LiqDroid: Towards Seamlessly Distributed Android Applications[*]

Luciano Baresi , Anita Imani
DEIB - Politecnico di Milano
piazza L. da Vinci, 32 - 20133 Milan (Italy)
[name].[surname]@polimi.it

Cristina Frà , Massimo Valla
TIM S.p.A. - Services Innovation
via Rombon, 52 - 20134 Milan (Italy)
[name].[surname]@telecomitalia.it

ABSTRACT

Mobile devices have changed the way we live, but most applications are still conceived for isolated devices and do not allow the user to take advantage of the different devices (e.g., phones, cars, watches, televisions, etc.) opportunistically, efficiently, and dynamically. Multi-device interactions are currently mainly conceived as independent cooperating applications, which then require the a-priori definition of the set of communicating elements, along with the responsibility carried out by each participant. This paper tries to flip the perspective and fosters the idea of *liquid*, loosely coupled distributed Android applications by extending intent-based app communication, usually limited to the same device, to proximal devices. While changing the operating system would have been too expensive, the concept has been implemented through LiqDroid, a middleware that eases the creation of distributed Android applications and oversees their execution on a dynamically changing set of Android devices. Specifically, LiqDroid is an Android service that both augments each single Android device and manages their cooperation. Some example applications demonstrate the main characteristics of LiqDroid and provide interesting insights for possible future developments.

CCS CONCEPTS

• **Human-centered computing → Interaction design process and methods**; *User interface management systems*; *Ubiquitous and mobile computing systems and tools*; • **Software and its engineering** → Message oriented middleware;

KEYWORDS

LiqDroid, liquid computing, Android, Distributed apps.

ACM Reference Format:
Luciano Baresi , Anita Imani and Cristina Frà , Massimo Valla. 2018. LiqDroid: Towards Seamlessly Distributed Android Applications. In *MOBILESoft '18: MOBILESoft '18: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3197231.3197247

[*]This work has been partially funded by TIM S.p.A., Services Innovation Department, Joint Open Lab S-Cube, Milano.

1 INTRODUCTION

Mobile devices are nowadays service enablers that have changed the way we carry out many "common" tasks. While everything started with mobile phones, now we have many different devices (e.g., phones, tablets, watches, cars, televisions, just to mention the ordinary ones) and each user tends to create his/her own group of personal devices. We need different devices [9] because of their different characteristics (e.g., connectivity, screen size, or available sensors), because of ease and context of use, for example, at home, while moving, or at work, and also because of software availability, which means that some apps could only be available on some devices, but not on others.

This diversity is not calling for a new class of devices able to support all the aforementioned situations at once, but paves the ground to more, better, and smarter cooperations among the different devices. Most of these devices tend to work in isolation even when they are close to each other or belong to the same user. Cooperations are only foreseen between "independent" applications that communicate through the cloud, or via Bluetooth or similar protocols. Moreover, they require some form of preliminary configuration and synchronization, and cannot simply exploit the proximal devices opportunistically, efficiently, and dynamically. For example, one would like to be able to switch watching a video from a small phone to a big screen if s/he had the opportunity without wasting time installing and configuring things. S/he could also be interested in sharing some information, or tasks, among the members of a group (e.g., during a family meeting or at work). The execution of a task should continue sequentially or in parallel on the other devices by seamlessly transferring its state and the user experience from the previous device. These are the requirements that motivated *liquid computing* [16], that is, the idea of moving data and computations seamlessly on connected computers, but mobile devices, their availability, and the ease with which they can be aggregated into groups have amplified the need for liquid, mobile computing.

Various studies [1, 7, 8, 10, 13, 26, 35] have already discussed the shift from standalone, unconnected devices to groups of interconnected devices. There are also proposals that address usability models for grouped devices ([18, 21, 27, 33]), but these solutions are often limited, do not really support the simple and seamless flow among devices, and do not exploit the specificities of available mobile operating systems (e.g., Android or iOS).

While usually the integration among devices is conceived as quite static cooperation between independent applications, where the set of participating devices and their role must be defined statically, this paper aims at a wider and more general solution. The idea is to support *liquid* computing [7, 13] on Android devices by distributing the execution of applications on a group of devices, created opportunistically by aggregating devices that are discovered and used dynamically. This means that different devices could carry out the same activity, but the device in charge of executing it is only selected at runtime given the specific needs.

More technically, since Android *activities* and *services* are selected and managed by means of *intents*, our goal is to extend intent management from a single device to a set of federated ones to allow for the distributed execution of tasks. We also want to manage data in a distributed setting to let the different components work on required artifacts as if they were local to the specific device. This also changes the "usual" idea of single-user, single-device application and turns it into a cooperative environment where a single user can distribute applications on diverse devices, but also where different users can cooperate through the same application (task) on different devices.

We aim to look forward and consider a group of connected devices as a shared infrastructure that supports the ideas behind liquid computing. Since extending the operating system would have been a too heavy solution to start from, we materialized the shared infrastructure through a dedicated middleware called LiqDroid[1]. LiqDroid becomes a special-purpose service on each of the grouped devices. The service is in charge of discovering proximal devices, overseeing the execution of activities and services on a single device, propagating the requests/notifications, and taking care of required data to let information and computation be available everywhere and anytime. The goal is thus to support spontaneous, efficient, and dynamic user interactions in a seamless way.

LiqDroid has been used to implement a wide range of different "extended" Android applications, both to test the proposed middleware and demonstrate its capabilities. LiqDroid does not simply support data synchronization or communicating applications, but it also provides a single, uniform framework to implement more complex and sophisticated cooperations among devices. The interactions among the application components are still based on the mechanisms provided by Android (i.e., intents, intent filters, content providers, and back stacks), but a few additional APIs turn a single-device operating system into a distributed execution environment.

The rest of the paper is organized as follows. Section 2 briefly introduces the key characteristics of Android to let inexperienced readers understand the proposal. Section 3 presents the structure of LiqDroid and describes its main features, while Section 4 focuses on its implementation. Section 5 proposes a set of exemplar applications we used to test and assess LiqDroid. Section 6 surveys related approaches and Section 7 concludes the paper.

2 ANDROID IN A NUTSHELL

This section provides a concise description of the key characteristics of Android to let the reader better understand the proposed middleware and its specialties. The interested reader can refer to the official documentation[2] for a more detailed presentation of the operating system.

