At the Edge of the Cloud: Improving the Coordination of Proactive Social Devices

Niko Mäkitalo  
Tampere University of Technology / Department of Pervasive Computing  
Tampere, Finland  
niko.makitalo@tut.fi

Tommi Mikkonen  
Tampere University of Technology / Department of Pervasive Computing  
Tampere, Finland  
tommi.mikkonen@tut.fi

Abstract

Today’s Internet-connected devices, such as tablets and mobile phones, have excellent computing power, which creates a possibility for complex, cooperative multi-device platforms. However, coordinating these devices typically requires implementing the coordination process separately in an application specific fashion, which takes focus away from the actual application development. For this purpose we have introduced Social Devices middleware, which allows developers to easily coordinate proactive interactions on a heterogeneous set of devices. Since the proactivity sets its own elements to the coordination, in this paper we introduce our research for coordinating Social Devices. Moreover, as cloud-based solutions typically assume established and fast Internet-connectivity, we also describe how we have complemented the coordination paradigm with Personal Area Network (PAN) based coordination. Social Devices applications can now adapt and choose between cloud and Bluetooth Low Energy based coordination as the JavaScript-based coordination logic can be executed on both, device and server side.

Keywords: Social Devices, Mobile Cloud, Proactive Interactions, JavaScript as a Coordination Language, iBeacon-based coordination

1. Introduction

In recent years, smart devices have become increasingly capable and connected. They are used for everyday purposes: for entertainment, for socializing with friends, and for sharing life events. Continuous connectivity enables the devices to utilize cloud services and perform tasks at the background. Additionally, new sensors are emerging and these devices can be used for tracking user activities and context. However, the cloud services are yet typically utilized by the user using the device, not by the device itself. Thus, cloud or social media services do not support seamless cooperation and interoperability of the devices but rather collaboration of the people. Vendors like Apple or Samsung have created their own standards for sharing resources among devices, like streaming music and videos for instance. These solutions, however, are usually initiated by users and typically require manual efforts to coordinate the devices and their resources. Moreover, these solutions are vendor specific and may eventually lead to vendor locks.

To support cooperation and interoperability in a heterogeneous set of devices we have introduced concept of Social Devices and its initial implementation named Social Devices Platform (SDP) [11]. The system infrastructure is mobile cloud based, abstracting the physical differences of the devices. The concept of mobile cloud here refers to a system where different types of devices are connected with some technology, and hence communicate with each other either directly or through a communication service. Social Devices support the heterogeneity and different resources of each device by regarding them as capabilities: The capabilities describe
what a device can do: the device may, for instance, have TalkingCapability installed enabling it to translate text to speech. Interactions between devices and people in Social Devices are described with a concept of action. An action contains the coordination logic, and hence defines how the devices interact with each other as well as with people. The actions are then proactively triggered by Social Devices applications, based on changes in devices’ context. The current Social Devices middleware has been depicted in Fig. 1.

Initially, Social Devices concept was implemented as a cloud service where the communication between the service and the devices was based on Comet-technology (HTTP long-polling), and the coordination language was Python. While cloud-based orchestrating offered a good starting point for the coordination of the devices, our goal from the beginning was to move towards more flexible system architecture and coordination paradigm where also devices within each others proximity could directly coordinate each others by utilizing various communication technologies, such as Personal Area Networks (PAN). In this paper we report our research of coordinating Social Devices, and describe how we ended up using Socket.IO and Bluetooth Low Energy (BLE) as communication technology, and JavaScript as a coordination language.

The rest of the paper is structured as follows. We start with motivating and presenting some related work in Section 2. Then, in Section 3, we describe the device coordination inside Social Devices ad-hoc mobile clouds, and evaluate the different technologies we have used. In Section 4 we present some future work, and finally, in Section 5 we draw some conclusions.

2. Motivation and Related Work

Currently coordinating devices typically requires implementing the coordination process separately in an application specific fashion. Due to this, the applications running on separate devices are not aware of each other, which make it hard to implement seamlessly cooperating systems. The current situation is unsustainable and the lack of seamless user experience has lead to manifestos like Manifesto for Experience of Things [9] and Liquid Software Manifesto [14].

