20th ICCRTS

Title of Paper

PISA: Platform Independent Sensor Application

Topics

Topic 6: Cyberspace, Communications, and Information Networks

Topic 4: Experimentation, Metrics, and Analysis

Topic 3: Data, Information and Knowledge

Name of Authors

Michael A. Krog[2]

Frank T. Johnsen[1,2]

Trude H. Bloebaum[1,2]

Marianne R. Brannsten[1]

Bård K. Reitan[1]

[1]Norwegian Defence Research Establishment (FFI)
Kjeller, Norway

[2]University Graduate Center (UNIK)
Kjeller, Norway

Point of Contact

Frank T. Johnsen

Norwegian Defence Research Establishment (FFI)

P.O. Box 25

NO-2027 Kjeller

Norway

E-mail: Frank-Trethan.Johnsen@ffi.no

This page is intentionally left blank

PISA: Platform Independent Sensor Application

Abstract

The current trend is towards an increased use of cheap consumer electronics for military applications. For example, the abundant smart-devices constitute a cheap yet powerful sensor, processing and communication platform. This paper highlights our work on making a platform independent application for reporting observations and positions of users moving out in the terrain. This application is intended for use on smart devices, e.g., smartphones and tablets. The idea is to make central instances aware of data available at the tactical edge, and allow the data to be transformed from information to knowledge. It is especially the iOS and Android operating systems we focus on, which are respectively owned and managed by Apple and Google. What we want is for the application to work in Disconnected, Intermittent, Limited (DIL) environments.

Several approaches for implementing this kind of application already exist, but they usually focus on one particular platform. In our work we use the freely available PhoneGap framework to gain platform independence. Also, we address the issue of civil technology often having a low level of trust by allowing data to flow from the devices into the infrastructure, but not the other way around. The evaluation of our experiments yielded both promising and successful results. The majority of testing was carried out on the functionality and performance of our application. In the DIL environments, we experienced either complete success or limited success. All in all, the tests were successful. The Graphical User Interface (GUI) was also tested, and it resulted in quite constructive feedback along with suggestions of potential improvements.

1 Introduction

Over the last few years there has been an increase in the focus on, and use of, commercial mobile devices within the defense sector. This development is largely due to the fact that current mobile devices are small, light and very powerful mobile sensor platforms that are available at a much lower cost than special purpose military hardware. Additionally, the end users are often already familiar with such devices, which enable them to be put to use with little additional training. However, utilizing such devices raises questions related to security and trust, as the civil market tends to value user friendliness over highly secure solutions (which often come at the expense of user friendliness). Thus, we treat the platforms as being fundamentally untrusted in our work.

Most previous efforts related to the use of mobile devices in a military context have focused on one specific operating system (OS) or device type (smartphone or tablet). In addition, these efforts have often been focused on developing a stand-alone system, and have only to a lesser degree focused on interoperability with other emerging standards and technologies. In the work presented in this paper, we aim for a more general and widely deployable solution by developing an application that is platform independent, functions on both smartphones and tablets, and seeks to achieve interoperability both with already existing systems as well as the emerging standards being identified by NATO. The goal was to develop a platform independent application for reporting based on available sensor data, such as with observations and positions. More specifically, the user should be

able to report and track his location, along with having the opportunity to take a photo of an incident and report it.

At the Norwegian Defence Research Establishment (FFI) we are researching the mobile complex in a military setting [6]. The mobile complex is the eco-system arising around smartphones and tablets, the networks such devices utilize, the (mobile) Internet with all its services and the increasingly digital and technology competent users. This complex is, in the non-military world, displaying new ways of doing things, so in this activity we are conducting experiments searching for insight into how the military may relate to the mobile complex. That is, how to exploit the possibilities and reducing the risks related to this complex. A central hypothesis of this work is that the mobile complex will be an integral part of future command and control (C2) arrangements. This work was performed in context of the NATO working group IST-118 "SOA recommendations for disadvantaged grids in the tactical domain" [1], and is a step towards building a functioning platform independent prototype for use in national and coalition experiments.

A Disconnected, Intermittent, Limited (DIL) environment is a common description of an environment characterized by the constant possibility of periodic communication disruptions, bad connectivity, and limitation problems when it comes to both network (e.g., low bandwidth) and node capabilities (e.g., battery life, storage capacity, CPU power).

Dandashi et al. [2] define the *tactical edge* based on both a user perspective and a technology perspective. From a user perspective, the tactical edge is users that are warfighters directly involved in executing the mission. Here, "users" are those executing the mission in a forward deployed position. From a technology perspective, the tactical edge is where users operate in environments that are constrained by such things as limited communications connectivity and limited storage availability, e.g., like the aforementioned DIL environments.

In this paper, we investigate using consumer technology at the tactical edge. The idea is that due to the ubiquity of smart devices, there may be situations where users bring their own device and should be able to just load an application for position and incident reports using a device they already carry and know well. The use cases are different from those of dedicated reporting systems and terminals, like for example Nett Warrior [12] or TIGR [5]. There will be cases where availability is essential, and the availability of an opportunistic system could make all the difference. This could happen with abrupt change in situations, or if personnel with no practical access to specialist systems happen to find themselves with information that should be shared quickly. Blue force tracking of low risk assets in low risk situations is another possible use-case.