[1]Because of the sponsorship, LiqDroid and the applications we developed to evaluate it are only available upon request.
[2]https://developer.android.com/index.html

Android only provides a broad definition of *application* as a set of related components packaged and distributed as a whole. Those same components, however, can also use components distributed with other applications or with the operating system itself. A *manifest* acts as application's table of contents and lists all components, access rights, and capabilities. This information is then used by the operating system to decide whether and how to execute the application.

Android components can be roughly divided into *activities* and *services*. Activities are in charge of rendering the graphical user interface (GUI), managing inputs and outputs, and executing the business logic associated with screens. Some activities, defined as launchers, are paired with icons on the device's desktop and act as starting points for the application. An application evolves through a sequence of activity invocations, which define a *task*. Only the activity in the foreground (i.e., whose GUI is displayed on the screen) is executed, and as soon as the user moves to another activity, the previous activity is suspended and added to a task-wide stack (called back stack) of suspended activities. Services act as daemons with no GUIs and run in the background. Activities, even belonging to different applications, can start and stop services, but they can also bind to running services and start interacting with them as needed.

Events are distributed in a *publish and subscribe* way. Each application must have dedicated *broadcast receivers* to declare its interests and then receive events properly. Android broadcasts both system-wide events, for example, to notify that battery level is low or that there is no connectivity available, and specific events generated by the different applications.

Android executes each application in a dedicated process. This means that whenever one of the activities in a distribution package is activated, either directly by the user or through other activities, a new process is created. Each application is run in a dedicated *sandbox* to protect its process and data and components cannot communicate directly. *Intents* are used to inform Android of the intention of interacting with another component. The operating system always mediates the communication and uses the manifest to understand how to manage it. Intents are then used for starting activities and services and also for broadcasting events. *Explicit* intents are used when one knows exactly (through its fully-qualified class name) the component to interact with —typically in the same application. *Implicit* intents are used when one only knows the type of action that should be carried out, but not the name of the component. As said above, this allows one application to use components provided by others and also allows the user to select the component in charge of carrying out a certain action among all the components that could execute it.

The manifest lists all the application's capabilities by means of *intent filters*. The actions activities can carry out are specified by filtering the intents they can accept. The same happens with the broadcasts the application is interested in. Android uses these filters to match requested actions against available ones and to route events. If there is only one activity that can take care of the action, Android launches it, otherwise, it creates a dialog to let the user select the proper component and then launches it.

Android also regulates access to data through *content providers*. Each application can create, store, and manage its own data, through both files and SQLite databases. By default, these data are private to the application, but the application can use content providers to let other applications read/modify them.

3 ARCHITECTURE

LiqDroid helps distribute and manage the execution of tasks among multiple proximal Android devices and exploits a peer-to-peer architecture to avoid creating bottlenecks and relying on devices that may disappear. To materialize this idea, LiqDroid is installed on each Android device as a special-purpose service, runs in the background, and can exploit Android itself to provide applications with common services. Moreover, LiqDroid-compatible components must be created explicitly to deal with some special-purpose intents we defined. These intents are in charge of informing the different devices (i.e., the LiqDroid service running on these devices), distributing requested actions, and providing required data. LiqDroid-aware components can then start, or continue, the execution of activities and services on different devices. Given the use of specific intents, as soon as one of these intents is issued, Android forwards it to the running service on the same device, which distributes the request among the proximal ones. LiqDroid identifies the components —on the different proximal devices— that can carry out the request, and if multiple options exist, asks the user to choose. LiqDroid informs the proper running instance of the service, through standard inter-process communication, and instructs Android on that device on how to manage the request (i.e., the execution of a new action or the continuation of a running one). The new device then becomes responsible for the (distributed) execution of the task.

LiqDroid can work on both already-available components and LiqDroid-aware ones, that is, components developed on purpose. In the former case, it can only consider a component as a black-box and execute it. In the latter case, it can interact with the component at a finer level of granularity and can take application-specific commands into account.

LiqDroid is organized around the two layers of Figure 1. The **Connection Layer** is in charge of setting the infrastructure to let proximal devices interact. *Advertisement & Discovery* takes care of the creation of groups: source devices discover advertised ones (targets). The source collects the list of all possible targets, and the user can identify the one(s) of interest. Each selected destination receives an authentication message and, by accepting it, it becomes part of the group. This way, targets can only accept requests from sources; another authentication message in the opposite direction is needed to let the target issue requests to the source. This is to prevent the execution of unwanted activities on the source and also to avoid inconsistencies that may happen if source and target want to execute an activity at the same time.

Communication Channel & Group Manager takes care of the communication between devices. *Communication Channel* has been conceived to let LiqDroid support different communication protocols, like Bluetooth, Wi-Fi, Wi-Fi Direct, and others, to foster the idea of heterogeneous interactions. The current implementation only supports Wi-Fi-based connections, but conceptually LiqDroid is ready to support other communication channels. *Group Manager* organizes selected devices in groups. By defaults, created groups

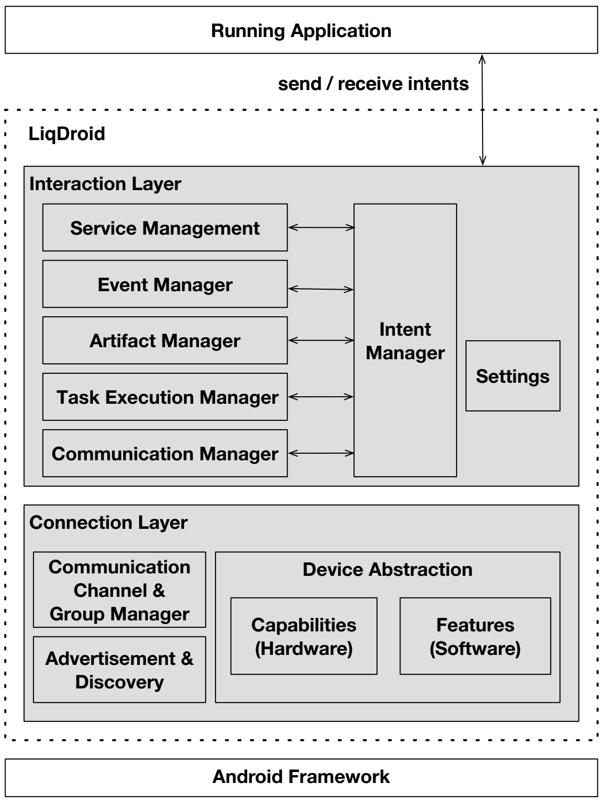

Figure 1: High-level Architecture of LiqDroid.

are private and only visible to the device (user) that created them. The creator can advertise them to let the other devices in the same context join groups directly, without wasting time to recreate them every time. Groups can also be used to filter connected devices, and restrict the execution of a task to the members, and to conceive a more *private* execution context. *Device Abstraction* provides an abstraction of the device, which includes the capabilities (hardware) and features (software) supplied by the device, to handle a wider range of possible scenarios than those that only consider the device type. LiqDroid does not distinguish and classify devices based on their types such as phones, tablets, or watches, but each device, which is part of a group, can then be discovered for its capabilities and features.

