it.

As LIQDROID has been developed inside the Android framework and is using the existing classes inside it, we can say that providing LIQDROID-compatible application components will be so easy for the developers. Since LIQDROID is completely following the Android OS and at the end will benefit from it to execute a task, we can say that the user will not have any difficulty working with LIQDROID. We have provided the proper titles and notifications for the user during the runtime and distribution of a task.

In the following sections, we will introduce the six motivating concrete use case scenarios in different domains that inspired this work. We will explain more in detail the general usage scenarios, their requirements, and the design parameters. Each of these use case scenarios includes some of the proposed features by LIQDROID to better visualize the aspects that we explained in the section 4.

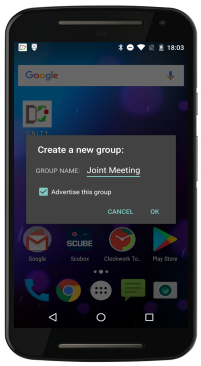
First we need to ensure that LIQDROID, which is an Android application, has already been installed in all the devices involved in the interaction and is running in the background. Second, we need to mention as the preliminary step in advertisement that discovery, connecting the devices and creating groups, will be the same in all the use case scenarios. We assume that discovery has already been completed by the user, so in order to keep from repeating this step in every scenario, we will not directly mention it in each scenario. The only difference in following this step in different scenarios would be the way that the user prefers to define the group name or the user's policy on adding members to groups specifically, which depends on the needs and requirements of the underlying use case scenario. The figure 6.1 shows different steps in discovering the proximal Android devices, selecting those devices that the user prefers and forming the group with the desired name.

In each of these sample developed use cases, we have provided a part which explains the complexity behind developing these applications by defining the number of required components, the lines of codes and the required attempt to learn new things. But what we need to also mention here as a general comment is that one of the main features of LIQDROID is letting devices become properly integrated so that they can communicate (under a user's attention and his preferences). So besides the complexities

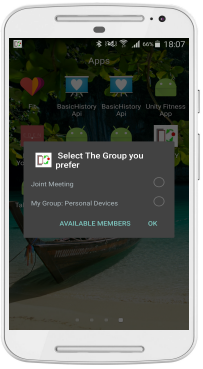


(a) *Android Devices*

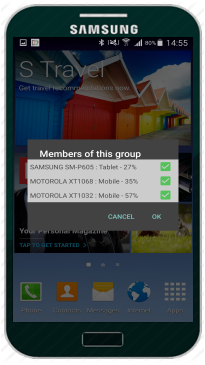
(b) *Device Discovery*



(c) *Group Formation*



(d) *Aavailable Groups*



(e) *Group's members*

Figure 6.1: *Discovery and group formation using* LIQDROID

that we have mentioned in each part, always bear in mind that those efforts should be added to the effort which is required to make two devices become connected if a device wants to interact with the other proximal devices and use their capabilities.

These sample scenarios will show how much distribution of the execution of a task can improve the interactions between the people and improve the usage of different devices while enhancing the features provided by the different components belonging to different applications on different devices, the quality of the achieved results, and finally the satisfaction of the user. We also attempted to provide new innovative paradigms of interaction between devices, which can provide solutions that were impossible without having these sets of integrated devices. These test case scenarios also motivated us to better understand the extra needed features that LIQDROID needs to support, and we will explain them in the related scenarios.

We need to mention that to better understand the industry needs and the users' needs, we also agreed with and applied the opinions of the Telecom Italia researchers at Joint Open Lab S-Cube, who are working more specifically on the innovative solutions to better support the interactions between people and also between people and their surrounding environment. The final applications were also tested and developed on the real case devices available in this lab.

In the end, we also provided some related use case scenarios considered by the other developers in this field to see how LIQDROID is able to support those domains of use and provide the same features.

6.1 Usage of LIQDROID with Available Android Applications

Â Before starting to explain the complete and well-defined use case scenarios that we have considered to test different aspects of LIQDROID, let us consider the most general and simple use case scenario that we can have by benefiting from LIQDROID. It can show you how LIQDROID can make works simple for the developer while at the same time bringing more levels of satisfaction for the users.

6.1.1 Sample Scenario

Consider that you have developed an email application. The user receives an email with an attachment that can be any type of file such as a video file, a document, etc. The user wants to open the file, but there is not any application already installed on his device that lets him view a video, or he may prefer to have a better application which provides more features for him to interact with the video. Here is the place that LIQDROID can facilitate your work a lot. You as the developer of the email application can easily provide a button inside your application that sends a request to the LIQDROID. LIQDROID will provide for your user a list of the capable components to play video on the proximal devices. As soon as the user selects his desired Media Player application, the file will be transferred to the other device and the user is able to view the file through his preferred component.

6.1.2 Application's Architecture and Complexity

In this case, the developer only needs to provide a few lines of code (9 lines), as we mentioned, to a sample Launch Intent in the section 5.5 to enable the

user to view a video that does not have a proper application already available on the device.

If you want to properly support your users' needs and attract more users to install your application, you may decide to provide a proper component inside your email application to support more features. Note that it will take much more effort to provide a separate component to support different kinds of files than to just create a usual intent.

For example, consider that you want to support playing video files through a video player component. To this end, beside the XML files that you need to define in a time-consuming process in order for videos to be viewed, you will also need to create an activity to define its logic. According to the codes that have been proposed by the Android developer website [2], for a very simple Media Player you need at least 62 lines of code (added to at least 56 lines of code for its view = 118) to develop its component, besides the time required to research and learn how to implement each part. On the other hand, you can benefit from LIQDROID by calling the best available applications that are installed on the proximal devices to provide the required services for your application with much less effort (an Intent with 9 lines of code).

One of the main reasons why each application needs to provide this button to request the list of capable devices and why we did not provide this general button inside LIQDROID is that most of the time the underlying content used by different applications is stored in a private file under the direct control of that application, and it does not allow any content provider that provides other applications such as LIQDROID to have access to its data.

6.2 Use case scenario 1: Joint Meeting Application

The main focus here is on the presence of the multiple users and devices in the proximity of each other. They need to have a fully cooperative infrastructure that empowers them to easily and quickly collaborate with each other to achieve more productivity, so the concern is: How can we benefit from the proximity of the users and devices to increase the productivity and enhance the cooperation among them to achieve better results?

More precisely, we can mention that the following aspects were our main focus to answer the above concern and to support the design phase of this use case scenario:

[2] https://developer.android.com/guide/topics/media/mediaplayer.html

- Supporting the availability of different users (with different roles) that may join and leave the context more often
- Supporting the availability of different types of devices and configurations required to make them cooperate
- Supporting the distribution of different types of tasks (Device Shifting, Complementarity, Synced Devices)
- Supporting the exchange and availability of different types of data for different tasks

6.2.1 Sample Scenario

Today's life is more social, and teamwork is highly appreciated as it can have a significant effect on the outcomes and the accomplished responsibilities. But if the meetings are not conducted properly, they could bring negative results along with the waste of time and resources. Although different platforms and technology-enabled devices have been introduced to support interaction between people, still there is a place to improve them more. And this is the place where direct interaction between multiple devices can be highly effective. To better clarify the usage, let's focus on the meetings, which is a good example of teamwork. Also, meetings are more probable to happen in different contexts, including those related to business, education, or more informal purposes such as among friends or colleagues for brainstorming. In analyzing the currently available platforms such as [3] that are widely used for handling conference meetings, it was determined that one of the big drawbacks of using these platforms is that even if two proximal members join to a meeting while the other members have been joined from long distances, these two proximal members will be treated the same as those remote members. And each of the members needs to have their own account to have access to the platform. Besides the expensive membership prices of these platforms, they will cause other issues, some of which we will explain further. These issues can restrict the users' activity while will also wasting resources, which in the end will result in most of the employees feeling as if they are wasting time during these meetings.

- Separated Accounts: Each participant needs to enter the platform environment as a separated member, so each of the employees even in the same office needs to have several accounts to be able to join to the meeting and have access to the meeting's resources.

[3] https://WebEXhttps://www.webex.com/products/web-conferencing.html

- Devices occupied: As the user has been joined to the meeting through his private account, he needs to keep the meeting application running on his device even if he is not going to present anything.
- Restricted access: While the user is attending the meeting, he is not able to interact with his personal devices that may exist in his proximity.
- If a new member wants to join the meeting, he needs to have been invited in advance to have access to the link (or the pass code) of the ongoing meeting.
- The Internet bandwidth usage during the meeting can be problematic.
- The interferences that these devices can cause to each other can be disruptive.

Some of the features which should be considered to increase the quality of the meetings are those that allow easy collaboration, easy configuration, and freedom for the participants. The presence of multiple users with multiple devices in a context that can form various types of interactions between them was a good motivation for us to develop this app and better test LIQDROID in a highly collaborative situation. This helped us to provide comprehensive support for these multi-device interactions which can increase the user's enthusiasm to join the meetings and significantly improve their productivity.

To overcome these limitations as well as provide a better experience that the multi-devices interaction can bring to the user, we have developed a meeting application integrated with LIQDROID in our lab which is called "Joint Meeting Application." This application includes the features which can be found in the current meeting applications and also innovative solutions to overcome the existing issues. We have provided the Joint Meeting Application with the possibility to integrate with the available meeting platforms to better support remote participants. So the provided application is able to handle the meetings that have participants ranging from the remote members to the local employees/staff. This LIQDROID-compatible application has two separated parts based on the role of the person that wants to join the meeting, which could be the presenter or the participants.

6.2.2 Application's Architecture and Complexity (Presenter Version)

The version that will be used by the presenter of the meeting will include the following capabilities to support each of the meeting requirements.

One of the advantages of the group formation here is that when the presenter connects to the proximal devices, he can create a group with the

name of the place that the meeting will be held or the subject of the meeting and it will advertise it. It will be more efficient for the new incoming members that are not formally invited but want to participate in the meeting. They can discover the groups' list and be informed about the ongoing meeting and quickly join the meeting. The other invited participants can also take advantage of it to easily find the list of the participants' devices that are attending the meeting instead of going through the whole list of all the available proximal devices.

The presenter will receive the connection request and can accept those members that he prefers to actively participate during his presentation. To eliminate the interruptions, at any moment the presenter is able to stop the advertising so that he will not receive any connection requests during his presentation.

In the local meetings, the presenter's device will only be connected to the projector to support the ongoing meeting. While in general meetings that remote members are attending, the presenter's device will use a companion application of an integrated meeting platform-here we have used WebEx. So the presenter will use his account credentials to enter the WebEx application to share his screen with the remote members. We kept the presenter's view as simple as possible with only the necessary features. The version of the application that he uses includes only the following parts:

- List of the participants: This is the list of the local participants that the presenter's device is connected to through Wi-Fi.
- Shared Interface: This part that will be used to show the presenter's view (camera).
- Role Exchange: The presenter is able to perform a long click on the participant's name and exchange the role of the presenter with them for a short time.
- Assign New Presenter: For any reason such as the presenter's device has a low battery or the presenter wants to leave the meeting, the presenter is able to share the meeting link with one of the connected members that he prefers to take his place. The new members will use the same credentials to enter the WebEx and share his screen with others as the new presenter.

We have integrated our meeting application with the WebEx meeting platform to support the remote members. In this way only the presenter needs to join the WebEx platform through using a dedicated general account.

The participants will use our developed application to join the meeting. In the following section, we will discuss how LIQDROID can handle the meeting without the need for each participant to have a private WebEx account. The user is also able to more actively participate in the presentation and benefits from his device at hand to interact and share his related ideas and materials with the others. As the presenter has full access to the big screen and direct access to the WebEx platform, the application is simpler. We will discuss the main features that have been included in the participant's version in the next section.

6.2.3 Application's Architecture and Complexity (Participant Version)

The version that will be used by the participants of the meeting is an Android activity component which will include the following capabilities to support each of the meeting requirements:

- Share Files:
  One of the time-consuming parts of the meetings happens when a participant wants to convey his opinion by sharing a document. In this situation, he will need to connect his device to the big screen in the local meeting or change his role to the presenter in the remote meetings to be able to share the files with others while he is providing his explanation about them. By benefiting from LIQDROID, the participants can share files simultaneously between each other or by accessing the presenter's device through their device and executing the task there. To this end, each of the participants that are connected to the presenter's device can distribute the execution of the task there. Consider this simple scenario: in an ongoing meeting you have a PDF file which is highly related to the subject of the discussion and you want to mention some parts of it during a formal meeting. Currently, there is no way other than using another application to send the data to the presenter's device, launching the proper application there, selecting the PDF file, opening it, and then you can start explaining it. These set ups usually require long configurations and waste a lot of time during the meetings, so most of the time people prefer to bring the printed version of the required documents to give to all the other participants.

  Instead, by using LIQDROID, each of the connected participants can use the "Share File" to use the presenter's device to share the material and their opinions in a few quick steps as follows: the user will select the PDF file from his device and send the execution request to LIQDROID. The LIQDROID receives the request and shows a list of the

capable components to show PDF files on the proximal connected devices. The user selects one of the (preferably LIQDROID-compatible) components which are available on the presenter's device and distributes the execution of the task to that device. The LIQDROID on the user's device uploads the PDF to the Google cloud storage and sends the execution request to the presenter's device. The LIQDROID on the presenter's device downloads the file then executes the request. The user starts to discuss his idea while he has also opened the file on his device. The user is able to control the execution of the PDF file executed on the presenter's device through his device. So as soon he goes to the other pages, the state of the PDF file on the presenter's device (and also on the big screen) will be updated and will shift to that same page. Here we have explained the PDF file, but the user is able to select any type of file and send it and control its execution.

At the end of the task, the user will finish the video player activity on the presenter's device. Another possible concurrency of the usage that may happen here is that another user will want to also provide his opinion and needs to use the presenter's device. In this situation he will send the execution request to the presenter's device and LIQDROID will handle the new request as we have explained in the section 5.7.2. Either this participant will wait until the previous participant has finished the presentation of his opinion or he will persist on executing his task. These two choices will be available in the dialogue box provided by LIQDROID.

- List of the Members:
  Besides connecting to the presenter's device, each participant is able to connect to all or a certain set of the other participants that he wants to collaborate with during the meeting. The list of the available members will be accessible through the application, which includes the connected devices to the participant's device through Wi-Fi.