Allowing users to bring their own devices means that we cannot trust the devices. The devices can be rooted or compromised with malware, meaning that they cannot be trusted with classified information. Thus, we focus on getting information from the devices in the form of position and situation reports, using them as a valuable tool for information gathering and otherwise supporting C2 and situational awareness in the HQ, transforming data to information and finally knowledge. The server in the HQ can perform data fusion and sanity checks on the data, and use it to support the understanding of the situational awareness. At this point we do not allow any information to flow back to the devices from our infrastructure, apart from report delivery acknowledgements. If information were to flow from the infrastructure to the device, we would require the platform to be trusted or do suitable information filtering in a gateway.

We have implemented a prototype application that is platform independent. This paper presents an evaluation of our approach in several typical DIL networking conditions that can be encountered at the tactical edge. Also, we present an evaluation of the application GUI performed by a group of technically savvy but otherwise untrained personnel. These evaluations gave us insights into further developing our solution, and they also enabled us to identify a general set of guidelines for developing applications. We think these guidelines may be of benefit to others seeking to leverage Commercial of-the-shelf (COTS) consumer products for military purposes as well.

Outline of the remainder of the paper: Related work is discussed in Section 2. Section 3 introduces the design and implementation of our PISA prototype. The tests we performed and the lessons we learned from them are summarized in Section 4. Section 5 concludes the paper.

2 Related work

The authors of [3] describe a set of prototypes that demonstrate the use of Web services in collaboration with SOAP in tactical environments. The users here are employing handheld devices to obtain situational awareness data, i.e., from observations. The testing was performed with Android devices exclusively.

Operating at the tactical edge, most soldier systems have some relevance to our work. Many systems now also offer clients running on standard smartphones and operating systems like Android or Windows. For example, the Nett Warrior, taking advantage of the low cost of consumer smartphones, has a solution with a modified Samsung smartphone, and an NSA-approved version of the Android operating system, as an end user device [12]. None of the radio components of the smartphone is used in this configuration and the smartphone becomes the centerpiece of a system with multiple components.

Marti is an advanced information management platform that provides a publish-subscribe-query interface for beyond line-of-sight information exchange between varying numbers of sensors and tactical war fighters in tactical network environments [13, 15]. The Android Tactical Assault Kit (ATKA) is an Android-based user interface and Situation awareness kit for Marti. The kit runs on Android tablets and smartphones and was by the Army Geospatial Center recommended for the Nett Warrior end user device [14].

The Tactical Ground Reporting System (TIGR) [5] is a cloud-based application that facilitates collaboration and information sharing at the patrol level (tactical units in the field). The system enables collection and dissemination of fine-grained information on people, places, and events. TIGR offers a media-rich view of the battlefield with digital photos, videos, and high-resolution geo-spatial imagery. This is a fully tested system in real life scenarios, and it has proven to be very successful in multiple theaters of war with minimal impact on disadvantaged networks. The system has a web-browser client and a distributed server architecture utilizing disadvantaged tactical communication networks.

Smartphone-Assisted Readiness, Command and Control System (SPARCCS) [4] is a system that utilizes smartphones in conjunction with cloud computing to extend the benefits of collaborative maps to mobile users while simultaneously ensuring that the command centers receive accurate and up-to-date reports from the field. Their focus is both on military and civil-military operations and the

system addresses the difficulties on receiving reports from the field entities as well as ensuring these entities also have good situational awareness. The end-user devices run the Android OS and the system utilizes HTTP servlets for communication between clients and the backend.

The "social tactical reporting system" Collective Environment Interpretation (CEI) System [7] is previous work by FFI. The focus is directed at the tactical edge, and consumer technologies are employed. REST is used for messaging, and the Internet and commercial mobile networks are utilized. Both a web-browser client and an Android native client has been developed, but only the native Android application is relevant in DIL environments and has been used in experiments. Being a Norwegian prototype, we continue building on parts of this work (i.e., the back end server) in the prototype solution discussed in this paper. Now we pursue a platform-independent solution (supporting the two major platforms iOS and Android) as a first step towards allowing users to bring their own devices. Further, we must ensure that data collected by the mobile users can be delivered into the infrastructure where it can contribute to gaining situational awareness. We consider interoperability as important; therefore we also include NATO standards where applicable.

3 Design and implementation

We identified a set of requirements that the prototype should fulfill. In essence, we found that the application should