The **Interaction Layer** manages the distributed execution of tasks within grouped devices. This layer is in charge of both preparing the devices selected for execution and managing the devices while executing the activities. *Settings* takes care of some device parameters, where the LiqDroid-related name of the device is the most important one. There is no need for changing the actual name of the device, but this LiqDroid-related name can be used to ease the discovery of the device according to the different contexts and needs. *Communication Manager* oversees the authentication among devices and checks the connection status of proximal devices to always provide an up-to-date list of connected devices to each group member.

Intent Manager takes care of intents. Since Android components must interact through intents, the communication with LiqDroid works the same way. Application components issue special-purpose intents to LiqDroid. These intents embed the original ones, and thus LiqDroid supports any Android and user-defined action, along with some extra fields required by LiqDroid to properly identify the component that issues the request and to handle specific requests (e.g., the cloud location of a video we want to play). *Task Execution Manager* then guides the execution of the task through three different phases: activation, synchronization, and termination. During activation, LiqDroid collects all the activities —on all the devices of the group— that can carry out the action requested by the user. As soon as the user selects the activity to execute, LiqDroid launches it on the proper device. The synchronization among devices mostly depends on the action required by the user. LiqDroid supports the following synchronizations.

- Shift occurs when the user decides to move the execution of the current activity on another device. This may happen because the active device is running out of battery, because the user moves and the device cannot follow her/him, or because the user can exploit a better device in proximity.
- Complementarity occurs when the same activity is executed independently on the devices in the group. The initiating device becomes the *master* of the interaction, and all the others act as *slaves*. The master is then responsible for controlling the execution on slave devices and for providing required artifacts.
- Parallelization occurs when the user wants to execute the same activity on different devices and keep the executions synchronized. Again, the initiating device becomes the *master* and all the others act as *slaves*. The execution's state is updated on the slaves as soon as it changes on the master (i.e., the device the user is interacting with).

Since different activities can execute in parallel, the user can interact with all their GUIs at the same time. This option is clearly not available when we consider single-device apps, given that each Android device can only have a single activity in the foreground[3].

The termination on a device starts when the user moves to another activity (task), and thus the one that belongs to the distributed task goes to the background. LiqDroid notifies the user, who can then decide to terminate the activity, move it to the back stack (standard behavior), or overrule the user's decision and move the activity back to the foreground. Because of the limitations Android imposes to the execution of third-party applications, this feature is only available to LiqDroid-aware components.

The *Artifact Manager* oversees how artifacts are stored and shared. First of all, it allows the devices in a group to share storage space and facilitate the co-management of the artifacts required and produced by the different activities. As long as the different activities share the same memory space, intents can only contain a reference to required data. In contrast, LiqDroid considers multiple memory spaces and thus cannot handle references directly.

The first option is that LiqDroid allows for the explicit distribution of data by means of intents: agreed access rights are preserved, and a component can only access received data after providing required credentials. However, since devices can enter and leave a group dynamically, data management through copies and replicas could be quite risky. A significant number of copies should be created to avoid that the only device where some data are stored leaves the group, and devices might want/need to leave a group while data are still to be copied. To avoid this kind of problems, and privilege consistency over true distribution, LiqDroid exploits the cloud to offer a shared and easily-accessible storage space to the different tasks. Generated data can thus be stored locally or on the cloud, and then become available to the others. For example, if one takes a picture on a device, the image is first stored in a private folder on the device provided by LiqDroid, to avoid that the other components on the same device can access it. Then it is moved to the cloud to allow other components (on different devices) to use it in the future. All interested components can change and save their working instances on the cloud; only the initiator can choose among the different versions. This way multiple devices can read-/write the same data and execute the same activities at the same time (in parallel).

LiqDroid also takes care of events through the *Event Manager*. Events can be roughly divided into two groups: those that impact the capabilities of a device, and which are usually generated by the device itself, and those that can shape a particular execution. For example, in the former group, we have *low-battery* events. If a device is close to run out of battery, LiqDroid registers it and does not consider it anymore as candidate device for executing activities. In the latter group, we have notifications of incoming calls or messages. LiqDroid intercepts the message and lets then the user decide how to manage the event: (a) on the device that received it, (b) on another device, or (c) on no device, and the event is then ignored. LiqDroid will then behave accordingly.

Although so far we have mainly considered distributing the execution of activities, LiqDroid can also manage services. A user can launch an activity on a device and start/stop a service running on another device or bind to it. Again this means executing different services (daemons) on neighbor devices and exploiting their features concurrently. *Service Management* provides the link between activities and services in a device-independent way. For example, the user could use LiqDroid to select some songs stored in the shared storage area through a dedicated activity on a mobile phone, discover a music player service on a proximal Android TV, connect to it, play the songs on the television, and control them through the activity on the phone. Different activities on different devices can then bind to the same service[4], and different services on the same device can also be used simultaneously. This way services can easily be shared among proximal cooperating devices.

4 IMPLEMENTATION

The implementation of LiqDroid started with the goal of exploiting as many existing Android features as possible. This is why LiqDroid both redistributes standard intents —to support conventional Android components— and defines some special-purpose ones to provide additional functionality. Since our *Intent Manager* receives

[3]Even if with the new versions of Android and multi-window display things can be a bit more complex.

[4]If a service receives a request while it is still serving a previous one, the running execution is stopped and a new one, to serve the new request, is initiated.

"any" intent, their distribution is carried out by wrapping standard intents into LiqDroid-specific ones. The *Intent Manager* acts as an intent gateway and is responsible for serializing intents on the source device and deserializing them on the target to let Android use them and start required components. Extra intent fields are used to manage specific features.

When the user starts LiqDroid on a device, a setup menu becomes available through the notification bar. Users can define a LiqDroid-specific name for the device, they can enable/disable authentication requests to allow other devices to exploit the device, and they can start discovering proximal devices to create groups.