- Private Chat:
  The user can select the preferred members from the list of the members and send them messages during the meeting without the need to directly talk to them and cause disruptions during the meeting. These messages would be considered as regular Update Intents and will be sent to all the connected devices, but only those members that the user has selected can see the messages in the view box.

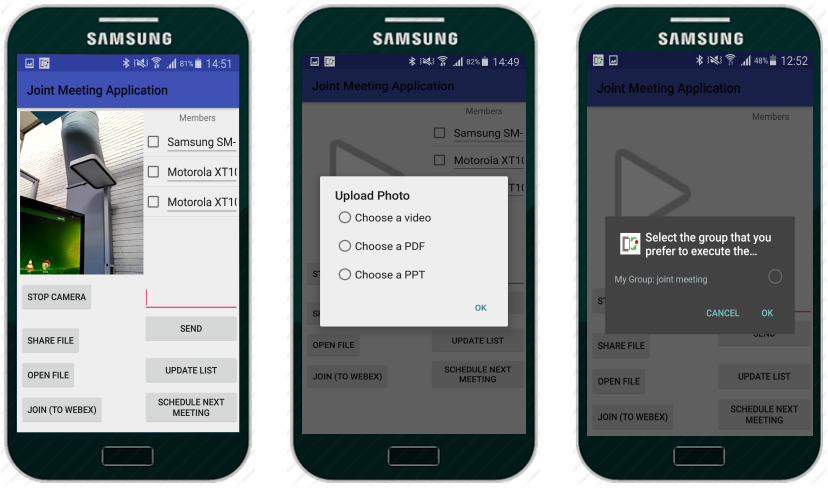
- Camera Streaming:

As we mentioned earlier, the participant is able to use the presenter's device to share the file needed to better explain his opinion to others. Sometimes it may happen that the presenter wants only to discuss something. In this case, he is able to start the camera and share his view with the presenter's device without the need to reach it. So the members will see the participant's view in the presenter's Shared Interface while he is presenting something. It will use the regular Update Intents defined in LIQDROID for streaming the video on the presenter's device.

- Acquire Presenter Role:
  On the other hand, if for any reason the presenter's device becomes unavailable, such as when the battery of the presenter's device gets low or the presenter wants to leave the context, one of the members can easily share his device to take the presenter's role. In this case, clicking on "Take Presenter Role" will notify the presenter about his choice and the potential to take his role. The presenter is able to make the most appropriate choice from among the received requests. As soon as he has chosen one, the other reached requests will be discarded.

- Download Meeting Material:
  At the end of the meeting, participants can download all the materials that have been used by the presenter or shared by the other participants. To this end, the meeting's participants will have access to the files used by the presenter's device (Source Component).

- Schedule Meeting:
  Based on the subject of the meeting, different types of collaboration between the members may be required. For example, it may happen that the users need to vote to select a choice which is in the interest of all the participants. To this end, it is required that the same task should be started on all the devices to receive the user's choice. Here we have provided a general use case which is more probable to be used by different meeting categories, which is voting to select a proper time slot to arrange the next meeting. "Schedule meeting" is a service included in the joint meeting application that lets the members share their preferred date and time among each other to conclude arranging the next meeting. For this purpose, one of the members clicks on the schedule meeting button, selects his preferred date and time, adds them to a list and sends the service execution request to LIQDROID. The LIQDROID provides the list of the connected devices that have this



(a) *Participant version* **(b)** *Material To share* **(c)** *Device selection*

Figure 6.2: *(a)Set of available features on the Participant component during an ongoing meeting, (b) The possibility to share various files during the meeting (c)Possibility to select and share the materials with a sub set of a group's members*

service component, the user selects those members that need to attend the next meeting, and LIQDROID runs the service on that set of devices.

Upon the users' acceptance on the selected devices, the service will start running and will open a dialogue box. The users define their proper time slots and send the responses back (Feedback Intents) to the requester device (source device). When the user on the source device has received all the responses, they will define the final date and time and send it as an Update Intent to all the members, which will notify them of the time of the next scheduled meeting.

As the participants are not directly connected to the WebEx, they can use their devices and only launch the joint meeting application when they want to present something or want to interact with the proximal members.

6.2.4 Application's Architecture and Complexity

Most of the complexity of this application was related to defining it by collecting the current status of the meeting applications and defining the participants' requirements during a meeting. While developing it, we have used three main activities: (a) Entering into the application based on the

participants' role; (b) Presenter activity; and (c) Participant activity. Here we just mention the complexity in the development of the Participant activity as the two others are not very complex.

As you can see in the figure 6.2, there are different buttons to send requests to LIQDROID. The user will have the possibility to view different contents through the available components on the same device or the other proximal devices. We provided three other activities for viewing different files such as PDF, video, and PPT, which enables the user to interact with them while running on the proximal devices.

As an example, if a developer wants to implement an application to open a PDF document on the other proximal device(s) and control it through the source device, he needs to connect the devices, transfer the PDF file, launch the proper application there, open the PDF file and then update the states of the two devices. Let's make it so simple and just consider the last part. Sending the state updates as the other parts to seamlessly resume a task was one of the main concerns of LIQDROID to solve. In order to send regular updates to synchronize the state of the proximal devices, the easiest method is for the developer to benefit from the Google Nearby Message as mentioned on their website [4]. It needs at least 55 lines of code solely for implementing this part to use for sending the state messages, besides learning how to use it. He also needs **each** device to implement a service on both sides to send and receive the state message and to communicate with the PDF activity on the other device (at least 40 lines of code [5]). Whereas by benefiting from LIQDROID the developer just needs to provide an Update Intent as follows, which requires only 11 lines of code to update the state of the PDF file on **all** the other proximal devices (in comparison to 95 lines for each device). Additionally, he does not need to put forth more effort to learn new things.

```
//The Original Intent for updating the state of a pdf on the proximal
    devices.
1-Intent pageNoToSend = new Intent();
2-pageNoToSend.putExtra("updateType","String");
3-pageNoToSend.putExtra("PageNumber",pageNumber);
4-pageNoToSend.setAction("Update");

//The Intent which will be used to communicate with the \textsc{LiqDroid
    }-service and to deliver the Original Intent
5-Intent startLiqDroid = new Intent();
6-startLiqDroid.setComponent(new ComponentName(("com.android.imani.
    liqdroid", " com.android.imani.liqdroid.MyService"));
7-startLiqDroid.putExtra("originalIntent", pageNoToSend);
8-startLiqDroid.putExtra("SourceAppComponentName", "com.example.apple.
```

[4] https://developers.google.com/nearby/messages/overview
[5] https://developer.android.com/guide/components/services.html

```
        jointmeetingapplication.PdfControllerActivity");
9-  startLiqDroid.putExtra("SourceAppPackageName",
"com.example.apple.jointmeetingapplication");
10-startLiqDroid.setAction("appIsRunning");
11-startService(startLiqDroid);
```

This was only one of the simplest comparisons for the attempts that the developer needs to assign to provide the same feature with and without LIQDROID to update the state of an activity on the proximal devices.

6.3 Use Case Scenario 2: Cameo Application

One of the main characteristics of the mobile technology enabled devices is that the user can benefit from them on the move while entering into different contexts. This feature will considerably affect the usability of the applications as mentioned, and most of the time the user needs to perform some preliminary steps to initialize and resume the task on the other device. Here the main concern is: How we can facilitate the task distribution between the devices that exist in different contexts? More precisely, we can mention that the following aspects were our main focus to answer the above concern and to support the design phase of this use case scenario:

- Supporting the task distribution while the user is moving
- Supporting the seamless interaction between the devices
- Supporting the synchronization and collaboration between the different sets of devices

6.3.1 Sample Scenario

The second use case scenario is an application for Android Auto which was developed in our lab called "Cameo Application." The application offers the passengers inside a car connectivity on board and access to the entertainment content, travel information, services, and context-based messages. One of the main features of the application is the possibility of planning a trip, which allows the passengers inside the car to be able to suggest their points of interest and collaboratively create a trip. Each user can add their desired destination, and the application will calculate the route by using the Google Map API.

Consider this use case scenario: Maria and her colleague, Tom, have a meeting with one of their customers. She will join Tom in a predefined location, and then they will go to the meeting's location together. Maria launches the Cameo application on her smartphone and inserts the final

destination as well as the middle point, which is the place that she will meet Tom. The application calculates the path; she leaves the lab while she navigates toward the destination. In the middle of the path, she reaches Tom and enters to his car, and they follow the path. As soon as Maria enters Tom's car, the discovery starts and connects to the Android tablet available in his car. Tom accepts the authentication request on the tablet. Maria selects the transfer path button on the Cameo application. The LIQDROID on her device shows a list of the available capable components on the tablet and her own device. She selects the Cameo application's component on the tablet from the list. Then the application's map component is launched there with the information that is in the launch intent (current location and the destination). They start navigating to the destination by using the car's tablet without configuring anything manually. Maria continues interacting with the Cameo application on her own device to find a place to eat lunch on their path. As soon as she finds the proper place, she sends an Update Intent to the tablet device through LIQDROID and the Cameo application, which includes an added middle point that is the location of the restaurant. The component on the tablet updates the path with the new information that has been received through Update Intent.

6.3.2 Application's Architecture and Complexity

This is one of the sample scenarios that have been implemented in our lab by the Telecom employees, and I had the chance to collaborate with them. One of the obstacles that exists in the Cameo application that made us integrate it with LIQDROID was about launching the Cameo application on the tablet as soon as the user enters to the car. So the execution of the running task on the user's device will be seamlessly distributed to the car's tablet without the need to reach the tablet. The cameo application is launched and configured there. So in this case as soon as the driver enters the car or a passenger enters, they are able to resume an ongoing task from their cell phone to the car's tablet. This is one of the very important features provided with LIQDROID, and other provided middleware technologies -specifically the AllJoyn [1] that has also been used by the Cameo application- are not capable to handle it. In order to start the interaction, the user first needs to start the Cameo application on the car tablet and then is able to synch their state or start their interactions through the AllJoyn middleware.

The cameo application has two different versions-one for the passengers and the other for the dashboard tablet. The user is able to provide the information of the destination and also the middle points through the

Android activity on his own device. To use LIQDROID to resume the path on the dashboard tablet as soon as the passenger reaches the car, they need to provide the Launch Intent for LIQDROID. As soon as the user launches the Cameo application on the dashboard tablet, he will also have the possibility to update its states. To this end, they have also considered using the Update Intent from LIQDROID to synchronize the state of the devices of the passengers and also the dashboard tablet.

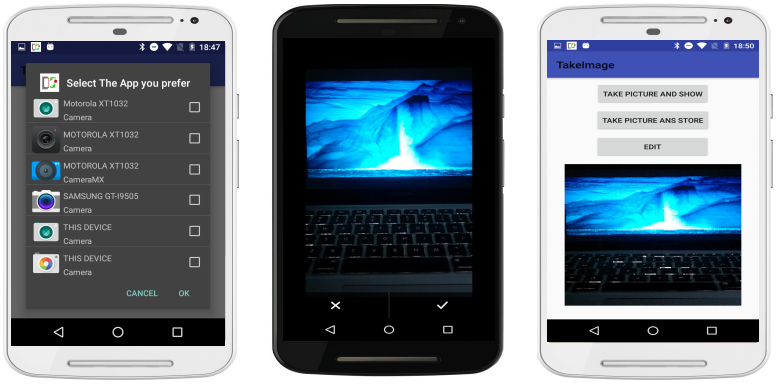
It was also interesting for them that LIQDROID would give them two possibilities to handle the task distribution at launch time as they could benefit from their own Cameo application (through using an explicit intent) on the distribution of the task or they can also let the passenger have the possibility to choose another component available on the dashboard tablet (through using an implicit intent) to resume the path. They decided to provide both possibilities for their end users.

This sample shows just how easy integrating any application with LIQDROID is. It can happen in a few minutes and just minor changes need to be applied.

6.4 Use Case Scenario 3:Take and Edit Image Application

Most of the time it happens that the user is forced to use different devices during the execution of a task. The reason could be because of the device limitations or his preferences to achieve better results. In this case, first the user needs to know what the available devices are and needs to be able to compare the task requirements and the devices' capabilities. On the other hand, he also needs to be able to easily move the task and its artifacts between them. The real problem arises when the number of devices increases in the interaction. So here the main concern is: How can the user get a general view about his possibilities to perform a task? And how is he able to manage the distribution of a task while using several devices interchangeably? More specifically, we can mention that the following aspects were our main focus to answer the above concerns and to support the design phase of this use case scenario:

- Supporting the task distribution and redirection based on the device's capabilities
- Supporting the storage and management of artifacts
- Supporting the data availability for the third parties in the interaction before the execution of the dependent task happens



(a) *Capable components* **(b)** *Captured image* **(c)** *Revie the captures image*

Figure 6.3: *(a) Possibility to select the proper device and the proper component on it, at the same time (b) Capturing an image through the desired component on the destination device, (c) Reviewing the image on the source device.*

- Supporting the synchronization and collaboration between the different sets of the devices

6.4.1 Sample Scenario

Considering that Tom is working in an architecture company. He needs to take a picture from the Politecnico di Milano's main building, and after editing it, he will show the updated image to his boss and other colleagues while he is giving his presentation.

First, in order to see how good the cameras of the other devices are, he takes sample images and review them on his device. To this end, Tom launches the Take and Edit Image Application (a LIQDROID-compatible application) on his device and asks LIQDROID to bring him the list of the devices that are able to capture an image. Each time, Tom selects one of the Camera Applications already installed on the proximal connected devices and tests the quality of their camera and the provided images. So LIQDROID saves the images that Tom has taken on the other devices and shows them to him on his device. The figure 6.3 shows these interactions between the devices.