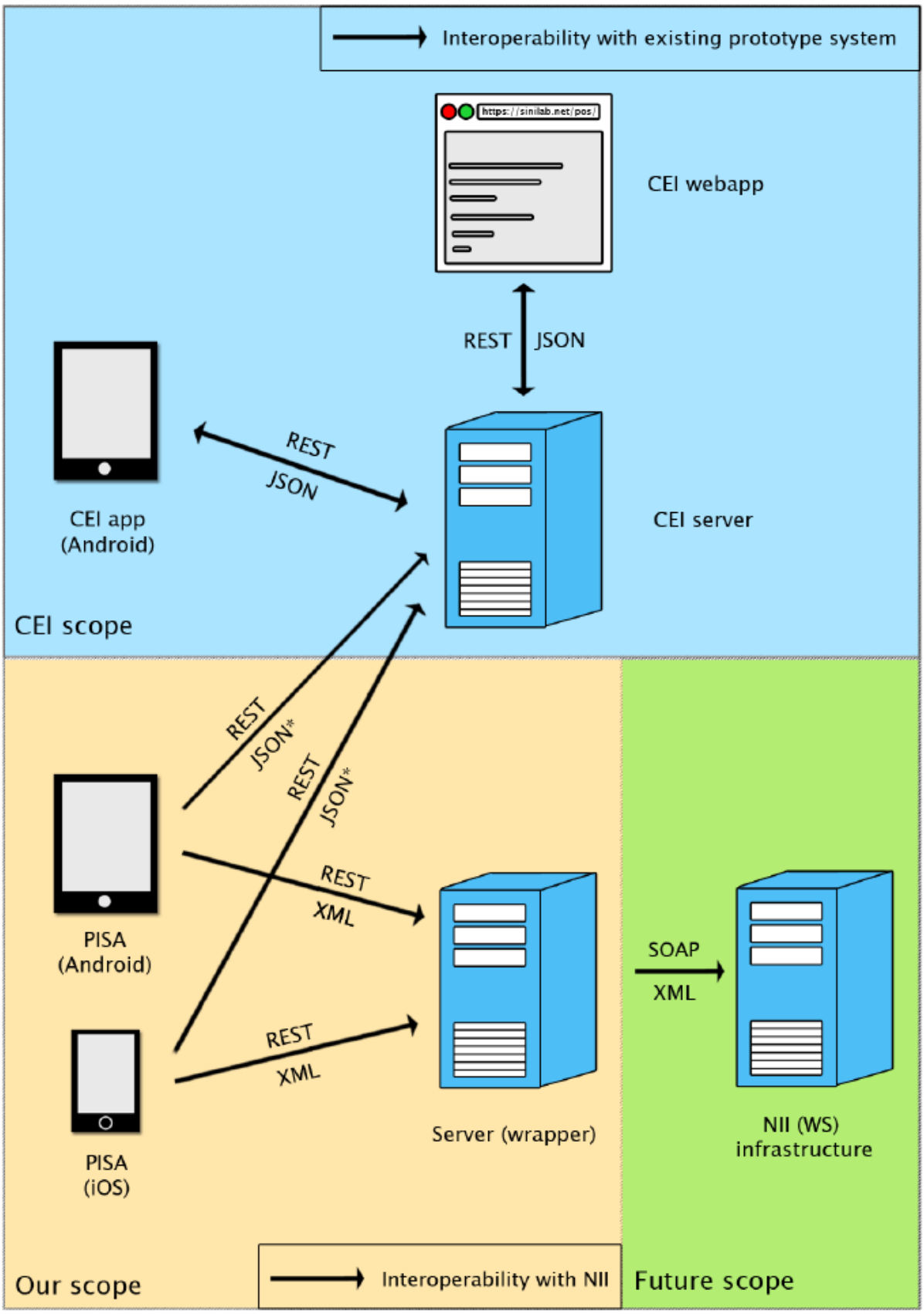
- be platform independent
- support major smart device OS's (iOS and Android)
- be implemented using a free and open source framework
- work in DIL environments (e.g., tactical edge)
- support unreliable message delivery (for loss tolerant messages, e.g., positions)
- support reliable message delivery (for messages that are not tolerant of loss, e.g., observations)
- minimize app footprint (i.e., use REST as communication method)
- have a simple and intuitive Graphical User Interface (GUI)
- use standards when possible
- facilitate interoperability towards back-end infrastructures (SOAP Web services for NATO compatibility, CEI native communication to leverage existing server prototype)

This led us to investigate application development frameworks that could support us in our endeavor. We considered several different frameworks, but ended up using PhoneGap. The rationale for this was that in addition to it being completely free and open source, it was very well-known, with a large user base (seemingly the largest among the reviewed frameworks[1]). Furthermore, the framework is frequently updated and easy to integrate with major development tools. In essence, PhoneGap supports important features of the underlying hardware in a platform independent manner. Hence, it provides a good foundation for developing a prototype application for position and incident reporting. See Figure 1 for an overview of features supported.

[1] In addition to PhoneGap we had a look at Rhodes, Kony, Mono, FeedHenry, webMethods, and Mowbly. For the complete comparison, see the work in [8].

Feature	Description	iOS (3GS+)	Android
Camera	Provides access to device's default camera application. Can take photo using the camera, or retrieve one from device's image gallery.	✓	✓
Capture	Provides access to device's image, audio and video capture capabilities. Can capture images using device's camera application and return information about captured image files.	✓	✓
Geolocation	Provides access to location data based on device's GPS sensor, or inferred from network signals. Can return or watch for changes on device's current position.	✓	✓
File	Providing functionality to read, write and navigate file system hierarchies. Can upload and download files to and from a server.	✓	✓
Connection	Provides functionality for detecting any available type of network, and provides access to the connection information.	✓	✓
Notification	Providing access to visual, audible and tactile device notifications. An alert, sound and vibration are examples of notifications.	✓	✓
Storage	Provides functionality to store data locally on the device. Utilizes HTML5's alternatives for storage: LocalStorage, WebSQL and IndexedDB.	✓	✓

Figure 1: PhoneGap - overview of supported features (adapted from [9] and [10])



* Position and observation requests are sent with JSON, with the exception of observations with an image attached. These are sent as a form submit with encoding type "multipart/form-data" to allow sending of entire files in the data.

Figure 2: Overall Design

The requirements we identified led us to the overall design shown in Figure 2. The figure illustrates how PISA communicates and works with the other components of our solution. The figure is split into three separate areas, which define our scope, CEI's scope and the scope left for future work. These distinct zones are included to explicitly show the scope that belongs to our solution, which already exists, and what is left to be implemented by others.