The approaches for coordinating multiple devices have mainly been focusing on information presentations (e.g. [6, 8, 10]), or for multimedia resource synchronization (e.g. [13]). How-
ever, our work is different, since with Social Devices we are not aiming to offer only automated services or new kinds of interfaces. For example, in [6], we find similarities in the approach for coordinating the devices, but the aim is different. As [3, 6, 10] focus on generating user interfaces and coordinating them on the devices, the system philosophy is more user-centric than ours. We, in contrast, aim to make devices interact and socialize independently, and make the operations visible for the users. When the majority of approaches focus on coordinating the devices in predefined locations, such as smart spaces or homes, our focus is in coordinating the devices wherever they are in the proximity of each other in any location. Several approaches have also been proposed for the modelling and specification of collective actions (e.g. [2]) and for coordinating computational resources (e.g. [1, 4, 5]). We are revitalizing the idea by applying it to mobile clouds, where actors correspond to individual devices forming the cloud, and whereas a centralized entity is responsible for coordinating the execution of mobile devices. In previous research, the closest relative to our PAN-based coordination approach is constituted by coordination languages for mobile agents (e.g. [12]). However, the PAN-based coordination works differently, since we are treating complete mobile devices as agents.

3. Elements of Coordinating Social Devices

The initial approach of coordinating Social Devices was implemented as a cloud service where the communication was based on Comet-technology. Basically, with Comet-based implementation the client maintained HTTP/TCP connection to the server until the server responded with a remote method call. After receiving the HTTP response the client executed the method call and connected again by sending method call response. The coordination language was Python as the coordination service was Django/Python based and the script language worked well in this centralized approach. In the following we describe why we chose to use JavaScript as a coordination language, and how Socket.IO and Bluetooth Low Energy measurably improved coordinating proactive interactions.

3.1. Minimizing Lag and Communication Latency with PAN-based Coordination

The Social Devices concept is meant to support all types of applications that can be proactively triggered in various situations. Consequently, there are also differences how well different interactions tolerate lag. Many of the Social Devices actions are not too critical about the latency or lag, as they are meant to happen in background mainly offering users support in their daily activities by automating things and informing what is currently going on. On the other hand, many actions are much more critical as they require real-time communication and fast interacting with other entities. A self-evident example of these requirements are games where multiple Social Devices take part and need to be coordinated according to the behaviour of other devices. To support faster coordination and situations where Internet connection cannot be utilized, we implemented the coordination process with Bluetooth 4 sockets (RFCOMM) for Android. In this paradigm one of the devices that participates to the interaction is selected to take the role of the coordinator (Fig. 1, phase #A.1), and hence it commands other devices as well as itself (Fig. 1, phase #A.2).

Social Devices are coordinated by invoking their capability methods by a coordinating entity. Basically this means that before a device can be commanded to start next process or update a running process, the coordinating entity needs to receive response from some other device. In a way this requires the device coordination to be synchronous, although the processes running on the devices can be asynchronous. As with any distributed systems, the communication latency between system entities becomes a relevant thing to consider while defining the interactions for Social Devices. The latency in communication affects heavily on the lag that a user typically experiences. In Social Devices the total lag consist of the following:
\[ L_{\text{total}} = \Delta r_{\text{self}} + p + \left( \sum_{i=0}^{n} (g + \Delta c_i + P(x) + \Delta r_i + p) \right) + g + \Delta c_{\text{self}} \]  

(1)

In the equation, \( \Delta r_{\text{self}} \) and \( \Delta r_i \) reflect the latency of relaying the return value from device to the coordinator, \( p \) is the time spent in parsing the return value and passing it to the action body on the coordinator, \( g \) is the time spent in generating a method call, \( \Delta c_{\text{self}} \) and \( \Delta c_i \) reflect the latency in communicating the method calls to the receivers, and \( n \) is the number of capability calls invoked on the other devices before invoking the next method call on the measuring device. \( P(x) \) reflects time spent on executing the capability method on a device, and thus, always depends on the function and the implementation of the method call.