After that Tom selects the device that he thinks has a better camera. He decides to take the image from the building with that device and send it to his colleague for edits. So he repeats the procedure to take the image through selecting the camera application on his desired proximal connected

(a) *Device with a good Camera* **(b)** *List of Edit applications* **(c)** *launched edit application by the image*

Figure 6.4: *(a)Captured image by the device that has better Camera, (b) List of the available applications for editing an image, (c) The availability of the captured image on the third destination which proposes a better application for editing it.*

device. Then he receives a message which shows that the image has been taken properly and has been stored in the Google cloud storage successfully through LIQDROID. At this time he clicks on the Edit button in the application, which sends a request to LIQDROID to show him the list of the applications capable of editing an image on the proximal connected devices. As the figure 6.4 shows, Tom selects the edit application that he prefers on his colleague's device, Stefano (a different device than the one that he has used to take the image). The LIQDROID on Stefano's device downloads the image and then launches Tom's selected edit application by using that image. Stefano starts editing the image and saves it through LIQDROID. As soon as the boss arrives, Tom is able to ask LIQDROID to provide him with the edited image on his own device, or he can simply can use the big Android screen to show the final edited image to his boss and other colleagues, who can then give their opinion about that nice building.

6.4.2 Application's Architecture and Complexity

There would be two activities; one of them will include three buttons running on Tom's device which sends the requests to LIQDROID. The other one will be used to view the final edited image. The first button will send a Launch Intent to LIQDROID to show to Tom the list of capable components to capture an image. The proposed list will also include the general camera

applications which are available on the other connected devices. After Tom has selected the proper component (which is one of the general applications), LIQDROID will store the result of the execution of that activity (the captured image) through its Artifact Manager Module for Tom (the running activity on Tom's device). And he will be able to ask LIQDROID to use it as an input for the other components that he chooses and will be running on the other devices.

This sample shows that in order to benefit from LIQDROID we don't need to have a LIQDROID-compatible component to distribute the execution of the task on the destination devices, but LIQDROID will also work fine with the available components in the Android market and can benefit from them to execute tasks and store the final results achieved from their execution. On the other hand, the consistency between using the output of an executed task as an input for the other task is fully supported, and there is no need to perform extra tasks (other applications' components) to manage the availability of the data on the other devices to continue the task.

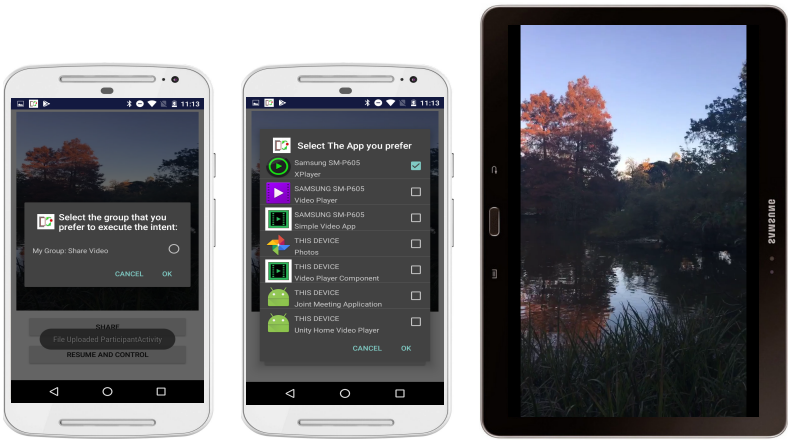
6.5 Use Case Scenario 4: Home Video Player and Controller

While the user moves between different contexts, it may happen that a different set of devices are available in his proximity. This will enable the user to have the possibility to shape different interaction types and assign the execution of each part of a task to a specific device. So here the main concern is: How capable is the user of distributing the execution of a task based on the device availability as well as the interaction requirements?

More specifically, we can mention that the following aspects were our main focus to answer the above concern and to support the design phase of this use case scenario:

- Supporting different combinations of the devices in handling the execution of a distributed task
- Supporting the availability of different devices in the user's proximity and sharing the task based on them

The user may be placed in different contexts where a diverse set of devices would be available in his proximity. The possibility to be able to resume a task on the other device is important, but the possibility to make devices collaborate with each other to perform a task is another nice feature provided by LIQDROID. You may be sitting on a train with a friend, and you want to share with him the video that you are watching on your smartphone. So you connect to the tablet of your friend and launch the video component



(a) *Connected to friend's device* **(b)** *Selecting a video player* **(c)** *Video player on tablet*

Figure 6.5: *(a)Sharing a video with your friend (b)* LIQDROID *will show the list of the available video players on your fiend's device upon it uploaded the file successfully to the cloud storage (c)The desired video player will be launched on the destination device while playing the video that has been downloaded successfully by* LIQDROID.

by selecting the video through your device. You reach the destination, and you stop the video until you reach home and watch it with your family on the big screen.

The home video controller lets you distribute the execution of the video that you are watching, and you will be able to resume it on different devices. You will also have the possibility to distribute different parts of a video player to the set of devices which are in your proximity. All these pieces will come together in your device at hand when the other devices are not accessible anymore. In this case, you are able to watch the video on the big screen while you are also able to control its execution through the controller options that are available on your smartphone.

After the video view has been launched properly on the big screen, LIQDROID on the user's device will receive an acknowledgement. The LIQDROID will inform the running activity that the distributed task has been handled properly, so the activity provides the list of available options to control the execution of the video on the big screen. Whenever the user performs a click, the effect of it will appear on the big screen instead of the method executed in his device.

If the user selects another task to be executed on the big screen, which

could be another video or another content, the previous activity will be finished there, and then the new one will be launched. So the user does not need to reach the big screen later to close the activities which rest in the background. In the video controller, the options to handle concurrent activities such as receiving a call while he is watching the video also have been considered.

6.5.1 Application's Architecture and Complexity

For this scenario, we have developed an activity which has a part to view and control the video and two buttons to share a task and to transfer a task. The share button lets us share the video that we are watching with a friend who is nearby. In this case, the video player on our friend's device is a general one. We transfer the video and execute it with one of the applications which are available there.

Let's focus on the transfer button, which lets us assign different parts of a running activity to different available devices in our proximity. To this end, we have only developed an extra activity (it can even be a dialog box) which includes some button to control the video which will be played on the big screen. These are just simple buttons with the name of the methods that we want to be performed on the big screen such as play, pause, stop.

When the user clicks on the transfer button, an intent will be created which also includes the *"Device Role"* field as we mentioned in the section 5.5.1. The running video's name along with the other fields are required to put in a Launch Intent. As soon as the Launch Intent is received by the big screen, the main activity will be launched while the two buttons will disappear. As we have defined in the logic of the main activity, if it receives an intent which has the role of Controller inside it, these two buttons should be visible on the screen.

On the other hand, LIQDROID also will consider the role of the big screen as the Client. So as soon as the user decides to close the current Controller activity on his device or launch another activity, it will ask the user to choose the proper action for the main activity which is running on the big screen. If the user decides to finish it, the main activity will be resumed with the last state of the video on his device.

Here we had the same application on both sides, but by benefiting from LIQDROID, we could show two different experiences and views of it to the user without the need to put more effort on designing and implementing extra activities to handle these cases.

6.6 Use Case Scenario 5: Music Player Service

As we mentioned already, LIQDROID gives us the possibility to benefit from the services which are available on the other devices and control and interact with them through the activities which are available on our device. So here the main concern is: How capable is the user of binding an activity to a service and controlling it while they are executing in separated devices?

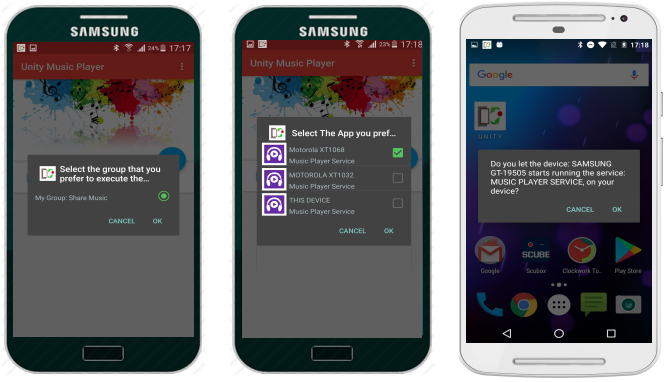
More specifically, we can mention that the following aspects were our main focus to answer the above concern and to support the design phase of this use case scenario:

- Supporting the interaction between an activity and a service in separated devices
- Supporting the availability of the data which will be required by the service

6.6.1 Sample Scenario

One of the limitations that already exists in the accessories which are enhancing the available hardware features of a device is that you can only connect them to a single device at a time. For example, if you want to listen to music through an external speaker which is currently connected to your tablet, you first need to disconnect it from the tablet and then connect it to your smartphone. Then you are all set to play the song by using it. These configuration sets sometimes take longer than usual if the devices are not able to properly find or connect to each other. But by using LIQDROID, you can easily discard these extra configuration sets as you can open a music player on your device, bind it to a service on your tablet, and start to play the music. In the version that we have developed, the user is able to select music from his smartphone and send it along with the service execution request to the tablet. The LIQDROID on the smartphone uploads the song to the Google cloud storage while LIQDROID on the tablet downloads it and starts the music service to play it as soon as the download finishes. While the song is playing, the user is able to control it through his smartphone or can go to the next song that is available in the storage of the tablet. Or vice versa-if a music player service is available on your smartphone, you will be able to transfer music from a tablet to the smartphone, and you are able to listen to it after you leave home.

The next time that the user wants to send the same song to play on the tablet, LIQDROID will notify him that this song has already been sent as it is available in the file assigned to the component in the source device.



(a) *Music Player Activity* **(b)** *Music Service on other devices* **(c)** *Permission to run service*

Figure 6.6: *(a)Through the Music Player activity the user chooses to run the service using on the other proximal devices, (b) The user selects the Music Player Service on one of the connected devices, (c) The user on the other device receives a permission request to let the other user runs the Music Player Service of his device.*

6.6.2 Application's Architecture and Complexity

We have developed a LIQDROID-compatible activity which contains some mp3 songs as well as the buttons to control the execution of the song on Device A. And on the other hand, we developed a LIQDROID-compatible service which is in charge of playing an mp3 song and installing it on Device B. The activity will send his requests to launch the service as well as control it through the Launch Intent and the Update Intent.

Designing the view of an activity and providing a good logic behind them is always a hard task for developers. With LIQDROID you can benefit from any third party's activity to bind it to your service and benefit from its features. For example, there could be an activity that shows more information about a song or it can perform some statistics and provide you with a list of songs you might be interested in. So you can easily find and select the proper song while benefiting from your service to execute it without the need to develop another activity with these new features.

6.7 Use Case Scenario 6: Inside Shoe Store Application

One of the main problems in the multi-device contexts is that sometimes we need to execute a single task in parallel or to synchronize the state of several devices at the same time. This could be even harder if those devices

are not easily accessible. So here the main concern is: How is the user able to initialize a task on several devices at the same time, synchronize them, and also retrieve and manage their results?

More specifically, we can mention that the following aspects were our main focus to answer the above concern and to support the design phase of this use case scenario:

- Supporting the initialization and synchronous execution of a distributed task on several devices
- Managing the results achieved by the parallel execution of several distributed tasks

6.7.1 Sample Scenario

Consider a shoe store that has different sellers which are responsible to add the newly arrived products after they put them in their places in the shop. To this purpose, they will use a LIQDROID-compatible application to add the product items. Additionally, if a customer asks about the availability of a pair of shoes, they are able to check its availability through the application. As we have mentioned earlier, LIQDROID is capable of storing the structured data as well as unstructured data. In order to handle this situation, we want to use only one of the devices which is more reliable (We call it *Storage Device*). We have placed the required database to keep the values. While the other proximal devices are able to insert the information of the newly arrived products or even make a query upon the customer's request on that adhoc database, in this way all the products' information will be saved on a single device which is always available in the store while all the sellers are able to have access to it and update its data. In the following sections, we will explain more about how the user is able to apply, insert and make a query through LIQDROID on a remote database.

6.7.2 Application's Architecture and Complexity (Insert Products)

In the middle of the day, the manager of the shop decides to have an overview of the availability of the clothes in the shop. So he starts the Cloth Shop application developed by us. He starts the discovery and connects to the devices of the sellers that he wants to add available products to. As you can see, the figure 6.7 shows the Activity 2 on device 2. The user will add the product information and save the values in the cloud storage by using LIQDROID. The LIQDROID will treat the results based on the rules that we explained in the section 5.6 to manage the result of an activity. Later,

the manager on the *Storage Device* sends a query request to LIQDROID to update the results of the database. The LIQDROID downloads these values that have been uploaded by different devices and inserts them in the database. The manager applies the show results and query on all values available in the database to check the products that he needs to order.

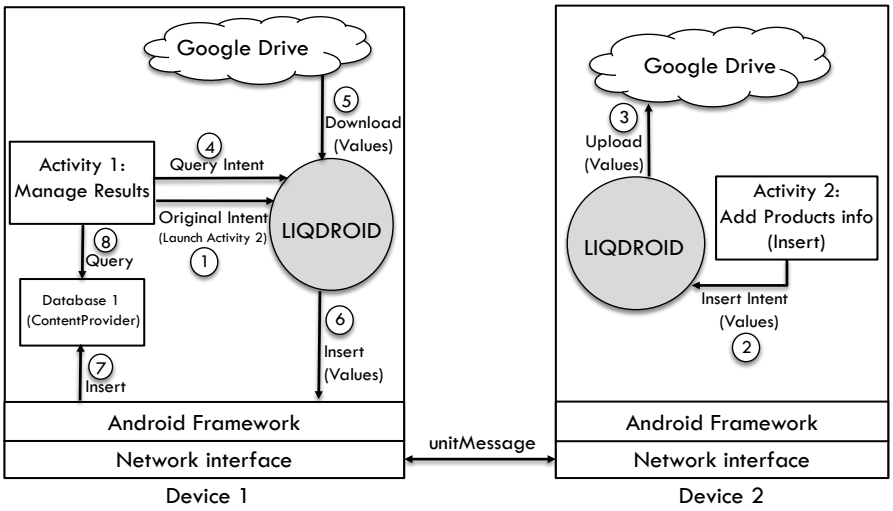


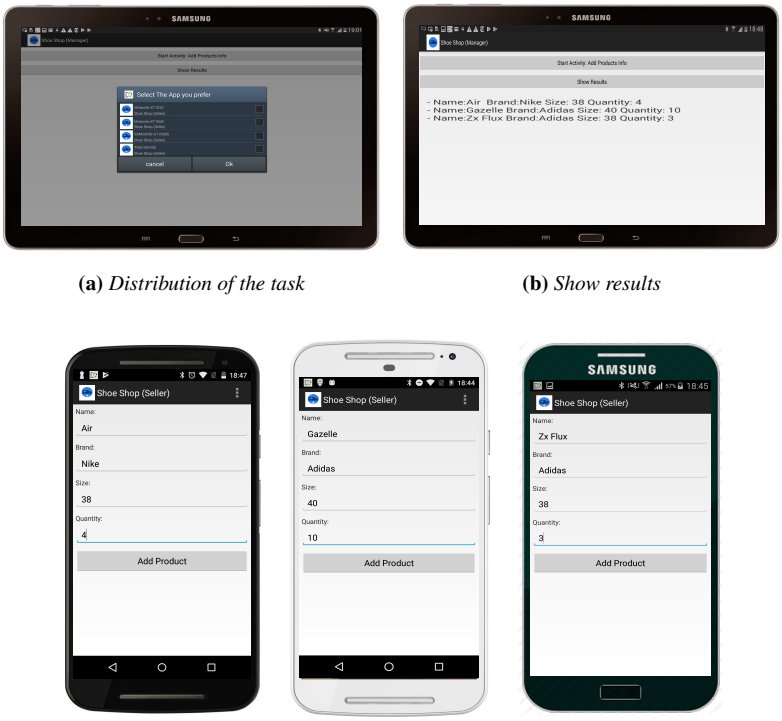
Figure 6.7: *Inserting unstructured data through* LIQDROID

This will enable the user on a device to periodically save the values in the Google cloud storage and, with the user's permission, on the device which has the required database. So the user does not need to provide the same instance of the database on several devices, but it is enough to store the results on one device.