Before exchanging messages, devices must exploit the functionality provided by the **Connection Layer** to advertise themselves, discover the others, and establish a communication channel. In the discovery phase, each LiqDroid device must exchange messages with the others. The Google Nearby API[5] stores the actual message in the cloud and proximal devices broadcast a key among them to retrieve the message. The message contains the device name and type, the name of the network it is currently using, its IP address and also the socket port it would like to use for cooperating with the others. While the concepts and implementation could be generalized, this phase is currently available for Wi-Fi networks. Note that to check if devices are in proximity, Nearby can use different communication protocols, e.g., Bluetooth, Wi-Fi, and ultrasonic modems. Wi-Fi must then be used to connect the different devices. In fact, when discovered, if a target device is on a different Wi-Fi network than the source device, LiqDroid notifies that target to switch network. The discovery presents a list of available devices to the user. When, one is selected, LiqDroid sends an authentication message, and if the other device accepts, LiqDroid creates a socket-based communication channel between them.

Whenever two devices want to connect an authentication message is sent to the destination device. If the response is positive, the source device can execute activities and services on the destination device. This module is also in charge of providing the updated list of connected devices to LiqDroid. This requires that the status of the Wi-Fi connectivity among devices be monitored.

The *Task execution manager* plays a key rule since it is in charge of overseeing the execution of tasks on the different devices. As said, it is not just the execution of an activity on another device, but the user may want to transfer the execution of a running activity, along with its current state —and data, if needed— to resume it.

In case of Shift, the synchronization requires that the state of the current activity be transferred, and LiqDroid uses the Android bundle to manage the state in case the activity is suspended. Privacy imposes that bundles be only transferred after explicit consent by the user. When the user selects the same component on the destination device, LiqDroid asks the source activity to provide the bundle, serializes the bundle, and transfers it along with the intent that is supposed to launch the activity on the other device. As soon the transfer is complete, LiqDroid kills the activity on the source device. In case of Complementarity, where different activities on different devices are supposed to cooperate to carry out the task, LiqDroid considers the source device as controller of the cooperation and coordinates the others: it can even terminate activities if needed. When devices exploit Parallelization, to keep the state of all the instances of the same activity running on different devices, LiqDroid sends intents to the others as soon as the state changes in the master instance. Given that the state can change in many different ways, this feature can only be exploited if the developer issues these intents, along with required data, properly; LiqDroid will then update the state of the slave instances.

As said, LiqDroid exploits intents to let the different activities (and services) interact. The fine-grained control offered by LiqDroid is obtained through some special-purpose actions:

- Launch must be used to start the execution process and ask LiqDroid to provide a list of all connected devices (along with their components) that can provide the action required in the original intent. The user then decides which devices (and components) to use; if s/he wants to resume an activity on another device, the current state of the original activity must be added as an extra field to the newly-created intent. The list of available devices is obtained by using the same approach used by Android itself to provide a dialog in case of implicit intents and different activities that can execute the action. LiqDroid creates the local list on each device and assembles them on the source one (e.g., Figure 2).
- Update is required when devices work in mode Complementarity or Parallelization to change the states of the activities running on slave devices. The specific intent will then contain also either the newly-provided user input or the changes that must be shared. To overcome privacy concerns and data leakage, the developers are required to use unique names for the intent fields and encrypt sensitive data.
- Feedback is used when slave devices must provide acknowledgments or responses back to the master device. This mechanism could be used to cope with the dynamism of created groups and impose that periodic notifications be sent to the coordinator to let it know they are available and avoid possible problems.
- Terminate kills activities (or services) on proximal devices. Since LiqDroid is a third-party component, it cannot kill activities that are not included in its own package. To allow LiqDroid to kill an activity, and allow Android to remove it from the back stack, the developer must implement the inherited kill method. This capability is key to let LiqDroid mimic activity switching on remote devices. Clearly, when the foreground activity gets killed, the one on top of the stack is resumed.

Besides managing the execution of different components, one should also manage the artifacts (e.g., files and key-value pairs) these components need. LiqDroid can transfer data among connected devices either directly over the network or through the cloud.

As for outsourcing data management to the cloud, LiqDroid exploits the API provided by Firebase[6]. Data can easily be loaded and retrieved, and this also enables scenarios where producers and consumers are not available at the same time. The user can store some data and use them later when the proper device joins the

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

[6]https://firebase.google.com/docs/android/setup

group. Data can be both structured or unstructured. The former elements exploit Firebase as a database management system (DBMS). The latter items are stored in files and can be videos, images, text files, and other. For this kind of data, LiqDroid follows the same convention as Android and only allows the components in the same package to access these data, unless an explicit content provider is defined. LiqDroid creates a special-purpose folder in the cloud and links it to the particular activity (or service), package (application), and device. The folder is removed as soon as the activity is canceled. As for structured data, LiqDroid acts as proxy between the source activity (service) and the database itself. The developer interacts with LiqDroid, which then forwards the requests to the DBMS. For example, one device can "only" provide the database, and the activities on the other devices can use it through LiqDroid.

LiqDroid also allows the user to transfer data directly among devices by means of proper fields added to the LiqDroid-specific intents. Since Android does not allow one to transfer large chunks of data through intents, one should use the URI of the content and then the content provider is in charge of providing it to the other components. To mimic this behavior among devices, and since provided URIs can only be used locally, LiqDroid takes care of transferring data and then provides a correct "local" URI to interested components.

The *Event Manager* plays a key role since Android generates and uses many different events. LiqDroid uses a general-purpose broadcast receiver to capture battery status, storage capabilities, and wireless connectivity on the different devices and share them within the group to prioritize the execution of activities/services given the actual capabilities of devices. In addition, developers can also share specific events: for example, one could intercept a *click* event on a device, transfer it through LiqDroid, and react to it on another device.

Events also play a key role to govern the life-cycle of activities (and services). Given that tasks are executed on different devices, there is one back stack on each device and LiqDroid must manage them properly: each activity must distribute an intent when their methods *onPause*, *onResume*, and *onDestroy* start executing. This is to inform LiqDroid, as a third party application, of the state changes in the life-cycle of the activities previously on the devices. In addition, the same device may receive concurrent requests from different devices. If two devices would like to start an activity on a third device, LiqDroid can either accept the first request and notify the second device that the third is busy, or accept both requests, but this would mean that the first activity is only executed for a while and then paused to allow the second to be launched.