To compare communication latency with Comet and Bluetooth (BT) 4 based coordination and later on with Socket.IO and Bluetooth Low Energy, we minimized the method processing time, \( P(x) \), and implemented a `TestCapability` containing a `dummyMethod` that only saved a time delta between two method calls when it was called by the coordinating entity. In `LagTestAction` (see Fig. 1) the `dummyMethod` was invoked eleven times in a row on one device, resulting in ten time delta values, which were then used to calculate the average time delta to reflect how long it takes to coordinate a device. No other method calls were invoked in between the `dummyMethod` calls. Consequently, the average lag in these measurements mainly consists of the communication latency, and also invoking, parsing and generating the method calls and their responses as described in equation (2).

\[ \bar{L} = \frac{1}{n} \sum_{i=0}^{n} (\Delta r_i + r + g + \Delta c_i), n = 10 \]  

(2)

The results in Table 1 show that the latency in communication clearly affects to the lag, and that the difference between Comet and BT 4 based coordination paradigms is prominent. On average, the latency is 10 to 20 times longer with Comet-based coordination, depending on the used internet connection. Moreover, this asymptotic difference in the lag becomes even more substantial if we consider that the reaction time of a human is typically around 150 to 200 ms for auditory and visual stimulus, and hence, it could be assumed that soon after this time has passed people start wondering why the system is not working properly.

The measurements results also reveal that the used Internet connection in Comet-based coordination has a strong influence on the latency in coordination. The average lag with 3G connection is almost one second, and twice as slow as with wlan connection. Basically this means that Comet-based coordination over 3G connection is too slow in cases where people participate to the interaction or otherwise intensively follow the interaction of the devices. On the other hand, with a decent wlan connection (or very fast 3G or 4G) the 0.45 second lag can still be tolerable in cases which don’t require or offer intensive user input or output.

Compared to BT 4 based coordination there also seems to be more variation in the communication latency in Comet-based coordination, as even a fast Internet connection can become slow at times. Whereas the standard deviation in BT 4 based results is about 2 ms, with Comet-based coordination it is between 23 to 70 ms. This kind of fluctuation in coordination speed may confuse and frustrate users if they cannot be sure if the action execution has ended.

### 3.2. Improving Cloud-based Coordination with Socket.IO

Although the measurements clearly show that the device coordination with BT 4 sockets is much faster, not all the devices yet support Bluetooth, and thus we decided to try improving the cloud-based coordination with Node.js and Socket.IO technologies. Node.js is a server-side JavaScript platform built on Google’s V8 JavaScript Engine. Socket.IO, on the other hand, is a new, officially non-standardized protocol for relying events between client and server, typically
Table 1. Communication latency in device coordination in milliseconds.

<table>
<thead>
<tr>
<th>$\Delta t_i$</th>
<th>Comet (HTTP)</th>
<th>Socket.IO</th>
<th>BT 4</th>
<th>BLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wlan1</td>
<td>wlan2</td>
<td>3G</td>
<td>Android</td>
</tr>
<tr>
<td>1</td>
<td>402</td>
<td>471</td>
<td>976</td>
<td>77</td>
</tr>
<tr>
<td>2</td>
<td>418</td>
<td>421</td>
<td>953</td>
<td>91</td>
</tr>
<tr>
<td>3</td>
<td>371</td>
<td>493</td>
<td>972</td>
<td>73</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>406</td>
<td>973</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>453</td>
<td>476</td>
<td>1078</td>
<td>74</td>
</tr>
<tr>
<td>6</td>
<td>434</td>
<td>469</td>
<td>966</td>
<td>75</td>
</tr>
<tr>
<td>7</td>
<td>395</td>
<td>433</td>
<td>958</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>393</td>
<td>446</td>
<td>1153</td>
<td>71</td>
</tr>
<tr>
<td>9</td>
<td>412</td>
<td>450</td>
<td>970</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>415</td>
<td>441</td>
<td>940</td>
<td>69</td>
</tr>
<tr>
<td>$\bar{\Delta t}$</td>
<td>409.3</td>
<td>450.6</td>
<td>993.9</td>
<td>75.2</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>22.88</td>
<td>26.85</td>
<td>67.36</td>
<td>6.46</td>
</tr>
</tbody>
</table>

utilizing WebSockets as a communication protocol (Phases #B.1 and #B.2 in Fig. 1 represent the current cloud-based coordination). Both of these technologies are especially designed to support fast input/output operations, and hence the new coordination layer also offers an efficient way for the devices to update their contextual information to the Device Registry (Fig. 1, phase #C) and further notify applications through publish/subscribe interface.