6.7.3 Application's Architecture and Complexity (Search Products)

The second use case scenario is related to the sellers that want to check the availability of the products. Although the database is on the *Storage Device* which is in their proximity, the sellers are able to apply a query on it through LIQDROID and view the results of the activity on their own device.

The figure 6.9 shows how the user is able to make a remote query call on the data available on the database on Device 1. In the following, we will explain more about how the procedure will work in a Shoe Shop that we have considered previously. The Seller, Elena, who has Device 2, applies the discovery and connects to Device 1. She launches Activity 3 to check the availability of the products. To this end, she needs to apply a query on



(a) *Distribution of the task* **(b)** *Show results*

(c) *Sller1 adds product inof* **(d)** *Sller2 adds product inof* **(e)** *Sller3 adds product inof*

Figure 6.8: *(a)The user selects to launch the activity to add products on the Sellers' devices. (c)(d)(e) The selected sellers will add the product info in the launched activity through their devices. (b) The manger later is able to apply query and view all the inserted results through* LIQDROID *in the database on his device.*

the data available in the database that exists on Device 1 (*Storage Device*). As she needs to have access to the updated values in the database which will contain the latest available products, he first launches Activity 1 there. So she first sends a launch request (Original Intent) to launch Activity 1 on Device 1. In order to make the diagram simpler and easier to understand, we removed the preliminary steps to launch Activity 1 through LIQDROID that we have explained in more detail previously. As soon as Activity 1 has been launched on Device 1, it first will update Database 1. Updating the database also includes receiving the latest values from the Google cloud storage by LIQDROID (*Query Intent*), which we have explained in the previous section 6.7.2. Then Elena will receive an acknowledgement from LIQDROID on Device 1 that shows that Activity 1 has been launched properly there, which

means that the database is updated now. Elena clicks on the button "search results," which will send a Query Intent to the LIQDROID. The LIQDROID will send this request to LIQDROID on Device 1, which has the database, to perform the query method on its data on behalf of Activity 3. The received results by LIQDROID on Device 1 will be stored on the Google cloud storage. Whenever Elena clicks on the button "Show results", LIQDROID on Device 2 will download the results and Activity 3 will show them to her.

Besides giving the feasibility to have an ad-hoc database, this mechanism also enables the user to use the data as an input of the different activities (or services) available on the other devices. For example, the marketing manager of the shop is able to use another activity to analyze the data for making the next orders. So instead of Activity 3, there could be another activity which uses the updated data with the previous data that it stored to check how many products have been sold or to perform other types of statistics.

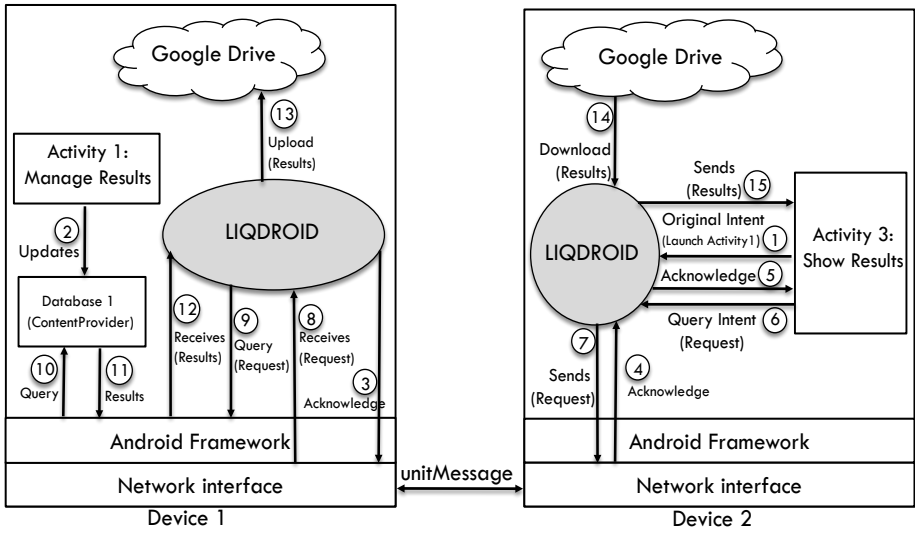


Figure 6.9: *Query unstructured data through* LIQDROID

The proposed use case scenario was only one of the potential examples of using LIQDROID to store or retrieve the unstructured data from an ad-hoc database. Another domain that can highly benefit from this feature is to work with the data provided by the sensors. Today we encounter more often than before that the user is able to have access to the data provided by different sensors available on different types of devices that he holds. These data values are mostly available on his smartphone which is connected to

those devices, but the other devices are not able to have access and use those data values. By using LIQDROID other proximal devices are also able to have access to those data values that have been stored on the user's smartphone without the need to directly connect to the sensor or install other applications to interact with the sensors. In order to save the data received from the sensor, the user needs to provide a database on a preferred device (such as his smartphone) and store the values based on the defined roles. The LIQDROID makes it possible for the proximal connected devices to have access to this database to insert values or make queries.

6.8 Users' perspective about LIQDROID

We have conducted an experiment including ten different users ranging from 18 to 65 years old with different levels of expertise in using new technologies and engaging in different occupations. The participants include a combination of expert users-both in the sense of being more familiar with the Android OS and those that have been following new technologies on smart devices-and also some participants that were older and only use their devices for performing simple tasks.

In the first step before introducing LIQDROID to the participants, we interviewed them through a questionnaire to acquire some preliminary information about their daily activities and their needs such as:

- what devices they have and what activities users carry out in their daily life?
- which devices they have access to at home, at work or other places?
- how often they exchange their devices (move to another device in their proximity) and why?
- whether they have used a combination of devices to pursue their needs, and if so how they did it and what their requirements were?
- whether they have encountered a situation where they wanted to continue a task on other devices, and if so which kinds of tasks, what the reason was for the device exchanges, and what they had to overcome to do it?
- how much information they have about the capabilities supported on their devices?
- how often they use the applications installed on their different proximal devices interchangeably in the process of executing a task?

The following table 6.1 shows the number and variety of devices that each user has. We have also asked them to include the devices that are not too old (preferably SDK version higher than 16) that they don't use currently. This would help us to have more included devices for the experiment also be able to evaluate the performance of LIQDROID considering different ranges of devices and technologies. While in the table 6.2 you can find the important reasons that makes participants prefer to have different types of devices. As you can see 70% of the participants prefer to have different devices for work and for personal usages because they think that they have more privacy in this way.

NO. of devices / Types of devices	None	One	Two
Android Smartphone	0	6	4
Android Tablet	2	5	3
Android Smart Watch	3	4	3
Android Auto	8	2	0
Android Television	7	3	0

Table 6.1: *Types and the number of Android devices that each participant has.*

After the participants completed the questionnaire, we have explained the above-mentioned scenarios to them, and asked them to propose their solutions for performing those scenarios. Based on their level of expertise in new technologies or their experiences, they proposed different solutions which mainly were focusing on transferring data while they needed to reconfigure the other device and resume their work manually by reaching that destination device. This trend was more common with participants that were older because they often were not aware of the existing applications in the Android market that could help them in better performing different

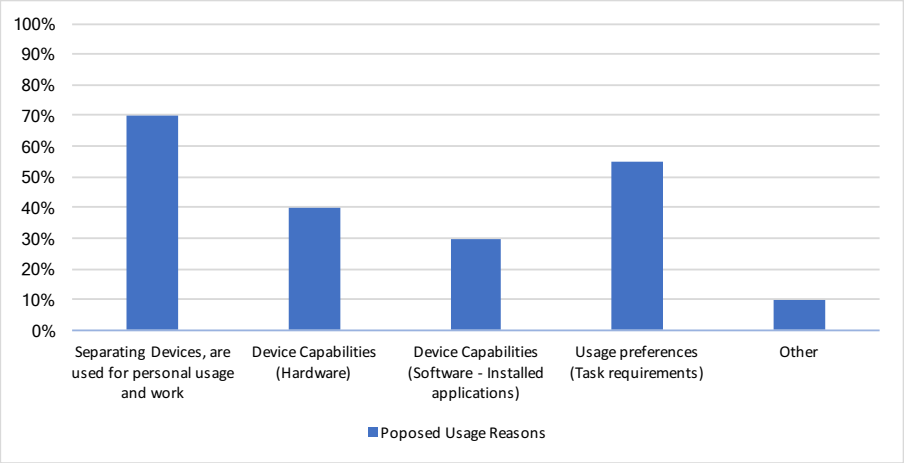


Table 6.2: *Reasons proposed for using multiple devices by the participants.*

activities. When we proposed some questions about the capabilities that their devices support, most of the participants (even young and expert users) did not remember how many different applications they had installed on their devices that could benefit them in the execution of their currently running tasks. We noticed also that 60% of their applications were commonly installed on all of the devices they owned. Based on the collected results, we have depicted the table 6.3, which shows how often they encountered the same situation and found the scenario practical. Here we need to mention that we did not consider the exact scenario but the features that were provided for users. For example, maybe none of the users worked in a shoe shop, but it happened more often that they needed to save their information in a database and wanted to apply some queries on its data later.

Then we have explained LIQDROID, its features and have installed it on their devices and assigned different scenarios to different participants. While later we also let them choose those that they were interested in, to have their own experience and provide their feedbacks. We used the results achieved from the activity logs, surveys, and interviews to evaluate different characteristics of LIQDROID. These characteristics include the user's acceptance, LIQDROID's performance at runtime, overhead and scalability. In the following, we have presented and reported the evaluation results achieved from the users' experiences and their opinions about LIQDROID while the other mentioned aspects along with a comprehensive analysis will be explained accordingly in the sections 6.10, 6.11 and 6.12.

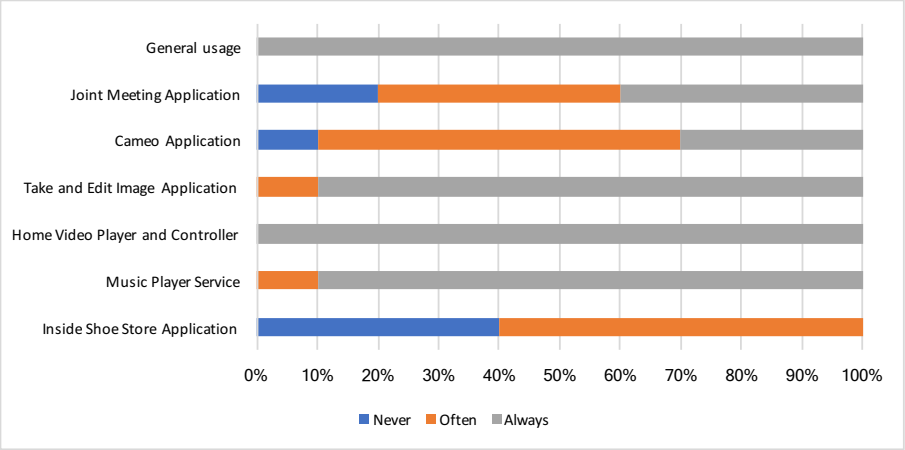


Table 6.3: *Levels of the participants' engagement with the proposed use case scenarios*

6.8.1 Discussion of the Results:

In the following table you can find the general opinions of the participants of using LIQDROID through considering different factors. As it shows they found LIQDROID very innovative and simple to use which are very important for us. Because in our point of view and based on our experiences, one of the important aspects that is needed to be considered in the design and usability of applications is that different users-even those who are not experts-should be able to easily conceive and be able to benefit from all the features that the proposed application is providing.

Although most of them commonly agreed about the simplicity in usage and effectiveness of LIQDROID and its innovation, their main concerns were related to privacy. They also pointed out their concerns by proposing some suggestions as follows:

- One of the suggestions received from the users was to provide the possibility to give external users access to a subset of applications instead of all of them. While it sound also interesting for use and we can add this feature to the next version of LIQDROID. In this case, we add a new feature to the setting menu of LIQDROID that will allow the user to provide a list of the applications that he is interested in giving the external users access to. So later in the process of providing the chooser list, LIQDROID will remove those items that the user is not interested in sharing with other members.Â The user will be able to change the list of these applications in the setting whenever his preferences change maybe according to the context of use, proximal

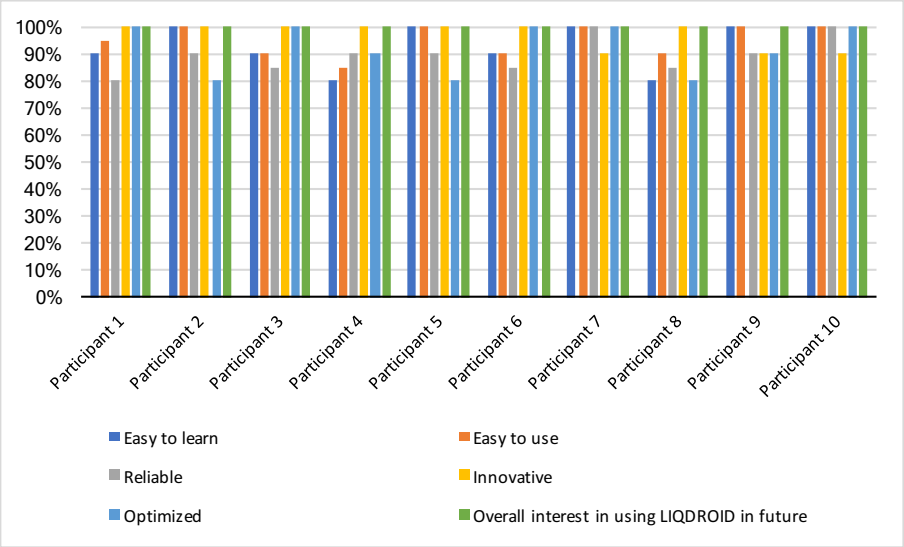


Table 6.4: *The participants' perspective about the usability of LIQDROID*

users or other factors.