The bottom left area of the figure defines what we have developed now. The area on the bottom right is the future scope. It illustrates what is left for future work to make our solution completely compatible with NATO or the national infrastructure(NII). This is a necessary component to make PISA work with other solutions. This ensures future compatibility with NATO networks, as the wrapper can be extended to communicate with the desired back-end systems. Implementing this, however, is left for future work.

The topmost area of the illustration shows the components developed by CEI. Here, we do not use the native Android CEI app, since PISA is intended to be a platform-independent replacement. We do, however, integrate with the existing CEI backend system, the CEI server. This area, and the arrows leading to the server from PISA, are included to illustrate how our solution also supports the existing CEI prototype systems.

All communication between PISA, the wrapper server, and CEI server is transmitted through the REST interface. The data format used between PISA and our wrapper server is XML, specifically the NATO Friendly Force Information (NFFI) format (draft STANAG 5527) and a proprietary format that we call "FFI-incident". By utilizing NFFI, we ease future interoperability with NATO systems. The other format, FFI-incident, is not a standard. It is a proprietary format for reporting observations with XML. We use it here because, to the best of our knowledge, there is not a single agreed-upon format for observations in NATO that we could use instead (see [11] for a complete description of the FFI-incident format).

CEI uses JSON as the chosen data format for communication between application and server. Only completely proprietary formats are used for reporting positions and observations in this solution. Hence, our app can talk to two different back-ends: The all-proprietary CEI, and the wrapper that leverages XML and relevant standards in an attempt to ease future compatibility with NATO.

Application functionality

The application revolves around two central aspects: Reporting one’s own position and reporting incidents/observations. Here, the position can be provided periodically without user interaction. It is also possible to manually trigger sending a position update, which can be useful if one has been disconnected for a duration of time and wants to ensure that a position update is shared as soon as possible. Observations are only sent manually, and can constitute a number of different things. In its simplest form, an incident report can contain only a written text describing the event. An incident can also be tagged with a position if desirable, and it is also possible to include an image taken using the built-in camera.

Server Design

There are two different servers PISA submits its reported data to; our wrapper server, and the CEI server. The designs of these servers have some distinct differences, even though they also are quite similar in the way they communicate.

The wrapper was implemented using PHP. This component's sole purpose is to work as a test server that our application can interact and communicate with. Currently it provides the minimum of functionality necessary to accept and store data pushed from the app. Hence, interoperability with NATO should be enabled with some further work. The data stored in the server is in XML format, and is ready for setup with SOAP and Web services in the future. The idea is that this server can be used to create an interoperability gateway between our app and Web services-enabled back-end infrastructures.

The CEI server is more complex and complete in relation to our wrapper server. It includes more functionality, and provides several interfaces. We use the interfaces related to position and incident reporting in the work described here. The server accepts data built in JSON formats, in addition to other data structures.

Graphical User Interface

Designing a good user interface is challenging. However, by following the guidelines gleaned from related work and the CEI system's developers at FFI, we could compile a set of design guidelines:

1. Mind the font size.
2. Don't overwhelm the user.
3. Use big buttons for fat fingers.
4. Shallow menus require few clicks to navigate.
5. Adapt the menus to the device capabilities.
6. Important functions should be quick to access.
7. Ease of use: Aim for an intuitive app requiring few clicks and little scrolling to achieve a task.

The first design rule is to beware of having too small font size on most of the text used in the app. Since PISA is designed to be used on different device types, ranging from compact smart phones to larger tablets, the GUI must function equally well across a range of different screen sizes and resolutions. It is important that the user does not spend all focus on squinting on the mobile screen to read the small text. It is also essential to get an overview of the application quickly, due to pressure or stress if the user runs into an incident of importance. We have used an appropriate font size of all the essential text in PISA.

Another design guideline is to be careful not to display too much text or information to the user. Since the user often could be in a stressed situation, and if the information is not short and concise enough, this may result in the user failing to register vital details. With PISA, we operate with as concise messages as possible, letting the user understand the essential information with as little text as possible. Figure 3 shows an example screenshot from PISA.