The Table 1 shows that the coordination speed in cloud-based coordination was improved substantially, and is now 2-6 times faster, and almost as fast as with BT 4 sockets while using good quality wlan connection. A notable point is also that Socket.IO-based coordination with a wlan connection is faster than human reaction time, which makes it possible to utilize it in applications that require intensive interaction. Based on our experiences, and supported by the measurements standard deviation, the Socket.IO-connection seems to be more stable and only rarely drops compared to the Comet-technology. Also, as Fig. 2 shows, the interaction initialization now takes less time with the new implementation of the communication layer, mainly because of the faster communication. This supports the proactive nature of Social Devices applications, as some interactions are very critical about this. For instance, when people meet or pass by each others, the need for interaction between their devices may be over within seconds.

3.3. JavaScript as a Coordination Language

In the original Social Devices system the interactions were defined with Python programming language as this platform was Django-based cloud service. However, different mobile platforms, like Android and iOS for instance, natively use different programming languages, and hence cannot directly execute Python. As a solution, automated translation from Python to other programming languages could be applied, but most likely these solutions would be very error prone. Moreover, the actual deployment of the code would still be an issue. In the first experiment of using BT 4 sockets for PAN-based coordination, we used class loader to dynamically load Java byte code from Android application package files (.apk) as we deal with deploying new device capabilities to Android in our current implementation. However, this solution only worked with Android, and the Social Devices concept is designed to support all types of devices.

In Worldwide Developers Conference (WWDC) 2013 Apple introduced its officially sup-
ported JavaScriptCore.framework for executing JavaScript code on iOS 7 and OS X. This allows creating virtual machines or contexts where JavaScript code can be executed, and also allows invoking native Objective-C methods from the JavaScript code. Android has similar support for running Google’s V8 engine on Android devices, although this is not currently part of the official SDK and needs to be separately compiled to the application. As the new communication layer was implemented with Node.js, and JavaScript support on the device-side also seems to be emerging, it was a natural choice for the new coordination language. The jump from Python to JavaScript was easy as they both are dynamically typed languages that can be used for scripting. The end result is that we can now run exactly the same interaction definitions on both, server and device sides. However, generating the device communication stub is done differently. Whereas with the cloud-based coordination the device stub utilizes Socket.IO sockets for sending and receiving events to the clients, with the PAN-based solution the device stub invokes a native Objective-C method on iOS that can communicate directly with the devices nearby. JavaScript seems to fit extremely well for this type of heterogeneous multi-device coordination purposes as the developers can implement the actions that directly run with both approaches. What is more, the support for JavaScript gets better all the time, which makes it possible to implement the PAN-based coordination on many other platforms as well.

3.4. iBeacons and Bluetooth Low Energy

The first implementation of PAN-based coordination that was based on Bluetooth 4 sockets had two major problems. Firstly, it required pairing of the participating devices. Fortunately, the paring only needs to be done once between each device, and thus would not be that big concern with user’s own devices. However, the idea of Social Devices is also to support proactive interactions with friends’ devices, as well as with non-personal and public devices, and thus paring with these devices would have to be conducted before the device can be utilized for the first time. The extra work for the user would have been against principles of Social Devices as one of the main ideas is to reduce the manual tasks that currently requires users’ attention. Additionally, based on our measurements even though the devices were already paired, it took approximately 3.6 seconds to discover and establish connection between two devices. What is more, receiving the initialization command and retrieving participant device information from Device Registry it took about 6.3 seconds to start running the interaction with the BT 4 socket based approach. With cloud-based approach, on the contrary, the paring is never needed, and hence the action execution can be started more freely with previously unknown entities.