- The suggestion was, giving priority to the instructions that are sent to LIQDROID. For example, the execution of a request a user sends to his other personal device has a higher priority than the request of an external user even if his request has been sent later. Although the current performance of LIQDROID is fast enough that will not make the user to experience long delays, but we can add another item in the setting that let the user register his personal devices. This will enable the user to also connect automatically to his personal devices when they are in his proximity without needing to send permission requests. On the other hand, we can benefit from this list of personal connected devices in the process of prioritizing the tasks in the Task Execution manager module and the event manager module.

We need to note that the optimization aspect considers both the performance and energy consumption of LIQDROID during the tasks executions, which highly depends on the device type and its battery state. For this experiment, we also asked the participants to bring the old devices that they don't typically use, and we integrated them in the experiments. On these devices, the energy consumption will certainly be higher than the newly released devices. This has little effect on how the performance of LIQDROID which has been conceived by the participants, while at the same time they referred

to it as a great aspect of LIQDROID. Because maybe you do not prefer to use your old smartphone for watching a video as it does not provide good quality, but still you can use it as a controller.

6.9 Developers' perspective about LIQDROID

Among the use case scenarios that were mentioned above-although, as we said, all the scenarios agreed with Telecom Italia-two of them, the Cameo app and the taken image, were developed by their Android developers. While the others have been developed by us. To better evaluate LIQDROID, we have benefited from external Android developers, ranging from experts to those that have recently started developing Android applications, who developed the four other scenarios that have been developed by us. To this end, first we gave them an explanation of each scenario as presented above, and then we asked them about the possible solutions and the required time that they can offer us to implement those features. We collected this information, and then we started explaining LIQDROID and how it works. After a short presentation, we showed them the document that we have prepared for the developers to develop LIQDROID-compatible applications and asked them to choose one of the above use case scenarios and implement them. We have excluded the implementations of the layouts for the user interfaces as they were able to benefit from our previously written codes. During the experiment, we collected their questions, which some of them are as follows:

- How can we guarantee that the other components will not receive our intents?
- Do we need to execute the intents? And if no how does LIQDROID handle the execution of different intents?
- How can different developers that have developed LIQDROID-compatible apps can collaborate with each other?

Although we have answered these questions in sections 5.5.3 and 5.6.2, we will provide brief answers here.

- We can guarantee this by providing unique names for the extra fields and encrypt the sensitive data. In this case, even if even if other applications filter and receive the intents, they are not able to extract and use their data.

- The developer should not start the intent because in this case the Android OS of the current device will manage it, while this procedure should be handled by the OS of the destination device(s) that the user has selected through the Chooser List. To prevent this situation, LIQDROID will transfer the original intent to the destination device and LIQDROID on the destination device is in charge of executing the intent properly. By properly we mean being able to adopting the different types of executing the intents [6] based on the developer's request.
- To answer this last question we have conducted an experiment to evaluate the developers collaborations that we explain it in detail in the following.

Besides the above-mentioned use case scenarios, we have also conducted a new scenario to better evaluate if there is the need to add other features to LIQDROID. The proposed application was an application for a restaurant to automate the procedure of receiving orders inside the restaurant.

In this experiment, in order to evaluate the possibility of using different LIQDROID-compatible apps that has been developed by different developers, we asked two different developers to design and propose the two LIQDROID-compatible applications. The two applications are: an application that will be used by the restaurant owners and another one that will be used by the customers. The Owner version is for sending the request the customers enter to the restaurant to receive their orders. For proposing these orders the customer will use the Customer version that is installed on their devices. The interaction between the restaurant Owner and the customers will be handled through these apps. Generally, the customers will be able to receive the menu of the restaurant, ask for extra information about the proposed foods, submit their orders, and receive their bills through the app. And on the other side the restaurant Owner is in charge of providing the responses to the customer's needs through his app.

6.9.1 Discussion of the Results:

We have evaluated their effort, and the time it took them to propose the solution using the provided features of LIQDROID. At each step, they considered and implemented different features and evaluated whether LIQDROID was capable of handling the cases. The analysis of their required effort and time for developing different features showed us that both expert and non-expert developers assigned the same effort and time. This proves the simplicity of

[6] For more information you can find the different types of executing an intent in the official documentation of Android:https://developer.android.com/guide/components/intents-filters.html

LIQDROID in learning and developing compatible apps. While proposing and discussing other possible and innovative use case scenarios and benefiting LIQDROID to handle the situation showed us that they are looking forward to exploring more by using LIQDROID in their upcoming projects.

Developers were interested in using LIQDROID as a shared infrastructure to benefit from the other existing applications installed on the proximal devices to pursue the running task's needs. Decreasing the effort that is needed to provide an application compatible with a device was one of the main points that they mentioned. In this case, they assumed that by benefiting from LIQDROID, they would not need to develop compatible applications for different devices while they will be able to use the third parties developed compatible applications. In this case, a developer can develop an app compatible with the smartphone while can benefit from another similar app that is compatible with the tablets to propose his desired features there.

The possibility of using LIQDROID in different fields and places (public and private) such as games, restaurants (as mentioned above), and train stations was one of the other motivations for them to use LIQDROID to distribute their applications' components and tasks.

During our final discussions with them, the part which they were most concerned about was the privacy concerns. They were also interested in having the features inside LIQDROID to enable the user to separate his personal devices from other proximal devices. In which they thought that this would highly improve the current privacy concerns not only regarding the data but also to the accessibility of the devices. So, for example, a user may want to discover his devices and execute a task on them (when is a bit far from them) while at the same time he is not interested in letting other proximal users discover their devices.

6.10 LIQDROID's Performance at runtime

To evaluate the performance of LIQDROID at runtime, we conducted an experiment to measure the time that is required to perform different steps of transferring and controlling a video player on proximal devices. The reason for choosing this task is that it includes most of LIQDROID provided features. To this end, we used a 3.61MB video and a group of four proximal devices. We measured the time between when LIQDROID receives the user's request, when the proper component is selected, and the launch of the component on the destination device. We also distinguished between the time spent on

the source and target devices. Class TimingLogger [7] provided the means to carry out the experiment. The different parts of this task that are handled by LIQDROID are as follows:

1. Preparing the Chooser List, which includes: the steps that have already been explained in section 5.5.3
2. Sharing the data, which includes: loading the video file from the storage of the source device, uploading it to the cloud and sending the acknowledgement (after the file has successfully uploaded) to the source component.
3. Sending the original intent to the destination device(s), which includes: adding the required fields which will be used by LIQDROID on the destination device(s) to manage the intent and serializing the intent to send it through the network.
4. Receiving the content on the destination device, which includes: deserializing the intent, downloading the file to the destination device, and saving the file on the storage of the destination device(s).
5. Executing the intent to launch the user desired component with the new address (URI) of the data on the destination device.

The table 6.5 shows the results for ten times execution of LIQDROID and its performances by considering different factors such as Network fluctuations, Multitasking, and Scalability. In the following we have briefly explained these factors that have influenced each experiment.

- In the first execution, we have followed the experiments with four devices while the users have only launched LIQDROID on their devices, and we have collected the log files provided by the TimingLogger along with the information of the CPU usage overhead of their devices.
- In the second execution, we kept the settings of the experiment the same as the first experiment, while the log files and the results, as seen in figure 6.5, show that network fluctuations occurred, which resulted in an increment in the time of preparing the chooser list and also in uploading and downloading the data.
- In the third execution, a disconnection occurred on one of the destination devices as it took a large amount of time from LIQDROID on the source device to prepare the Chooser List.

[7] https://developer.android.com/reference/android/util/TimingLogger.html

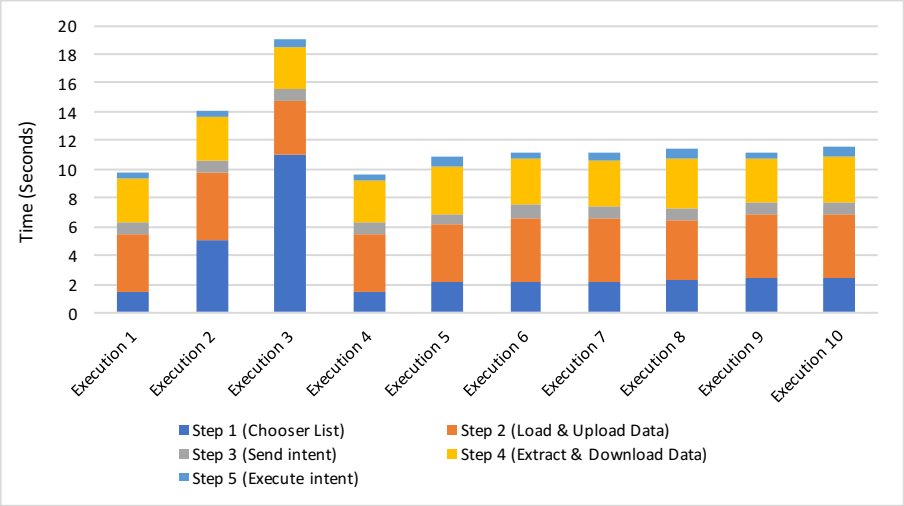


Table 6.5: *Performance of LIQDROID at run time (with sharing data)*

- In the fourth experiment, we had more or less the same results as the first experiment without any network fluctuations appearing on the log files.
- For the fifth experiment, we asked the participants to launch other services and activities on their devices. This increased the CPU overhead on their devices. In this case, the achieved results show little difference between the first and last steps as LIQDROID is used the Android OS to follow the processes.
- In the sixth experiment, we also launched other services and activities on the source device. The achieved results also show an increase in step 3 in addition to steps 1 and 5.
- The next four experiments concerned the scalability feature, which we integrated each time one more device to see how the scalability factor will affect the performance of LIQDROID. We need to mention that the CPU usage of the devices was the same as it was in experiment 6 on the source and destination devices because most of the time the participants left some launched application in the background of their devices, which is normal behavior for most users. The results have shown that the scalability just had a minor effect on the first step.

As we can see in the figure 6.5, the network interruptions had the biggest effect on the performance of LIQDROID. The reason is that we expected that

LIQDROID should wait until it receives all the responses (list of available components) from all the connected devices before proceeding to provide the chooser list. In this case, any network fluctuations that happen in the destination devices will have a direct effect on the performance of LIQDROID. In order to solve this problem, based on the above-mentioned experiment, we have considered a threshold time which is the maximum time LIQDROID needs to wait to receive the responses from the other device(s) before proceeding to provide the Chooser List.

On the other hand as a consideration for improving the performance time we can mention that if the developer uses the URL (for larger size data) or sends the data along with the Original intent (for small size data), it can highly improve the time of the performances. As you can see it also in the figure 6.6 which shows the subsequent effect of using URLs. We can also use this table as a presentation of the LIQDROID's performance in launching components (the existing components in the Android market or the LIQDROID-compatible apps) on the proximal devices.

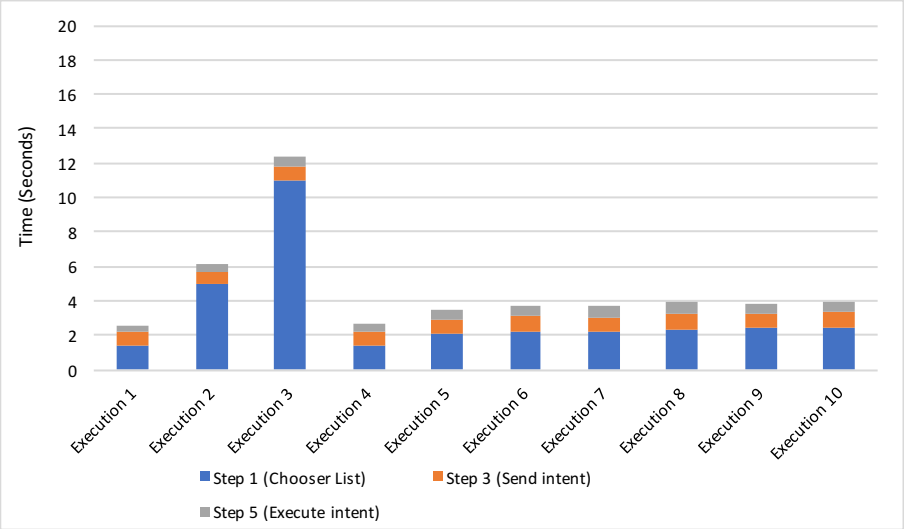


Table 6.6: *Performance of LIQDROID at run time (general usage)*

6.11 LIQDROID's Overhead of energy consumption at runtime

During the design and implementation of LIQDROID, we have considered the energy consumption as a great deal that we mention to them to explain

the overhead that the execution of LIQDROID at runtime when it receives a request or even its long execution in the background can cause in the device.

Recent empirical studies on the energy consumption of the Android application [43] have pointed out that most of the applications use 61% of their energy in the idle states when the user is not interacting directly with the applications. As LIQDROID is a service that is running in the background and as it does not occupy the user interface, its energy consumption even in long executions is much lower than activities. As well, to lessen energy consumption during the idle time, we have determined that all the executions will start when the user is interested in distributing a task. On the other hand, instead of bounding activities which are using a lot of energy, we have used pop-ups to receive user's preferences as well as to notify him and provide relevant information. All these pop-ups have a threshold time, and if the user does not interact with it during the defined time slot, it will be closed automatically.