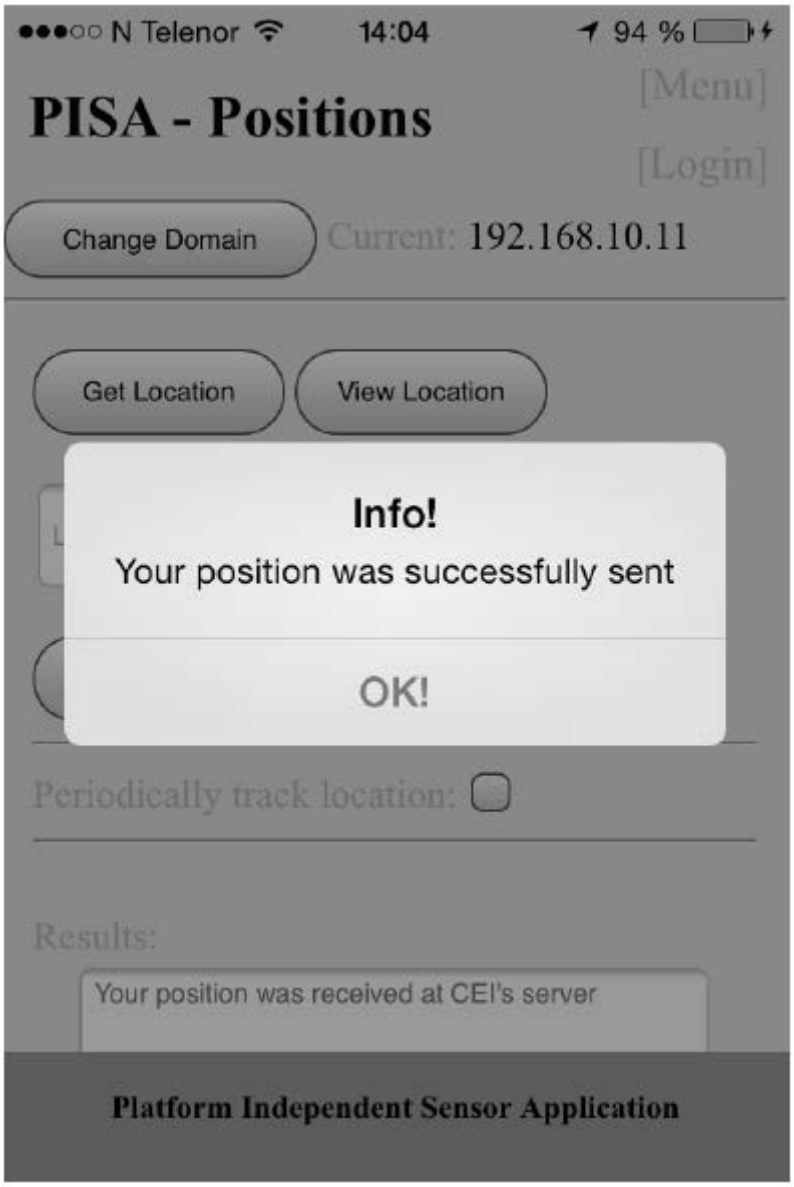


Figure 3: PISA screenshot, font size and information amount example

"Fat fingers" could be a problem when operating applications on handheld devices. Users can be out in the terrain walking or running, or they could be wearing gloves. If equipment with touch screens become standard gear, this of course requires that the potentially used gloves are designed for use with touch screens. This leads to a problem if the buttons they click on are too small or too close to each other. Therefore, big buttons are necessary in order to overcome the "fat fingers" issue. We generally try to have large buttons in our app, without making it feel abnormal. The buttons are enlarged to some degree, and we have different sizes on different groups of buttons, depending on how vital they are and how much they are believed to be pushed. Figure 4 shows PISA's menu, where we use big buttons.

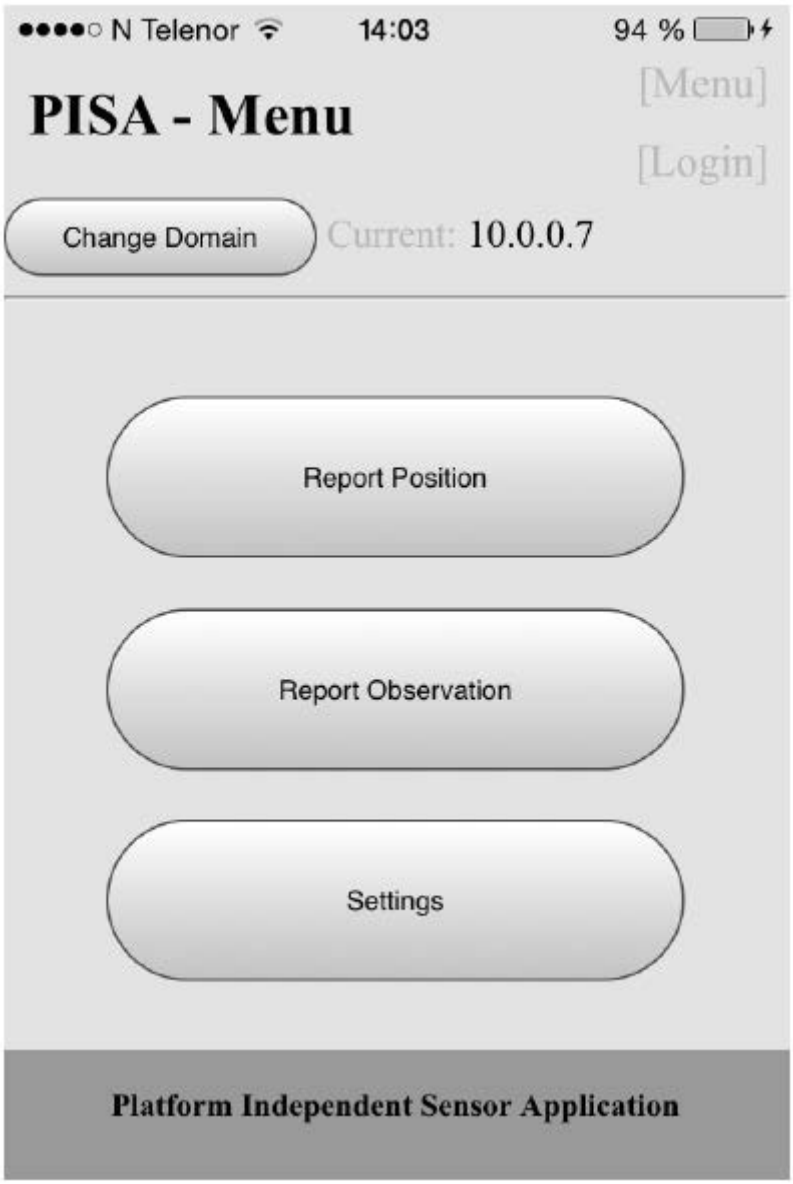


Figure 4: PISA screenshot, example of big buttons

Easy navigation is also a requirement for a good user experience. Easy navigation between pages is required to make the user feel that he always can get to the essential pages with just a few clicks. The navigation must be clear and visible, and also easy to press so the user does not have to feel any additional pressure if already in a stressed situation. A navigation button called “Menu” is included in our application. This always takes the user back to the starting position of PISA, which is the menu. Since PISA does not consist of a large amount of different pages, but only a few, this button might be all that is necessary to navigate the user in the most optimal way.