More precisely, as soon as LiqDroid receives a *launch* intent, it creates a dialog to notify the user that a new activity should be started (and thus the current one paused). The user can reject or accept the request explicitly; the action is also executed if no decision is made within a configurable time frame. If a request is rejected, LiqDroid informs the requester device to choose another device. If a controller activity is paused on a device, a LiqDroid-generated dialog allows the user to terminate or pause all launched activities on other devices (*client* activities), ignore the problem, that is, no action is taken on the other activities, or ask the other activities to carry out some particular actions. If a *client* activity is paused, LiqDroid notifies the controller, which can either ignore the notification or even terminate the activity. If a controller activity is resumed (e.g., Figure 3), those client activities that were selected to be paused (through dedicated preferences) are resumed as well. If a client activity is resumed, its controller must be notified about the changes on the slave, which can resume and control it.

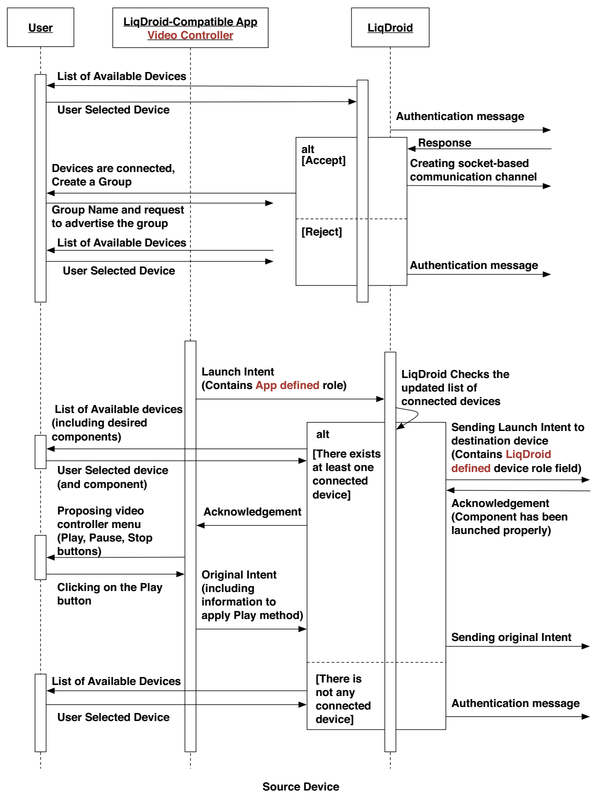
User
LiqDroid-Compatible App
Video Controller
LiqDroid
List of Available Devices
User Selected Device
Authentication message
Response
alt
[Accept]
Devices are connected, Create a Group
Creating socket-based communication channel
Group Name and request to advertise the group
[Reject]
List of Available Devices
User Selected Device
Authentication message
Launch Intent (Contains App defined role)
LiqDroid Checks the updated list of connected devices
List of Available devices (including desired components)
alt
[There exists at least one connected device]
Sending Launch Intent to destination device (Contains LiqDroid defined device role field)
User Selected device (and component)
Acknowledgement (Component has been launched properly)
Proposing video controller menu (Play, Pause, Stop buttons)
Acknowledgement
Clicking on the Play button
Original Intent (including information to apply Play method)
Sending original Intent
List of Available Devices
[There is not any connected device]
User Selected Device
Authentication message
Source Device

Figure 2: Example Sequence Diagram that describes how to launch an activity on the controller device.

The *Service Management* extends the execution mechanism provided by Android and allows an activity on a device to launch activities or services on other devices and to receive results back. This module is also in charge of stopping services when their distributed activities do not exist anymore.

Besides asking for the activities in the group that can provide some actions, LiqDroid also allows the user to retrieve available services. Intents *update* and *feedback* are used to interact with running services. The *Service Management* oversees the parallel execution of different services on the same device. It can also ease data transfer, through both the cloud service and dedicated messages. For example, this allows devices to share the values measured by a sensor on a device with the other devices in the group.

5 EVALUATION

This section discusses our experiences on using LiqDroid on some real case studies. Although LiqDroid could be used to call activities and services of standard, already-available applications, we decided to develop some special-purpose applications to fully evaluate the

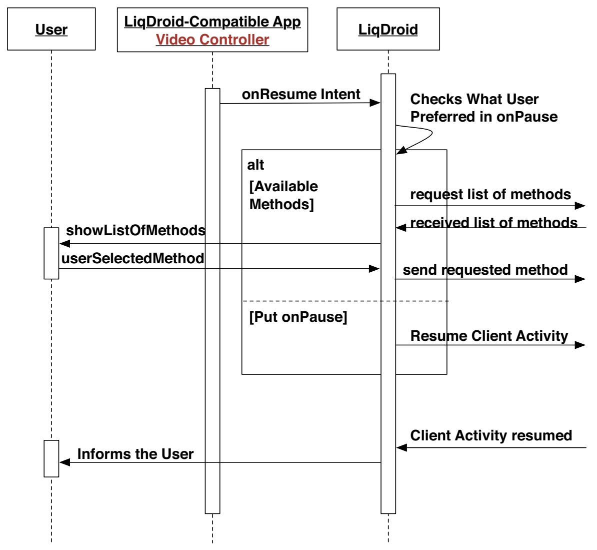

Figure 3: Example Sequence Diagram that describes how to resume an activity.

key capabilities and strengths of LiqDroid. The assessment presented here is twofold since it aims to cover the point of view of both the developer and the end-user. The former is interested in a simple and straightforward solution for implementing distributed Android applications, while the latter wants to seamlessly exploit and enjoy features provided by LiqDroid. These capabilities should not be annoying for the user; neither should they preclude or degrade the normal operation of the device.

While we worked on more applications, and we followed the suggestions of the industrial partners involved in the project, those presented here have been selected to cover and demonstrate the different features offered by LiqDroid. Note that these applications are Android applications that only exploit some special-purpose libraries.

The first experiment we carried out aimed to understand the complexity of using LiqDroid as a developer. The scenario we considered is quite simple: what one should do to use LiqDroid within a given application (activity) to allow users to play videos on the different devices of a group. For example, when one is close to a big Android TV, s/he might want to exploit it instead of limiting the video on the tiny screen of a smartphone. If the developer added a button to ask LiqDroid for the list of capable components, this would only cost nine lines of code — which comprise both interacting with the middleware (service) and launching the proper component(s). Then, the user could select the proper media player and LiqDroid would transfer the video and play it. Everything is done using the standard Android communication means (intents) and there is no need to learn new communication frameworks or even develop new dedicated components. Note that even the toy media player proposed by the official documentation[7] would require many more lines of code (some 120 lines of Java and XML).