Generally, we can consider three different phases of energy consumption for LIQDROID at runtime as follows:

- Discovery and Advertisement, which is handled by the Google Nearby API. - Although devices will continuously advertise themselves, the discovery will happen manually under the user's direct intention, and this has highly improved the energy consumption of LIQDROID. The reason is that, as we have already mentioned, Google Nearby uses a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and an ultrasonic modem to detect the proximity of the devices, which causes a lot of energy consumption. So we have restricted this energy consumption only to the time that the user desires to distribute a task. On the other hand, as Google has already started to improve the energy consumption of this API by providing the possibility to choose only a particular protocol to detect the proximity of the devices such as Bluetooth Low Energy, which consumes much less energy, we expect that we will be able to even improve more the energy consumption of this step.

- Each LIQDROID process, which is handled by the Intent Manager Module: this process starts when the user launches a LIQDROID-compatible component to send a request to LIQDROID and continues up to the time that the user terminates it. In the process of implementing LIQDROID, we have used two different methods. The first category of the methods were based on implementing compatible methods that can benefit the most from Android OS to provide required features. The modules that these methods have been used on their implementations

are: the Task Execution Manager Module for the Chooser List as well as the intent queue for managing the requests and also the Event Manager Module. This resulted in two benefits: first, we are sure that the execution of these procedures are optimized and, second, we can benefit from the parallel execution of several Android OS to handle different LIQDROID provided features.For example, in the process of providing the Chooser List while four devices are integrated, instead of implementing our own mechanism to collect the list of installed applications on all the devices, finding the capable components and then proposing the Chooser List on the source device, we used an existing mechanism inside the Android OS. In this case, the Android OS of all the integrated devices will work simultaneously and provide the list of their capable components for LIQDROID on the source device. While for optimizing the rest of the methods that have implemented by us, we benefited from Traceview [8] to profile their performances and improve them in sense of decreasing the resource usages and energy consumptions.

- network connection and managing the artifacts through the Firebase API - The above-mentioned study [43] also shows that the network and particularly making an HTTP request is the most energy consuming component in Android applications. Although benefiting from the Firebase storage allowed us to manage the artifacts better and give the user the possibility to postpone the procedure of uploading and downloading the data to a later time, it consumes a lot of energy. So by distributing the task to the proximal devices, especially those that are connected to a power source, we have been able to highly improve the energy consumption of LIQDROID at runtime on a device. This also is very helpful in improving the performance of the wearable devices because of their limited resources and battery capacity.

In general, in the conducted experiment that have mentioned above (in section 6.10) which in the worst case that has used the longest time of the execution (about 19 seconds) on the source device for sharing a video which also includes all the above mentioned three steps, LIQDROID has consumed around 2% of the battery of the source device. More in detail, LIQDROID has used around 26% of the overall energy consumed during this time in the discovery and advertisement, 11% for the processes managed by LIQDROID and 63% for the network connection (including the network fluctuation) and also managing the artifacts. In the other experiment, where instead of

[8] https://developer.android.com/studio/profile/traceview.html

transferring the video we used a URL for that video, we were able to save up to 56% of this overhead in energy consumption on the source device.

6.12 LIQDROID's Scalability

LIQDROID is capable of receiving several requests at the same time even from different components on a single device or from multiple proximal devices. In order to handle multiple requests simultaneously, LIQDROID creates a new thread for each request and runs them immediately instead of waiting for the previous request to finish. So in this case while it is providing the Chooser List, if the user launches another component (on the same device) and sends a request for receiving another Chooser List, these two requests will be processed by LIQDROID separately. Based on the available components (along with their application's icon and their name) in the list, the user is able to recognize which Chooser List belongs to which task and can choose the appropriate component and distribute the task.

On the other hand, in crowded places such as train stations, restaurants, and museums, it may happen that a device receives several requests from the other devices when it is free or while it is executing a currently distributed task. To better evaluate the consistency that may happen in this situation, we have conducted an experiment with 8 Android devices ranging from tablets to phones to monitor the performance of LIQDROID. One of the main characteristics of these places is that many short access and interactions may happen that should be properly managed. During this experiment, the Communication and Event Manager Modules ware playing the key roles. The Communication Module made users capable of restricting the access of the unwanted users to their devices as well as preventing them to run tasks while they were using their devices or were interacting with an already distributed task. On the other hand, the Event Manager helped them to select devices that are not currently busy. Killing the running activities on the destination devices that their users were not interested in anymore was helpful in the sense that it frees up resources and saves their energy while other users were able to distribute a new task on them.

Although LIQDROID is capable of connecting all these devices together and making them collaborate, users mostly prefer to only connect and interact with a subset of them. Their reason was that they think that in this way they are better able to manage the situation even from the privacy point of view, and they can also concentrate better on the task at hand. Because, as we said, LIQDROID will provide the list of all capable components on a device, and this multiplied by the number of devices will provide a long

Chooser List, some of the users find it difficult to choose the component they prefer.

6.13 Other Possible Domains of Use and Comparisons

In the previous section we have discussed the use case scenarios that have been implemented by us to evaluate different features of LIQDROID. Letting different users interact with these applications and discuss their experiences and feedback could even better help us to improve LIQDROID, but the time limitation did not permit us to cover this step. There are also other domains where LIQDROID can be useful, but as we did not have enough time, we could not touch on them. While reviewing the existing middlewares and frameworks, we have also encountered use case scenarios that have been used for evaluation purposes which we have also considered in the process of designing LIQDROID. Here we provide comparisons between LIQDROID and these works and will explain how LIQDROID is capable of proposing the same features.

6.13.1 Multiplayer Games

One of the interesting domains in which LIQDROID can enhance interactions is multi-player games where we can find multiple users with multiple devices that need to collaborate with each other to reach a final goal. What is more important in this domain is allowing several devices to be able to fluidly interact with each other and also keep their states synched.

In the [20] they have provided a Crossword Game. The users subscribe to the session by using their mobile devices and are able to participate in the game and enter the words. The user entered word will be shown on all the devices. The LIQDROID is able to support the same features as follows: the first member is able to create a group and connect to the other proximal devices that he wants to play with. He will advertise the group and the other participants will receive his connection request. If they are interested to play, they will accept it. The user starts the game component on his device and requests LIQDROID to synchronously launch the component on all the members of the "Crossword Game" group. The users of the other connected devices also apply the discovery and find the game's group and they connect to its members. Now everything is set and they are able to provide the words. Whenever a player provides a word, an Update Intent will be sent to all the other connected devices. The state of the game component will be updated on them and the word will become visible there.

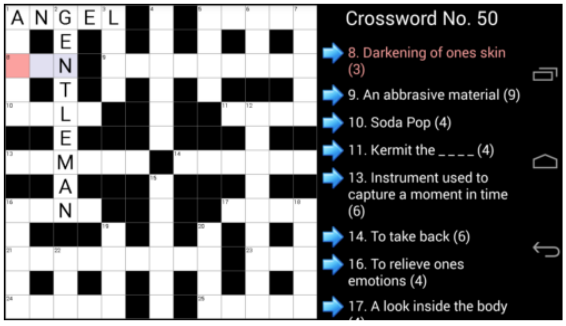


Figure 6.10: *Multi-player Games Sample - Crossword Game. [20]*

If a new member wants to join the game, after the connection phase the game initializer is able to resume the game component with its last state on the newly joined member, and he can see the game component with all the words that the other players have already predicted. On the other hand, he is able to discard the connection requests that want to use his device to distribute the execution of another task.

6.13.2 Video Streaming Application

In the [80] they have provided a YouTube browser which includes four *panels*. By their definition, a *panel* is an XML element, which will be used to categorize the controls of the multiple-device interactions. These four *panels* are defined in their use case scenarios as follows:

- A video stream panel
- A playback control panel
- A search panel and
- A related videos panel

The process of assigning each of these items to a *panel* is based on taking advantage of the Panelrama's automatic panel allocation feature. To this end, the developer needs to define the strong capability required by each panel at the implementation time, such as the video needs a big screen size or mobility is required by the controller. At runtime, the Panelrama will be capable of assigning these panels to the user's device.

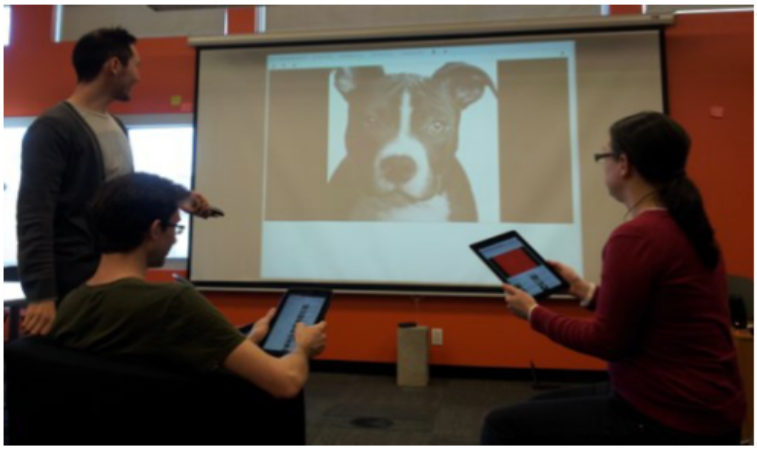


Figure 6.11: *Shared YouTube browser [80]*

The proposed scenario can easily be handled by LIQDROID as the user is able to choose the proper component to be launched on each of the proximal connected devices. He is free to choose any kind of device to assign the task to. For example, he may prefer to open the video on a tablet which is closer to him and send the suggested videos to the big screen to be able to further review the numbers of the videos in a single page. Meanwhile, he is able to use the controller on his smartphone which supports more mobility, and he is able to control the execution of the video on the tablet while also selecting the next video among the list of suggested videos on the big screen. The user can select the big screen to play the next suggested video or can select the tablet to play the video while the required information or content of the video as well will be sent there.

Having the possibility to launch different components on different devices based on user preferences at runtime is one of the very important features provided by LIQDROID. Because, as we mentioned in the section 2.1, there could be different factors which can affect the interaction between the multiple-devices such as the availability of the devices, the users or the context of use. So considering the device features is one of the important factors in assigning a task to a device, the other factors such as the availability, accessibility, and other existing constraints related to the user's preferences are the things that should be supported by the runtime. If the developer wants to consider all of them at the implementation time, it can be very challenging. By increasing the number of devices, the users and the context's features, the complexity will be much wider than he would be able to overcome.

6.13.3 Distributed PDF Reader

In the [80] several tablets have been used to show the consequence pages of a PDF file and that any added device will show the next page of the PDF file. We also had this same scenario in the Joint Meeting Application. One of the nice features that they also added to their framework is that the user can lock the screen of a device to a certain page. This will assign the page permanently to a device, which will be useful when the page includes a figures or references list that the user may need to focus more on.

To support this feature by using the LIQDROID, we can benefit from the possibility of the group formation provided by LIQDROID. These groups will be used in the process of distributing the execution of a task. If in the middle of an interaction a user wants to remove a device from the ongoing interaction, they are able to reach the members of the group that the devices belong to, and by applying a long click on the device name, he is able to choose to remove that device. This means that the devices will become disconnected and the state of that device will not be updated anymore until the user connects to it again. Here the user can also temporarily disconnect the devices so the Update Intent which will change the current page of the PDF will not be received by that device.

Excluding a device from a group list will also be useful when something such as a low battery event from one of the destination devices notifies the source device that in any moment the device will not be available. In this case, he can resume the task on the other proximal connected device before losing that device and also the results of the task.

6.13.4 City Guide

Another Android application that has been developed by the [20] for the tour leaders and tourists. The version which is under the control of the tour leader has some extra features that give him the possibility to enable or disable them on the tourist's version. The proposed feature is achieved in their framework through providing certificates that depend on the user's role, and it enables the leader to apply different rights in the environment. On the other hand, the leader will have a part which includes different images, and he can select among the available images to be shown to the tourists. But this feature is hidden in the tourist's version.

LIQDROID is also capable of supporting the role feature as we explained in the section 5.5.1, and this feature is able to control the execution of a distributed task on the proximal connected devices. This control mechanism can appear the same here in the shape of enabling or disabling some of the

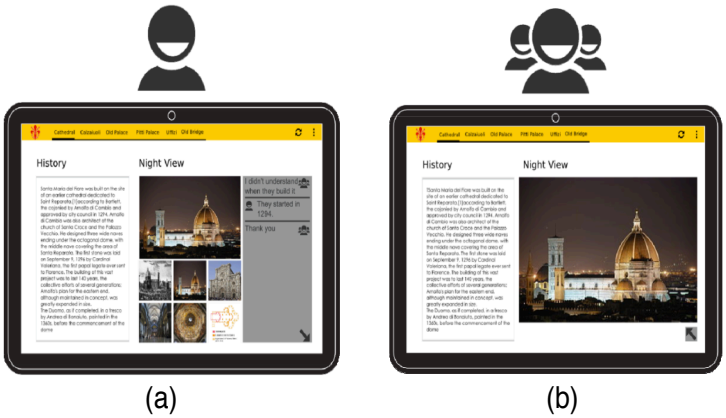


Figure 6.12: *City Guide Sample - (a) is the guide version with all the features visible and (b) is the tourist version with the hidden features. [20]*

available features on the same components that are executed on the proximal device(s) by using the Update Intent.

This is also another interesting domain to apply the multi-devices interaction and uses of LIQDROID because there is a broad range of applications available that benefit from the user's current location to provide some suggestions for the tourists to guide them to make better use of their time while also better promoting the place's attractions, for example, museums, restaurants, or ongoing events that are close to the tourist's current location. So the user of these applications is able to open each of these categories of the provided suggestions on his own proximal devices, or he can send them to his friends' devices to let them review them. This will highly improve the collaboration between several proximal people while significantly decreasing the time that is required to reach a conclusion between them on where the next destination will be.

6.13.5 Museum Guide

One of the widely used sample scenarios in the field of multi-device interaction is related to the Museum (or exhibition) Guides. A broad set of works [41, 57] has been developed to improve museum visits when there is no kiosk or responsible people available to help the visitors. Some of these works such as [13, 68, 73] also have benefited from games to enhance the cooperation among museum visitors and make the museum visits more attractive.