Screen sizes and screen resolutions vary with every handheld device, especially when it comes to the difference between smartphones and tablets. Smartphones are of course usually smaller, and tablets have bigger screen sizes and possibly greater screen resolutions or different dimensions. This application focuses on being deployed to iOS and Android devices, where the range in screen sizes could be quite big. Our design mostly includes percentage values when it comes to the size of the different graphical components of the app, thus resizing the text, buttons and input fields according

to the each screen size or resolution on its own. It is important to adjust the amount of information displayed to fit the screen size.

The main functionality of the application should be easily available, and it should be simple for the user to understand what the app does. This is essential for helping the user understand the area of use this app is meant for. This is why we wanted to highlight and emphasize the position and observation aspect of our application. The main functionality should also be fast and easy to use, while secondary functionality like the configurations is more accepted of being slower and more advanced. This is because users might adjust the settings on the app before they head out on a mission, while the main functionality could be repeatedly used when in a stressful situation. Therefore it is quite important that this functionality is fast, and easy to locate and use.

Novices and more experienced users have different requirements when it comes to how the functionality of an application is presented. For novices to quickly get started with the use of an application, it is important that the app is intuitive and self-explanatory, so that users understand what needs to be done fast. For a more experienced user, it is often more important that the application is efficient in use, meaning that the functionality that is used often is quick to access. Examples of such effective use is to have few clicks and little scrolling to carry out a task. It is not always possible to take full account of both these factors simultaneously, and in PISA we have chosen to focus mainly on novice users and having an intuitive design.

Another important thing to remember, is that the application should work equally well on both iOS and Android, and to some extent it should also look and feel the same. What we must keep in mind is that some users are very comfortable with one of the mobile platforms, and maybe not at all with the other. The iOS and Android operating systems are quite different from each other, and native apps on each platform could feel very different. We aimed for somewhat of a middle ground - a kind of neutral application that both iOS and Android users should feel comfortable using.

4 Evaluation

The evaluation was performed in three phases: Function tests, DIL tests and GUI tests. For evaluating the results of each test, we use the following terminology: Success, limited success and failure. Success is used for when the test is completely successful and works every time. Limited success means that the test is successful sometimes, or at least some portion of the test is successful. Failure is of course when the test is unsuccessful and fails every time.

Emulators were used much during the implementation of the app, but to better evaluate how the application works in actual use, we tested PISA using real devices (see Figure 5). The “Display”-column provides the device's screen size in inches, then the screen resolution in pixels (in parenthesis). The API level is terminology reserved for Android devices only, since iOS does not operate with this. API levels are intended for developers, and it is an integer value which represents what built-in functions and functionality are available for the developer. The higher Android OS version, the higher API level. As the level increases, offered functionality adds up. The rest of the columns (starting at the left) describe the devices’ OS, manufacturer and type of device, model and OS version. Only Google has named the Android OS versions.

OS	Manufacturer	Model	OS version	API level	Display
iOS	Apple smart-phone	iPhone 4S, 16GB	7.1.2	-	3.5" (960x640)
Android	Google tablet	Asus Nexus 7, 32GB	4.4.3 (KitKat)	19	7" (1280x800)
Android	Samsung tablet	Galaxy Tab 2 10.1, 16GB	4.2.2 (Jelly Bean)	17	10.1" (1280x800)

Figure 5: Devices used for PISA deployment and testing

Function tests

The first phase of testing performed was the function tests. In this phase we tested PISA without any forced errors or limitations like connectivity issues, packet loss, limited bandwidth, etc. That is, no emulated or manual errors were intentionally introduced into the network during these tests. With functionality we focus on the transmission of positions and observations to our two servers.

During the execution of these tests, we observed that all of the message transmissions were successful. Both types of position updates (on demand and periodic) and all types of observations were sent from PISA, and were successfully received by both servers. All three devices had the same exact behavior, and nothing was out of the ordinary. All the function tests were successful.

DIL tests

Phase two of the testing of PISA involved testing the application in a DIL environment. We divided this phase into three distinct tests, one where the main focus was on "D" (for disconnected), one where the focus was on "I" (for intermittent connections), and finally "L" (where communication was limited in throughput).

The objective of the disconnected tests (illustrated in Figure 6) was to ensure that the application behaved according to design while being without a network connection for an extended amount of time. During a disconnection, PISA should not ruin the user experience by freezing, or in some other way prevent the user doing other things. It should not crash either, and PISA should maintain control of how many of the periodic positions were lost. When on demand positions and observations are not successfully sent, the user should be informed of this and the data should be cached for later retransmission. The disconnection tests were successful for both the iOS and Android deployments of PISA.