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

As for the previous experiment, we also wanted to assess the time needed to distribute the execution. 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 request, that is, s/he has selected the proper component, 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`[8] provided the means to carry out the experiment.

On the source device, LiqDroid takes six seconds to retrieve the content of the user-selected file from the source device, upload it onto the cloud storage, and acknowledge the source application. This includes the time FireBase needs to complete the upload and LiqDroid to create the proper intent and to send it to the source application. It also takes four seconds to distribute the original intent to the other three proximal devices, compose the list of capable components, send it to the source device, and create the dialog for the user. Finally, LiqDroid takes one additional second to create the final LiqDroid message, which embeds the original intent, and send it to the target device. On the destination device, LiqDroid takes three seconds to interpret the message received from the source device and store the video locally. It then needs less than one second to execute the original intent on the newly-generated local URI, that is, to start playing the video. This overall time with respect to what users would need with existing approaches to have the same experience is considered to be satisfactory.

Moving to the user's point of view, we introduce a new application we developed to complement an existing platform for on-line meetings. Since our goal was to demonstrate the simplicity and flexibility of the distributed cooperation capabilities provided by LiqDroid, we speculated a bit on the features currently provided by the platform. First of all, it does not distinguish between remote and proximal members: each participant must have his/her proper credentials, and if a new proximal member wants to join the meeting s/he must have been invited beforehand (e.g., Figure 4).

Our application treats remote and proximal participants differently. The application integrates with a well-known existing web conferencing solution, hereafter called *MeetApp*[9] to manage remote participants. Proximal members can exploit LiqDroid to join a meeting dynamically, without dedicated *MeetApp* accounts, and share their devices as needed. The application distinguishes between presenter and participants, and only the former uses a dedicated general account on *MeetApp*. This means that as long the presenter is one of the proximal members, the role can easily be moved among them and if the current presenter has some problems (e.g., the device goes out of battery) a new one can be elected immediately.

A *MeetApp* meeting is formally between the presenter and the remote participants. Proximal participants can exploit LiqDroid and the presenter's device to distribute the fruition on their devices, share files, and get the list of the current "local" participants, that is, the list of connected devices. They can also use the group itself to start special-purpose services among (a subset of) the members, like chatting privately (and exchanged messages would only be visible in the view boxes of involved participants), scheduling meetings

[8]https://developer.android.com/reference/android/util/TimingLogger.html
[9]Being a commercial product, and since we have no agreements with the company, we are not allowed to reveal its name.

(by sharing preferred dates and adding them to the different Google calendars), and exchanging meeting materials. Since the group can change dynamically, new (proximal) members can join a meeting by simply using their devices —equipped with LiqDroid— to find the proper group, join it, and send a request to the presenter. As soon as a new participant is accepted, and s/he is thus connected to the presenter's device, the list of participants is updated and s/he becomes "visible" to the remote members.

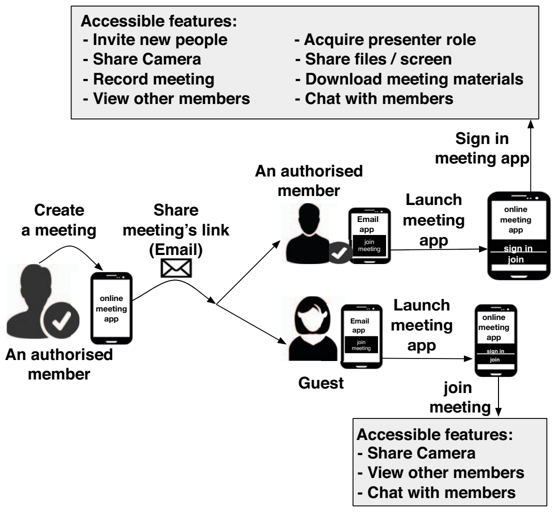

Figure 4: How MeetApp works.

After sketching the additional features, and the flexibility boosted by LiqDroid, we now want to focus on a particular functionality. Let us suppose one wants to open pdf documents on different devices and keep their fruition aligned with respect to a master device. For example,in a digital school scenario, the presenter reads the document and all the other participants must follow the same page. This means that the current state of the pdf document on the master device must be shared among all the group members. How can we then distribute state updates? Android would suggest the use of the Google Nearby Messages API[10], but it would mean learning a new library, sending state updates (55 lines of code[11]), and implementing a service (some additional 40 lines) to send and retrieve the update messages. In contrast, LiqDroid only requires the developer to create an *update* intent as summarized in Figure 5 (11 lines of code, in total) on all devices.

This is only a simple, but important, example to show the differences on implementing a given feature with and without using LiqDroid to manage the distributed management. Another important feature that *MeetApp* proposes to the users is that they can work simultaneously on editing the same document. Then they can share it, along with their changes, immediately or save the different versions to the cloud for future use.

The last experiment (application) we present refers to an integration —and extension— we carried out together with the industrial

```
// Intent for updating the state of
// a pdf document on the proximal devices.

Intent upIntent = new Intent();
upIntent.putExtra("updateType","String");
upIntent.putExtra("PageNumber",pageNumber);
upIntent.setAction("Update");

// Intent used to communicate with LiqDroid
// and deliver the previous Intent

Intent liqdroid = new Intent();
liqdroid.setComponent(new ComponentName("it.polimi.liqdroid",
                     "it.polimi.liqdroid.liqdroidService"));
liqdroid.putExtra("originalIntent",upIntent);
liqdroid.putExtra("SourceAppComponentName",
          "it.polimi.meetapp.PdfControlActivity");
liqdroid.putExtra("SourceAppPackageName",
                 "it.polimi.meetapp");
liqdroid.setAction("appIsRunning");
startService(startLiqdroid);
```

Figure 5: Example update intent.

partner involved in the project. TIM years ago developed an Android application, called *Cameo*, to allow car passengers to share contents and information among them and with a tablet, called console, used as a server. *Cameo* exploits Google Maps to allow people to plan for trips, but did not allow users to easily transfer tasks among devices: if one planned a trip on a device using *Cameo*, s/he could not exploit the plan on Google Maps directly on another device, for example on the console. Another limitation was that one was not able to modify the trip on a device and then send the new directives to the console. For example, one could not search for places of interest (e.g., museums or restaurants) on a device, modify the current trip accordingly, and then feed the console.