A museum visit could happen as an individual experience where the

user can have the possibility of having access to the information and the resources (devices) that are provided to help the visitors. Or it can be a group experience with a leader that will provide the visitors with extra information and materials besides those available in the museum.

In this context, the multi-devices interaction can extremely enrich the visitors' interaction with the leader and also the provided resources inside the museums. To this end, by benefiting from the infrastructure that LIQDROID provides to integrate the devices and distribute the execution of the tasks among them, the available systems and applications will be able to provide wider features and more organized services for the user instead of overloading them with information. For example, when the user is near to a big screen assigned to each art piece, they are able to distribute and control the execution of a task on that so that other visitors can also benefit from the provided information. This will enable the visitors to have a unique experience of active and relevant discussions to share their information and resources during their visits, which will end up being an overall interesting experience where they will learn so many new things.

6.14 Conclusion

In this chapter, we aimed to provide some usage scenarios to better test and improve LIQDROID provided features on real devices and in real contexts. In general, the proposed comprehensive infrastructure helps the user to better identify the resources (proximal devices) that are available in his proximity to easily benefit from them in the execution of his current task. LIQDROID can bring for the user(s) a fully cooperative interaction between the multiple devices that can help him to perform his desired task in much less time and with higher productivity. This productivity will bring better-achieved results and also extra available time for him to assign to other tasks.

The proposed infrastructure can also conceivably improve the developer's efforts as they can easily benefit fromLIQDROID without the need to put forth extra effort in developing the requirements of these cooperative interactions inside their applications. Instead, they can put their focus on how they can acquire more from this new interaction paradigm to develop innovative use case scenarios to facilitate their users' work as well as provide a unique experience for them to attract more attention.

Besides the fact that implementing these steps separately inside of each particular application requires much more time and effort, it can also decrease the performance of the available resources and have an adverse effect on the final results. For example, consider that several users at the same

time compete to distribute a task on the same proximal device. As each of them first needs to run a separate service on that destination device to be able to start the interaction, this will form two immediate problems. First is the interruption that this parallel usage can cause on the execution of the ongoing tasks. And second is that the execution of several services at the same time on that destination device to answer the request of their users, without the presence of a central unit to manage them, will highly decrease the performance of that destination device.

During the evaluation of LIQDROID on the real devices, we encountered some problems which made us review the architecture of LIQDROID and apply some changes on it. We mention just some of these problems here such as preventing the interruption that receiving the permission requests can cause while a user is doing critical tasks and is not interested in any collaboration. This made us think about providing a separate phase that will allow the user to be able to completely come out of the circle of these proximal devices by deactivating the advertisement phase. On the other hand, the Wi-Fi instability and instant disconnections while LIQDROID was collecting results sometimes caused an inconsistency in the system that we needed to handle better by monitoring the connection status.

These evaluations also reminded us that the developer may also need to have access to some of the information which is available on LIQDROID such as the list of connected devices and the occurrences of the events that happen in a device, for example when the screen goes off and on, or receiving an acknowledgement that shows the execution of the distributed task(s) or storing the achieved results that have been handled successfully through LIQDROID, etc.

The other concern which we always had in our mind during the evaluation phase as well as the architecture design and also the implementation was related to the changes that happened inside the Android framework and the updated versions of it that sometimes put some serious restrictions on the methods of accessing some of the previously available features. For example, accessing the Activity Manager in the previous versions was possible for us, while in the newly released updates because of some privacy concerns the Android framework has decided to restrict the access of third party applications to this feature. This had a serious effect on our work, but we solved it in another way as we have already mentioned in the chapter 5.

In general, we can say that working with LIQDROID will be very interesting and easy for the users and developers as a new paradigm of interaction between multiple devices. It introduces a beneficial way to explore the potential that already exists on the devices, whereas with the current so-

lutions on the multi-device interactions we do not properly benefit from them. And by introducing LIQDROID, we have implemented our idea about the possibility of **direct** interaction between the multiple proximal Android devices to distribute the execution of a task and also to attract the attention of other researchers to this kind of cooperation between the devices.

CHAPTER 7

Conclusion and Future Directions

In this work, we have proposed a middleware infrastructure that enables the user to seamlessly distribute the execution of a task between a set of proximal Android devices. The proposed solution will provide the required infrastructure to make the multiple-devices directly interact and cooperate with each other to achieve better results, or results that without this integration were not possible to achieve.

7.1 Answers to Research Questions

7.1.1 Research Question One [Q.1]

The question: What are the currently available solutions (methods and technologies) capable of supporting direct interaction between multiple proximal devices? (Chapter 2)

The answer: By reviewing the current available works which enable multiple proximal devices to interact with each other, we can mention the following important aspects and issues that still exists in the available solutions:

- The current solutions mostly focus on cross-platform aspect in order

to meet the needs of a wide range of devices. However, the solutions proposed are still affected by responding to some of the challenges that already exist in this field, such as data management and also synchronization between these devices. In addition, this generalization as regards the consideration of the type of devices will prevent us from proposing a solution that can benefit from the competitive features and potential that exist in each of these devices and platforms. So we ended up providing a solution, applicable for all the devices but is limited in the features that can provide for the final user.

- The other category of solutions tried to put their main focus on increasing the usability of the applications installed on this mobile technology enabled devices. But still the proposed models to design more usable application and the applications that have been developed based on them are not satisfactory. This is understandable, considering the limitations that each of these mobile devices put in the way of designing these applications, as well as the obstacles and restrictions that developers may face.

- Recent advancements in connection protocols and data storage technologies have also improved some degree of task continuity and state synchronisation between multiple devices. But the problem that exists with this category of solutions is that they are vendors' dependent. So it is possible to use them only through a set of specific devices or applications.

  Moreover, in order to obtain some level of privacy, they ask the user to provide credentials in advance, which restricts the user to only be able to use those devices that belong to him or that have his proper credentials already entered on them. The other approach is benefiting from a third external device in between to enable different devices to interact with each other. Besides the high price and the problem of portability regarding these devices, the user needs some level of expertise, as well to assign a lot of time to the configuration.

Despite the high attention that multiple device interactions have recently attracted and the advances that have happened in managing the interaction between devices, common usage trends still show that they were not successful enough to be adopted by users and provide their satisfaction. This led us to provide a novel paradigm which can enable multiple devices to directly interact with each other. The proposed solution will help users benefit from the maximum potential that exists in the devices during the execution of an

ongoing task.

Because of the potential that exists on the Android platform, we have based our solution on the Android platform. While there exist solutions that have been used on this platform, they are limited, which means that there still exist unsolved challenges in this field. The presented solution will solve some of these challenges, such as task distribution on multiple proximal Android devices, and managing the state synchronization, and also the data availability during these interactions.

In addition, we were also concerned about considering other factors such as multiple users, context of use and the input/output modalities that can have an effect on these multiple device interactions. This novel paradigm will help us to better exploit the potential capabilities and features available on these devices and change the state of single-user single-device usages to multiple-user multiple-device scenarios.

7.1.2 Research Question Two [Q.2]

The question: What do we need to consider to enable multiple proximal devices to directly interact with each other? This required answering the following sub-questions: (Chapters 3 and 4)

- What are the requirements of the final users, and how do we manage them to allow the users to seamlessly distribute the execution of tasks?
- What are the requirements of the developers, and how do we manage them to allow the developers to seamlessly distribute the execution of the components of their applications?
- What would be the immediate outcomes of the proposed solution?

The answer: In order to support the direct interaction between multiple devices and make them become cooperative, we need to support the following steps. The first step is having enough information about the available devices and their capabilities. The second step is having enough information about the execution of the distributed task, its requirements and the users' preferences. The last step is the ability to integrate the devices, properly distribute the task between them, and manage its execution and the artifacts achieved.

To this end, we proposed LIQDROID, which is a middleware infrastructure that lets a set of proximal Android devices become fully cooperative and create a bigger Android ecosystem. LIQDROID has a modular architecture and will benefit from some of the available potentials inside the

Android framework to provide the required services for running the application. Moreover, this modular architecture makes it easier for future changes and improvements. We did not restrict ourselves to the currently available features on the devices or in the Android framework, so it is easy to add and use upcoming APIs, devices and also technologies that support the Android OS for the currently developed middleware infrastructure.

The proposed solution will provide an infrastructure to integrate multiple devices and make them be fully cooperative. This will enhance the user's experience by providing him with the appropriate structure to better perform his desired task, through benefiting from the maximum potential and capabilities that the underlying context (including the devices or users available in it) can offer him in order to achieve better results. At the same time, it will decrease the effort required to change the devices and configure them, so he can assign his time to more important tasks and become more productive.

7.1.3 Research Question Three [Q.3]

The question: What are the technical aspects considered in the implementation of the middleware, and how have we addressed the possible barriers and inconsistencies that we were faced with in the process? (Chapter 5)

The answer: The Android framework has provided a complete and organized definition about the available features and how it is possible to benefit from the existing capabilities and technologies. But since we have said LIQDROID is an Android service, as a third-party application, privacy concerns sometimes put obstacles in our way to access some features and to implement our ideas. For example, although Android keeps track of the components running on the device, this information is not available for the third-party components. So we have obeyed all the concerns that Android had already provided, and in the case where the feature was completely safe, we have asked the application developer to provide the required data, such as the state of the activity's lifecycle for LIQDROID.

Finally, we need to mention that the continuous execution of LIQDROID in the background will not cause any interruption to the normal execution of the other applications' components on the device or affect the user's focus. The user has the possibility to control its execution (activate/deactivate) easily.

One of the main problems with some of the available middleware technologies to support multiple device interactions is that the developer needs to put in too much effort to learn how he can benefit from them in his application or make them compatible. In LIQDROID, all the complexities are on

the side of the middleware which this will facilitate the developer's work, so he does not need to learn new things to be able to benefit from the features that LIQDROID provides, and his current knowledge in developing Android applications is enough to be able to develop LIQDROID-compatible applications. By providing a unique infrastructure that all the installed applications on a device can benefit from it (even in parallel), developers do not need to make an extra attempt to separately manage these aspects on their individual applications to be able to distribute their execution.

7.1.4 Research Question Four [Q.4]

The question: How will the proposed solution validate itself in the real case situation? This is more specifically answered by the following sub-questions: (Chapter 6)

- How can it improve the process of developing distributed applications in both aspects of novelty and required attempt?
- What is the opinion of final users about it while they interact with it?
- What is the opinion of developers about it in the sense of usability, and how much can it improve their attempts to develop distributed apps?
- What is the overhead of it in terms of performance and energy consumption at runtime?

The answer: Although LIQDROID is capable of benefiting from the existing applications' components in the Android market in the process of distributing the execution of a task, we have also developed some LIQDROID-compatible applications with the purpose of testing different features of LIQDROID. These real use case scenarios have been defined by the collaboration of the Telecom researchers and us, and will be used later by them in their research laboratory. The proposed LIQDROID-compatible applications have been developed and tested on real devices, and the issues that were encountered have been solved. The comparison between the time assigned to developing the compatible applications and the results achieved during the interaction with LIQDROID were satisfying and interesting. We think that the potential that exists in LIQDROID in managing users' requests and handling interactions can make it a good candidate to attract developers and users to use it.

7.2 Future Directions

As one of our primary focuses was on the seamless interaction between devices, this will cause less user awareness about what is happening on the devices, and LIQDROID needs to entirely monitor the interactions in order to propose safer decisions. So privacy concerns are still an open issue in LIQDROID that should be thoroughly considered and evaluated. One of the possible directions is asking for permission on the destination devices when the distributed task wants to have access to the resources (such as camera, microphone, sensors, etc.) of the device. The other way to provide a secure access to the applications and their components is what has already been suggested by the participants of the experiment: to provide a subset of the user-permitted applications to share between other proximal users. In this case, for example, if the user does not give access to the Gallery application, other users that will connect to his device will not see the Gallery application and its components on the chooser list. So they will not be able to launch it or gain access to its data. This feature can be added in the setting module of LIQDROID so that the user will be able to change the list of permitted applications at any point of time.

The other aspect that we can consider about the privacy is letting the user have a list of his personal devices so that these devices will have a higher level of priority in distributing the tasks on them than those devices that have temporarily paired.

The other concern is related to the data replication mechanism that exists in LIQDROID. The complexity of the data replication mechanism directly depends on the usage of LIQDROID in the scenarios where several destinations devices are working in parallel on the same data. In this case, we need to handle the conflicts and data consistency more specifically. We have already provided a mechanism to store the different versions of the data and manage the access of the user to them. But the other aspect that needs to also be considered is how long it is necessary to keep these different versions of the same data and manage their replication or elimination based on the component's logic or the user's preferences. For example, if the user distributed a task on three destination devices to edit the same image on them, these different versions of the same image will be stored by LIQDROID, but the user may only select one of them and proceed to use it while it is not required to keep the two other versions. In this it is necessary to receive the user's preference and, based on that, go on to either keep them or delete the other versions. We have already made the developer capable of controlling the data replication based on his needs, but by providing other

specific rules and constraints, we can improve the mechanism of the data replication and storage management supported by LIQDROID. These rules can address concerns about the data privacy (which components and who can have access to this data), version controls or the time which they may need to be stored in the storage of LIQDROID.

In addition, managing the data streaming feature can also be considered as another direction to improve the data availability and transfer between the devices. This will enable LIQDROID to transfer only the pieces of data which are required by the distributed task instead of transferring and loading all the data at once in the destination device(s) and then starting the execution of the distributed task. For example, the user would be able to send the data related to the last five minutes of a video instead of sending all the content of the video, which is around 30 minutes. Data streaming will greatly improve the time needed to distribute the execution of tasks that contain larger amounts of data.

Bibliography

[1] AllSeen Alliance. Alljoyn - a peer-to-peer software development framework for ad-hoc proximity based d2d communication., 2013.

[2] Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong. Rio: a system solution for sharing i/o between mobile systems. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 259–272. ACM, 2014.

[3] Wolfgang Appelt. Www based collaboration with the bscw system. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 66–78. Springer, 1999.

[4] Till Ballendat, Nicolai Marquardt, and Saul Greenberg. Proxemic interaction: designing for a proximity and orientation-aware environment. In *ACM International Conference on Interactive Tabletops and Surfaces*, pages 121–130. ACM, 2010.