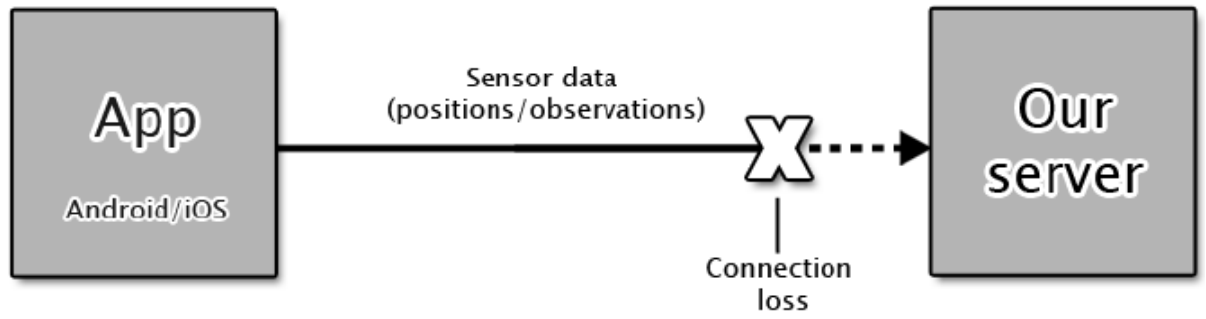


Figure 6: DIL tests – disconnected

In the tests involving intermittent connectivity (illustrated in Figure 7) we lose our connection to the server, and then regain it again within a short timeframe. The objective here is to verify that PISA not only keeps the data that it was not able to send when the connection went down, but also enables the user to resend this data when the connection is reestablished and the user instructs the app to attempt retransmission. The "I" tests were performed for both iOS and Android deployments of PISA, and were all successful.

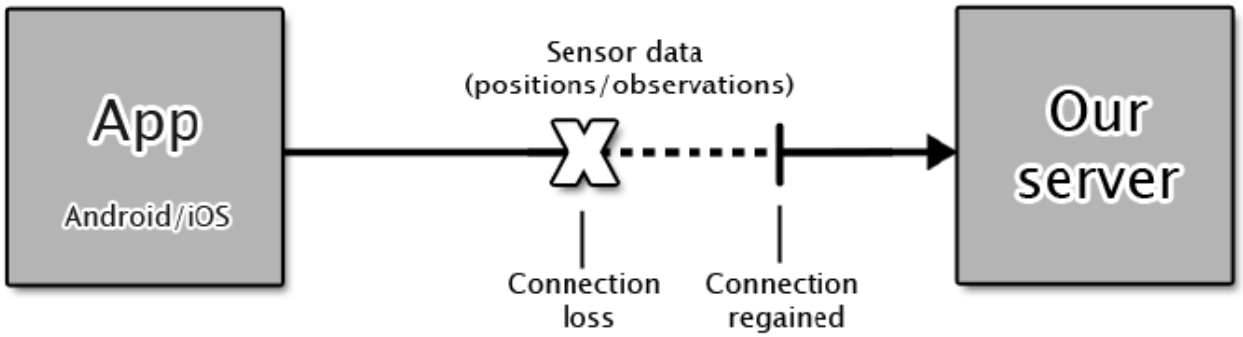


Figure 7: DIL tests – intermittent

The "L" of DIL is more complex to test than the two previous parts. This is because the network can be limited in several ways. High delay, high packet loss and low data rate all represent "limited" aspects of a network. We need to test and evaluate how the application behaves when introduced to networks with limited properties. We must test if positions and the different types of the observation messages behave as desired. When packets are lost, the data rate is low and/or the delay is high, the various messages may fail to reach their destination. This is what we evaluate in the subsequent sections, i.e., if our application handles the different capabilities of the limited networks.

We evaluated these aspects by using the Linux utility called "netem", which allowed us to emulate network constraints. The different emulated networks we tested were identified by the NATO IST-118 group, and involve adjusting three different parameters: Data rate[2], delay[3] and PER[4]. The network properties are summarized in Figure 8.

[2] The *data rate* is the amount of data that can be transmitted over a network per unit of time. It is often used to define how “fast” a network is. Another name also used for the same concept is the network’s “bandwidth”. Data rate is quantified using bits per second (bit/s or bps), often in conjunction with an SI prefix like kilo (kbps), mega (Mbps) and giga (Gbps).

Network	Data Rate	Delay	PER	Type of Network
Satellite Communications (SATCOM)	250 kbps	550 ms	0	Transit
Line of Sight (LOS)	2 Mbps	5 ms	0	Transit
Wireless Fidelity (WiFi) 1	2 Mbps	100 ms	1 %	Tactical edge
WiFi 2	2 Mbps	100 ms	20 %	Tactical edge
Combat Net Radio (CNR) with Forward Error Correction (FEC)	9.6 kbps	100 ms	1 %	Transit

Figure 8: Overview of network charecteristics. These networks were identified for DIL testing by NATO IST-118, and were also used to emulate the test networks in this paper.

A route through a network between a device and the server it communicates with, will in many cases consist of multiple networks that must be traversed to be able to communicate properly. Generally, we can distinguish between something called edge networks and transit networks. Edge networks are the networks the end user is connected to. Transit networks are networks that the information must cross to reach its destination. Figure 9 illustrates how the networks from Figure 8 can be encountered as either edge or tranit networks.