Secondly, although the BT 4 socket based communication between Android devices worked pretty well, there was no common way of making the communication work with other platforms, like iOS for instance. The problem with Bluetooth has always been that many devices support only some of the overspecialized subprotocols/services that merely allow communication with specific peripherals, but do not allow developers to specify their own communication protocols.

As the iOS has had Bluetooth Low Energy (BLE) support since version 5, we decided to try out this protocol for device coordination. BLE essentially works a bit different than its predecessors as it allows developers to define their own services. These services are then described with characteristics that can either be readable or writable. The biggest advance is that BLE does not require pairing the devices, but instead allows them to communicate freely if they know each others protocols. The downside with BLE is that currently a device can act only in one role at a time, either as central or peripheral. However, this not an issue with Social Devices as the role of the coordinator is chosen by the server, or the Social Devices application logic to be exact, and hence the coordinating device is commanded to acts as BLE central, and the other participants are commanded to act as peripherals. As the measurement results in Table 1 show the coordination with BLE is as fast as with Bluetooth 4 sockets.

In WWDC 2013 Apple also introduced iBeacons. Whereas iBeacons (at least currently) is
nothing more than Apple’s brand for BLE discovery this kind of branding may drive developers to start implementing proximity-based applications which, on the other hand, may improve the support for BLE as it is currently only supported by the iOS devices and the latest Android 4.3 devices. As from the beginning of developing Social Devices concept (since 2011) we have utilized various versions of Bluetooth discovery to detect other Social Devices nearby, and measured their distance with Received Signal Strength Indication (RSSI) values, we have encountered four major issues. Firstly, the biggest concern with Bluetooth discovery on Android devices has been that Bluetooth discovery at random times interferes with wlan, and breaks the phone’s Internet connection. This happens when the two radios happens to work on the same frequency as Bluetooth changes it channel rapidly. With iOS and BLE discovery we have not experienced this kind of issues. Secondly, doing the discovery has been quite slow, although there has been some research of making the query faster (e.g. [7]). With BLE the discovery is very fast taking only few hundred milliseconds. Thirdly, many platforms, such as iOS and older Android versions only allow making the device discoverable for a short period of time. Finally, doing traditional discovery constantly drains the battery of the discovering device. However, although BLE offers some improvements, the discovery power consumption can still be an issue.

![Theoretically composed diagram of device coordination with different protocols.](image)

**Fig. 2.** Theoretically composed diagram of device coordination with different protocols.

4. **Future Work**

Although Bluetooth definitely offers faster coordination, the big downside is that only few devices yet support Bluetooth LE. While the support is slowly emerging to Android phones, many other devices like smart televisions and Internet of Things smart objects typically offer Internet connectivity only. In this sense cloud-based coordination can currently harness wider spectrum of Social Devices. On the other hand, some smart objects offer BLE connectivity only, and hence supporting also these may help to extend the edge of Social Devices mobile clouds. Moreover, we are currently implementing a hybrid model where during the execution of the action the devices with no BLE support could be coordinated through cloud. Furthermore, at some point we also plan to study peer-to-peer coordination, where the coordination would be distributed to each participant device, and where a token would then be used to allocate capability execution turns on each participant device.

5. **Conclusions**

In this paper we introduced our research of coordinating Social Devices, and how we have improved the original cloud-based coordination paradigm to better support proactive interactions between devices and people. The improvement in coordination speed has been depicted in
Fig. 2: The composed diagram shows that due to Socket.IO and Bluetooth LE the coordination is now substantially faster, and the interactions can take place in less time. Moreover, Bluetooth LE allows PAN-based communication without paring the devices, which supports Social Devices goal to make simultaneous usage of multiple devices more seamless. At the same time, Bluetooth LE makes discovering nearby devices fast, which again improves the proactiveness of the system. Finally, using JavaScript as a coordination language allows flexibly deploying the coordination logic from cloud to device to support situations where fast coordination is required.

Acknowledgements

The research was funded by Academy of Finland (264422 & 283276) and by Nokia Foundation.

References