[5] Magnus Bång, Anders Larsson, Erik Berglund, and Henrik Eriksson. Distributed user interfaces for clinical ubiquitous computing applications. *International journal of medical informatics*, 74(7):545–551, 2005.

[6] Daniela Bourges-Waldegg, Yann Duponchel, Marcel Graf, and Michael Moser. The fluid computing middleware: bringing application fluidity to the mobile internet. In *The 2005 Symposium on Applications and the Internet*, pages 54–63. IEEE, 2005.

[7] Gerardo Canfora and Fabio Melillo. Sip2share-a middleware for mobile peer-to-peer computing. *ICSOFT*, 12:445–450, 2012.

[8] Xiang'Anthony' Chen, Tovi Grossman, Daniel J Wigdor, and George Fitzmaurice. Duet: exploring joint interactions on a smart phone and a smart watch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 159–168. ACM, 2014.

[9] Pei-Yu Peggy Chi and Yang Li. Weave: Scripting cross-device wearable interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3923–3932. ACM, 2015.

[10] David Dearman and Jeffery S Pierce. It's on my other computer!: computing with multiple devices. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 767–776. ACM, 2008.

Bibliography

[11] Alexandre Demeure, Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, Vincent Ganneau, and Jean Vanderdonckt. The 4c reference model for distributed user interfaces. In *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*, pages 61–69. IEEE, 2008.

[12] Charles Denis and Laurent Karsenty. Inter-usability of multi-device systems–a conceptual framework. *Multiple user interfaces: Cross-platform applications and context-aware interfaces*, pages 373–385, 2004.

[13] Riccardo Dini, Fabio Paternò, and Carmen Santoro. An environment to support multi-user interaction and cooperation for improving museum visits through games. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 515–521. ACM, 2007.

[14] Tao Dong, Elizabeth F Churchill, and Jeffrey Nichols. Understanding the challenges of designing and developing multi-device experiences. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 62–72. ACM, 2016.

[15] Daniel J Dubois, Yosuke Bando, Konosuke Watanabe, and Henry Holtzman. Shair: Extensible middleware for mobile peer-to-peer resource sharing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 687–690. ACM, 2013.

[16] Niklas Elmqvist. Distributed user interfaces: State of the art. In *Distributed User Interfaces*, pages 1–12. Springer, 2011.

[17] Lucia VL Filgueiras, Danilo O Correa, Joao S Oliveira Neto, and Renato P Facis. X-gov planning: how to apply cross media to government services. In *Digital Society, 2008 Second International Conference on the*, pages 140–145. IEEE, 2008.

[18] George W Fitzmaurice, Azam Khan, William Buxton, Gordon Kurtenbach, and Ravin Balakrishnan. Sentient data access via a diverse society of devices. *Queue*, 1(8):52–62, 2003.

[19] Murielle Florins and Jean Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *IUI*, volume 4, pages 140–147, 2004.

[20] Luca Frosini and Fabio Paternò. User interface distribution in multi-device and multi-user environments with dynamically migrating engines. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*, pages 55–64. ACM, 2014.

[21] Andrea Gallidabino and Cesare Pautasso. Deploying stateful web components on multiple devices with liquid. js for polymer. In *Component-Based Software Engineering (CBSE), 2016 19th International ACM SIGSOFT Symposium on*, pages 85–90. IEEE, 2016.

[22] Andrea Gallidabino, Cesare Pautasso, V Ilvonen, T Mikkonen, K Systä, JP Voutilainen, and A Taivalsaari. Architecting liquid software. *Journal of Web Engineering*, 16(5&6):433–470, 2017.

[23] Giuseppe Ghiani, Fabio Paternò, and Carmen Santoro. On-demand cross-device interface components migration. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 299–308. ACM, 2010.

[24] Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. Shared substance: developing flexible multi-surface applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3383–3392. ACM, 2011.

[25] Dagmawi L Gobena, Gonçalo NP Amador, Abel JP Gomes, and Dejene Ejigu. Delegation theory in the design of cross-platform user interfaces. In *International Conference on Human-Computer Interaction*, pages 519–530. Springer, 2015.

[26] G. Gruman. Welcome to the next tech revolution: Liquid computing., 2014.

[27] Peter Hamilton and Daniel J Wigdor. Conductor: enabling and understanding cross-device interaction. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2773–2782. ACM, 2014.

[28] Rachel Harrison, Derek Flood, and David Duce. Usability of mobile applications: literature review and rationale for a new usability model. *Journal of Interaction Science*, 1(1):1, 2013.

[29] John Hartman, Udi Manber, Larry Peterson, and Todd Proebsting. Liquid software: A new paradigm for networked systems. Technical report, Technical Report 96, 1996.

[30] Björn Hartmann, Michel Beaudouin-Lafon, and Wendy E Mackay. Hydrascope: creating multi-surface meta-applications through view synchronization and input multiplexing. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays*, pages 43–48. ACM, 2013.

[31] Ken Hinckley, Robert JK Jacob, Colin Ware, Jacob O Wobbrock, and Daniel Wigdor. Input/output devices and interaction techniques., 2014.

[32] Steven Houben, Paolo Tell, and Jakob E Bardram. Activityspace: managing device ecologies in an activity-centric configuration space. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, pages 119–128. ACM, 2014.

[33] Azham Bin Hussain, Sharaf Aldeen Abdulkadhum Abbas, Mustafa Sabah Abdulwaheed, Rammah Ghanim Mohammed, and Adil abdullah Abdulhussein. Usability evaluation of mobile game applications: A systematic review. *environment*, 2:5, 2015.

[34] Apple Inc. Handoff, 2014.

[35] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. Pointright: experience with flexible input redirection in interactive workspaces. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 227–234. ACM, 2002.

[36] Tero Jokela, Jarno Ojala, and Thomas Olsson. A diary study on combining multiple information devices in everyday activities and tasks. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3903–3912. ACM, 2015.

[37] HeeEun Kang, Kihyun Jeong, Kwonyong Lee, Sungyong Park, and Youngjae Kim. Android rmi: a user-level remote method invocation mechanism between android devices. *The Journal of Supercomputing*, 72(7):2471–2487, 2016.

[38] Amy Karlson, Brian Meyers, Andy Jacobs, Paul Johns, and Shaun Kane. Working overtime: Patterns of smartphone and pc usage in the day of an information worker. *Pervasive computing*, pages 398–405, 2009.

[39] Christian Kray, Gerd Kortuem, and Rainer Wasinger. Concepts and issues in interfaces for multiple users and multiple devices. In *Workshop on Multi-User and Ubiquitous User Interfaces (MU3I) at IUI 2004*, 2004.

[40] Michael Kruppa and Antonio Krüger. Concepts for a comined use of personal digital assistants and large remote displays. In *SimVis*, pages 349–362, 2003.

[41] Tsvi Kuflik, Oliviero Stock, Massimo Zancanaro, Ariel Gorfinkel, Sadek Jbara, Shahar Kats, Julia Sheidin, and Nadav Kashtan. A visitor's guide in an active museum: Presentations, communications, and reflection. *Journal on Computing and Cultural Heritage (JOCCH)*, 3(3):11, 2011.

[42] Michal Levin. *Designing Multi-device Experiences: An Ecosystem Approach to User Experiences Across Devices.* " O'Reilly Media, Inc.", 2014.

[43] Ding Li, Shuai Hao, Jiaping Gui, and William GJ Halfond. An empirical study of the energy consumption of android applications. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 121–130. IEEE, 2014.

Bibliography

[44] Tin-Yen Lin, Jing Chen, and Jian-Hong Liu. Enabling cooperative computing for android-based mobile platforms. In *Computer, Consumer and Control (IS3C), 2016 International Symposium on*, pages 763–766. IEEE, 2016.

[45] Kris Luyten and Karin Coninx. Distributed user interface elements to support smart interaction spaces. In *Multimedia, Seventh IEEE International Symposium on*, pages 8–pp. IEEE, 2005.

[46] Kris Luyten, Jan Van den Bergh, Chris Vandervelpen, and Karin Coninx. Designing distributed user interfaces for ambient intelligent environments using models and simulations. *Computers & Graphics*, 30(5):702–713, 2006.

[47] Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. Gradual engagement: facilitating information exchange between digital devices as a function of proximity. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, pages 31–40. ACM, 2012.

[48] Jérémie Melchior, Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 69–78. ACM, 2009.

[49] Tommi Mikkonen, Kari Systä, and Cesare Pautasso. Towards liquid web applications. In *International Conference on Web Engineering*, pages 134–143. Springer, 2015.

[50] Miguel A Nacenta, Dzmitry Aliakseyeu, Sriram Subramanian, and Carl Gutwin. A comparison of techniques for multi-display reaching. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 371–380. ACM, 2005.

[51] Kazuhiro Nakao and Yukikazu Nakamoto. Toward remote service invocation in android. In *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 612–617. IEEE, 2012.

[52] Bonnie A Nardi and Vicki O'Day. *Information ecologies: Using technology with heart*. 1999.

[53] Ingrid Nascimento, Williamson Silva, Bruno Gadelha, and Tayana Conte. Userbility: A technique for the evaluation of user experience and usability on mobile applications. In *International Conference on Human-Computer Interaction*, pages 372–383. Springer, 2016.

[54] Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira Norrie. Interactive development of cross-device user interfaces. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2793–2802. ACM, 2014.

[55] Michael Nebeling, Christoph Zimmerli, Maria Husmann, David E Simmen, and Moira C Norrie. Information concepts for cross-device applications. In *DUI@ EICS*, pages 14–17, 2013.

[56] Sangeun Oh, Hyuck Yoo, Dae R Jeong, Duc Hoang Bui, and Insik Shin. Mobile plus: Multi-device mobile platform for cross-device functionality sharing. 2017.

[57] Reinhard Oppermann and Marcus Specht. A context-sensitive nomadic exhibition guide. In *International Symposium on Handheld and Ubiquitous Computing*, pages 127–142. Springer, 2000.

[58] Antti Oulasvirta and Lauri Sumari. Mobile kits and laptop trays: managing multiple devices in mobile information work. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1127–1136. ACM, 2007.

[59] Fabio Paternò and Carmen Santoro. A logical framework for multi-device user interfaces. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 45–50. ACM, 2012.

[60] Pardha S Pyla, Manas Tungare, and M Pérez-Quinones. Multiple user interfaces: Why consistency is not everything, and seamless task migration is key. In *Proceedings of the CHI 2006 workshop on the many faces of consistency in cross-platform design*, 2006.

[61] David Randall and Pascal Salembier. From cscw to web 2.0: European developments in collaborative design. de. *From CSCW to Web 2.0: European Developments in Collaborative Design, Computer Supported Cooperative Work*, 1, 2010.

[62] Ahmed Salem and Tamer Nadeem. Colphone: A smartphone is just a piece of the puzzle. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 263–266. ACM, 2014.

[63] Samsung. Samsung flow., 2014.

[64] Stephanie Santosa and Daniel Wigdor. A field study of multi-device workflows in distributed workspaces. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 63–72. ACM, 2013.

[65] Taylor Savage. Componentizing the web. *Queue*, 13(8):60, 2015.

[66] Melissa A Schilling. Toward a general modular systems theory and its application to interfirm product modularity. *Academy of management review*, 25(2):312–334, 2000.

[67] Thorsten Schreiber. Android binder. *A shorter, more general work, but good for an overview of Binder. http://www. nds. rub. de/media/attachments/files/2012/03/binder. pdf*, 2011.

[68] Jolien Schroyen, Kris Gabriëls, Kris Luyten, Daniël Teunkens, Karel Robert, Karin Coninx, Eddy Flerackers, and Elke Manshoven. Training social learning skills by collaborative mobile gaming in museums. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 46–49. ACM, 2008.

[69] Katarina Segerståhl. Utilization of pervasive it compromised?: understanding the adoption and use of a cross media system. In *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, pages 168–175. ACM, 2008.

[70] Katarina Segerståhl and Harri Oinas-Kukkonen. Distributed user experience in persuasive technology environments. *Persuasive Technology*, pages 80–91, 2007.

[71] Dong-Hee Shin. Cross-platform users' experiences toward designing interusable systems. *International Journal of Human-Computer Interaction*, 32(7):503–514, 2016.

[72] Maria Shitkova, Justus Holler, Tobias Heide, Nico Clever, and Jörg Becker. Towards usability guidelines for mobile websites and applications. In *Wirtschaftsinformatik*, pages 1603–1617, 2015.

[73] Christos Sintoris, Adrian Stoica, Ioanna Papadimitriou, Nikoleta Yiannoutsou, Vassilis Komis, and Nikolaos Avouris. Museumscrabble: Design of a mobile game for children's interaction with a digitally augmented cultural space. In *Social and Organizational Impacts of Emerging Mobile Devices: Evaluating Use*, pages 124–142. IGI Global, 2012.

[74] Henrik Sørensen, Dimitrios Raptis, Jesper Kjeldskov, and Mikael B Skov. The 4c framework: principles of interaction in digital ecosystems. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 87–97. ACM, 2014.

[75] Antero Taivalsaari, Tommi Mikkonen, and Kari Systä. Liquid software manifesto: the era of multiple device ownership and its implications for software architecture. In *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pages 338–343. IEEE, 2014.

[76] Jari-Pekka Voutilainen, Tommi Mikkonen, and Kari Systä. Synchronizing application state using virtual dom trees. In *International Conference on Web Engineering*, pages 142–154. Springer, 2016.

Bibliography

[77] Minna Wäljas, Katarina Segerståhl, Kaisa Väänänen-Vainio-Mattila, and Harri Oinas-Kukkonen. Cross-platform service user experience: a field study and an initial framework. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 219–228. ACM, 2010.

[78] James R Wallace, Regan L Mandryk, and Kori M Inkpen. Comparing content and input redirection in mdes. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, pages 157–166. ACM, 2008.

[79] Fatos Xhafa, Daniel Palou, Santi Caballè, Keita Matsuo, and Leonard Barolli. Mobilepeerdroid: A platform for sharing, controlling and coordination in mobile android teams. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 961–972. Springer, 2016.

[80] Jishuo Yang and Daniel Wigdor. Panelrama: enabling easy specification of cross-device web applications. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2783–2792. ACM, 2014.