Originally, *Cameo* only allowed information sharing among already-running instances of the applications. LiqDroid helped move a step further and launch the application, or other applications (e.g., Google Maps), from a running instance of *Cameo*. While *Cameo* uses AllJoyn [1] to handle the media and information sharing among devices, it cannot be used to start specific activities. In addition, LiqDroid helped widen the scope of possible interactions and *Cameo* can now cooperate with any other application available on passengers' devices or on the console.

This example aims to explain how LiqDroid can help complement existing applications in a simple and easy way. The only requirements are the availability of the source code, to inject the use of LiqDroid properly, and the capability of running our middleware on the different devices.

We also developed some other LiqDroid-enabled applications for both testing our middleware and for becoming more familiar with its features. For example, we realized a simple controller for a home video player, to transfer the fruition of parts of a video on different proximal devices, and also a music player service to ameliorate the management of services executing on some devices through activities running on other devices through LiqDroid.

[10]https://developers.google.com/nearby/messages/overview
[11]This information is retrieved from the official documentation

6 RELATED WORK

LiqDroid borrows ideas and can be compared against many different existing solutions, from event-based middleware frameworks [6, 25, 29], to distributed virtual machines [34], synchronization and management infrastructures for heterogeneous software components [20], and others. However, here we prefer to concentrate on the solutions and ideas that most closely focus on the seamless management, distribution, and shift of computations on mobile devices.

Liquid computing [16] is probably the best way of thinking of LiqDroid since it fosters the flow of user tasks, and not just of data, among devices. At the beginning, a cloud infrastructure was in charge of enabling the interactions among devices, but Apple's Handoff [22] has extended this possibility to sets of Apple devices that communicate through Bluetooth and Wi-Fi Direct, directly. Samsung's Flow [32], still available on certain devices, provides something similar for Android devices. Other solutions, like the middleware for fluid computing [7], mainly support data replication and synchronization to let the data flow between devices. Also Nextbit [5] proposes a solution which is called Baton. It is similar to the Apple's Handoff, but for Android and allows users to switch between devices and continue their tasks by synchronizing data through the cloud. The solution was only supported by the devices running CyanogenMod, a discontinued experimental version of Android. LiqDroid does not support the synchronization and collaboration among devices based on data sharing or on the use of cloud-based services to share resources (e.g., connectivity, storage, data, computation). LiqDroid fosters the direct interaction among devices to allow them to exploit all available activities and services directly at runtime, and thus distribute and manage the execution of Android tasks dynamically, properly, and efficiently. All devices are active entities and can play the role of both provider and consumer [10].

More recently, Gallidabino and Pautasso [13] have also used the word liquid to name the software that allows users to migrate tasks among devices while minimizing the configuration of and synchronization among devices. They have identified different dimensions that one should consider to design liquid software and have used them to evaluate two web-based frameworks. Again, Gallidabino and Pautasso consider that data and state synchronization are the two pillars to enable a seamless flow, while LiqDroid exploits Android to materialize liquidity and distribution at a lower level of abstraction, and also to support more sophisticated interaction patterns.

Cooperating devices have also been investigated in the context of distributed user interfaces (DUIs), where different approaches have been devised to enhance the integration of various devices, improve usability, and increase user satisfaction [11]. The works in this area can mainly be grouped into those that focus on modeling the interactions among devices and those that concentrate on distributing and reconfiguring the elements of the user interface on different devices. As said, the first group focuses on specifying the interactions among devices by considering, for example, the best match between available devices and current task [15] and available hardware features [2, 31]. The modeling phase does not only cover the task at hand, but also the current state, predefined parameters, and content to continue the execution. Although different models and toolkits have been developed for specific problems, there is still no general model or toolkit that suits different scenarios [11]. All these approaches have taught us to consider three key enablers for LiqDroid: one or more *users* should be allowed to work in parallel on the same task, different *devices*, with different capabilities and resources, can be part of a group, and the *context* (environment) may change dynamically. The second group ([12, 14, 19, 23, 24]) is concerned with distributing the user interface and reconfiguring it given changes in the environment and in available devices. They are not interested in distributing the execution of tasks.

There is also a set of web-based frameworks developed to support the collaboration and teamwork among multiple users and devices for supporting collaborative learning and working [3, 30], multi-player games [12], sharing a screen among users and interacting with it simultaneously [28], or distributing a task among several devices [17, 36]. These approaches have been mostly designed for cross-platform applications to overcome device variations, but they are strictly constrained by the browser [28]. This limitation does not allow them to exploit all the resources and capabilities available on the different devices. Modern mobile technologies have made collaborative mobile applications a tough competitor for these web-based applications as they can exploit resources better and faster and they can provide the user with a smoother and specific user experience [35].

Also proximity-based technologies [4] have contributed to conceiving LiqDroid, and probably AllJoyn [1], is one of the most famous frameworks in the area. It is an open-source solution that aims to be independent of the operating system, programming language, and communication protocol and supports the proximal interactions of heterogeneous elements. AllJoyn is a pure communication middleware and does not directly address the problem of distributing computations and user interfaces. Neither is AllJoyn specific to Android and thus able to exploit its capabilities and features. While AllJoyn could complement LiqDroid at the lowest level, it may also be too complex and heavy for connecting proximal devices efficiently and dynamically.

Besides the above-mentioned solutions, which are platform-independent and support heterogeneous devices running different operating systems, we must mention middleware infrastructures, like Sip2Share [8] and the work by Nakao and Nakamoto [26], that augment Android to support collaborating devices. Sip2Share eases the creation of a peer to peer network to provide and request services among Android devices. LiqDroid does not consider the creation of any special-purpose network among devices, works at the application level, and handles all the interactions by means of intents exchanged between activities and services. LiqDroid allows the user to call any activity or service without registering these components in advance, and manages the state of their execution and produced results in a device-independent way.

Some other works address multi-device and multi-user contexts, but those mentioned above provide a good summary of what has been done. In addition, and to the best of our knowledge, LiqDroid is the first middleware specifically designed for Android and for exploiting its intrinsic features and capabilities. As already said, the main idea is to extend the Android virtual machine to a distributed context to allow for the simple, dynamic, and efficient distribution of task executions. LiqDroid moves a first step in that direction, and it also paves the ground to further and more sophisticated options for conceiving robust, distributed Android applications that can exploit available devices dynamically and in an efficient way.

7 CONCLUSIONS AND FUTURE WORK

This paper presents LiqDroid, a middleware infrastructure for distributing the execution of tasks on proximal Android devices. The proposed solution seamlessly manages activities, services, and data. The first experiments we carried out, that is, the first attempts to implement LiqDroid-based applications have provided encouraging results and clearly call for further experiments and more refinements of the proposed solutions. More specifically, our future work will concentrate on improving performance, on ameliorating data management, and on investigating security and privacy concerns.

REFERENCES

[1] AllSeen Alliance. 2013. AllJoyn - a peer-to-peer software development framework for ad-hoc proximity based D2D communication. (2013). Retrieved January 20, 2018 from https://openconnectivity.org/developer/reference-implementation/alljoyn

[2] Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong. 2014. 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 (MobiSys '14)*. ACM, New York, NY, USA, 259–272. https://doi.org/10.1145/2594368.2594370

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

[4] Luciano Baresi, Laurent-Walter Goix, Sam Guinea, Valerio Panzica La Manna, Jacopo Aliprandi, and Dario Archetti. 2015. SPF: a middleware for social interaction in mobile proximity environments. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, Vol. 2. IEEE, 79–88.

[5] KARISSA BELL. 2014. Baton promises to be the ultimate Android app switcher. (2014). Retrieved March 4, 2018 from https://mashable.com/2014/10/27/nextbit-baton-app/#MpJUBwYlKsqs

[6] Gregory Biegel and Vinny Cahill. 2004. A framework for developing mobile, context-aware applications. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, 361–365.

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

[8] Gerardo Canfora and Fabio Melillo. 2012. Sip2Share-A Middleware for Mobile Peer-to-Peer Computing. *ICSOFT* 12 (2012), 445–450.

[9] David Dearman and Jeffery S. Pierce. 2008. It's on My Other Computer!: Computing with Multiple Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 767–776. https://doi.org/10.1145/1357054.1357177

[10] Daniel J. Dubois, Yosuke Bando, Konosuke Watanabe, and Henry Holtzman. 2013. ShAir: Extensible Middleware for Mobile Peer-to-peer Resource Sharing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, New York, NY, USA, 687–690. https://doi.org/10.1145/2491411.2494573

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

[12] Luca Frosini, Marco Manca, and Fabio Paternò. 2013. A Framework for the Development of Distributed Interactive Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '13)*. ACM, New York, NY, USA, 249–254. https://doi.org/10.1145/2494603.2480328

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

[14] Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. 2011. Shared Substance: Developing Flexible Multi-surface Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 3383–3392. https://doi.org/10.1145/1978942.1979446

[15] Dagmawi L Gobena, Gonçalo NP Amador, Abel JP Gomes, and Dejene Ejigu. 2015. Delegation Theory in the Design of Cross-Platform User Interfaces. In *International Conference on Human-Computer Interaction*. Springer, 519–530.

[16] G. Gruman. 2014. Welcome to the next tech revolution: Liquid computing. (2014). Retrieved January 20, 2018 from http://www.infoworld.com/article/2608440/ios/article.html

[17] Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: Enabling and Understanding Cross-device Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2773–2782. https://doi.org/10.1145/2556288.2557170

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

[19] Björn Hartmann, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2013. HydraScope: Creating Multi-surface Meta-applications Through View Synchronization and Input Multiplexing. In *Proceedings of the 2Nd ACM International Symposium on Pervasive Displays (PerDis '13)*. ACM, New York, NY, USA, 43–48. https://doi.org/10.1145/2491568.2491578

[20] Mohammad Hosseini, Yu Jiang, Poliang Wu, Richard B. Berlin, Jr., and Lui Sha. 2015. SINk: A Middleware for Synchronization of Heterogeneous Software Interfaces. In *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware (ARM 2015)*. ACM, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/2834965.2834967

[21] Azham Bin Hussain, Sharaf Aldeen Abdulkadhum Abbas, Mustafa Sabah Abdulwaheed, Rammah Ghanim Mohammed, and Adil abdullah Abdulhussein. 2015. Usability Evaluation of Mobile Game Applications: A Systematic Review. *environment* 2 (2015), 5.

[22] Apple Inc. 2014. Handoff. (2014). Retrieved January 20, 2018 from https://developer.apple.com/handoff/

[23] Marco Manca and Fabio Paternò. 2011. Distributing user interfaces with MARIA. *Distributed User Interfaces* (2011), 93–96.

[24] Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. 2012. 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 (ITS '12)*. ACM, New York, NY, USA, 31–40. https://doi.org/10.1145/2396636.2396642

[25] René Meier and Vinny Cahill. 2002. Steam: Event-based middleware for wireless ad hoc networks. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. IEEE, 639–644.

[26] Kazuhiro Nakao and Yukikazu Nakamoto. 2012. 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*. IEEE, 612–617.

[27] Ingrid Nascimento, Williamson Silva, Bruno Gadelha, and Tayana Conte. 2016. Userbility: A Technique for the Evaluation of User Experience and Usability on Mobile Applications. In *International Conference on Human-Computer Interaction*. Springer, 372–383.

[28] Michael Nebeling, Christoph Zimmerli, Maria Husmann, David E Simmen, and Moira C Norrie. 2013. Information Concepts for Cross-device Applications.. In *DUI@ EICS*. 14–17.

[29] Peter R Pietzuch. 2004. *Hermes: A scalable event-based middleware*. Technical Report. University of Cambridge, Computer Laboratory.

[30] David Randall and Pascal Salembier. 2010. From CSCW to Web 2.0: European Developments in Collaborative Design. DE. 1 (2010).

[31] Ahmed Salem and Tamer Nadeem. 2014. 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 (UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 263–266. https://doi.org/10.1145/2638728.2638743

[32] Samsung. 2014. Samsung Flow. (2014). Retrieved January 20, 2018 from http://www.samsung.com/us/support/samsungflow/

[33] Maria Shitkova, Justus Holler, Tobias Heide, Nico Clever, and Jörg Becker. 2015. Towards Usability Guidelines for Mobile Websites and Applications.. In *Wirtschaftsinformatik*. 1603–1617.

[34] Andrew Whitaker, Marianne Shaw, Steven D Gribble, et al. 2002. *Denali: Lightweight virtual machines for distributed and networked applications*. Technical Report. Technical Report 02-02-01, University of Washington.

[35] Fatos Xhafa, Daniel Palou, Santi Caballè, Keita Matsuo, and Leonard Barolli. 2016. MobilePeerDroid: A Platform for Sharing, Controlling and Coordination in Mobile Android Teams. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer, 961–972.

[36] Jishuo Yang and Daniel Wigdor. 2014. Panelrama: Enabling Easy Specification of Cross-device Web Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2783–2792. https://doi.org/10.1145/2556288.2557199