The objective in the following tests is to verify that PISA manages and tolerates the different aspects of the limited DIL networks. The data rate can be low, the delay high and the PER also high. These are attributes which deteriorate the probability of successful transmission of messages. PISA must then be able to store the messages and free the user to try and retransmit them later. We also want to see how many of the messages are successfully sent, and if any are lost. Once again, PISA should not crash while trying to send the position or observation. With periodic positions, PISA should count the correct amount of successful and failed positions.

All the "L" tests were either successful or a limited success. Both iOS and Android deployments of PISA behaved alike, so there were no differences between the platforms involved. The limited successes arose in the emulated SATCOM, CNR, and Wifi2 networks. In the other networks all tests were successful. The limited successes stemmed from a too short timeout in the PISA app, which would erroneously count a message as lost when it in fact had arrived at the server. In CNR/SATCOM this timeout occurrence can be attributed to the high delay of those networks, which made the app time out but not the underlying TCP communication between app and server. Hence, the app counted a failure while the server counted a successful delivery. Finally, the Wifi2 tests were also a limited success. Yet again the issue stemmed from the app timeout value, this time triggered by the high PER, which again triggered retransmissions in TCP and hence resulting in a too high percieved

[3] The *delay* specifies how long it takes for a bit of data to travel across the network from node to node. It is typically measured in fractions of seconds. Milliseconds (ms) or nanoseconds (ns) are often used. Transmission delay or packet delay, which we use, is the time it takes to transfer a packet from the sending node to the receiving node.

[4] PER, short for *Packet Error Rate*, is the number of incorrectly received data packets divided by the total number of received packets. A packet is declared incorrect if at least one bit is erroneous. Packet errors could mean corrupted, lost or duplicated packets. PER is often measured in percent (%).

delay in the app to achieve a successful delivery. Adjusting the timeout in the app remedied this situation and made subsequent tests using these networks successful.

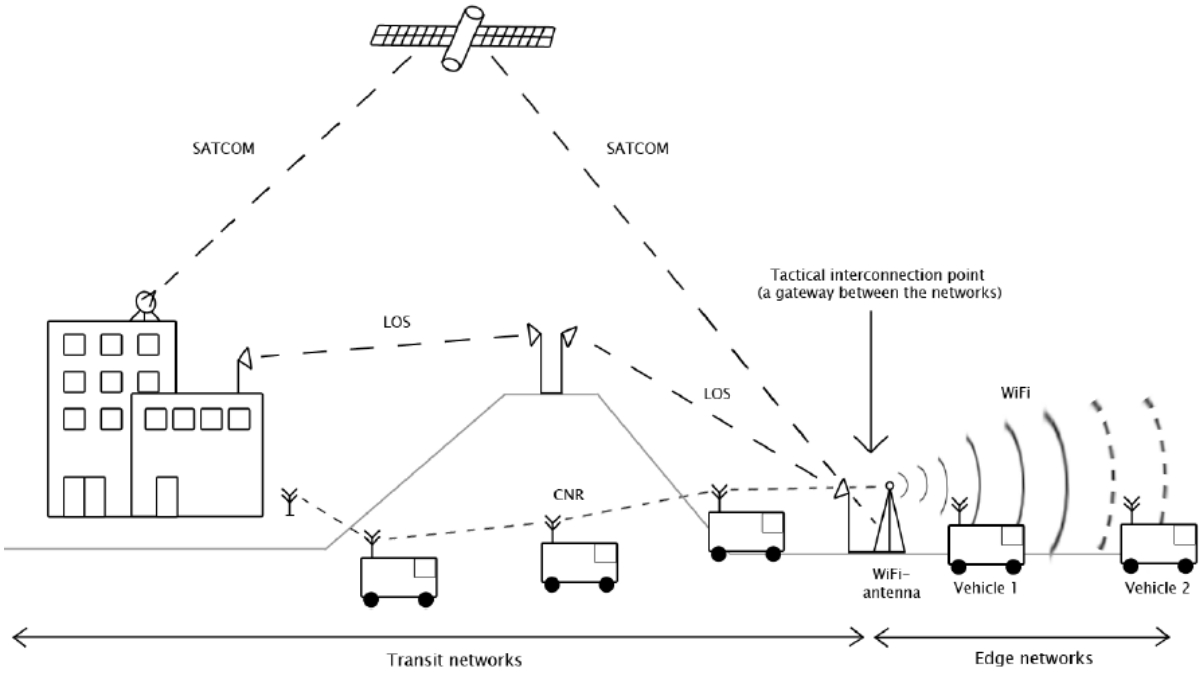


Figure 9: Overview of emulated networks (as identified by IST-118) in our "L" tests

GUI evaluation tests

Following the function and DIL tests we knew that the app was technically sound, and ready to be tested by a small team of users. This team had previous knowledge and user experience with either iOS or Android as a platform, but had not been involved in developing PISA. They saw the app for the first time when performing this evaluation. What we wanted to achieve with these tests was to identify where there was room for improvement for future work on PISA.

We performed a qualitative evaluation of the user experience of PISA. The methodology we chose to use was derived from “Classical aesthetic judgement” by Lavie and Tractinsky - a subjective evaluation method that has been found suitable for evaluating user interfaces [16]. The “Classical aesthetic judgement” method includes several criteria that are suitable for our testing purposes. Important keywords when it comes to elements of evaluation in focus are: Pleasant, clear, clean and symmetric design; these aspects are much what we look for in our application. PISA is a simple app, which should be easy to use and have a clean and symmetric look. The pros of this method are that it is both simple and quick, and we can score the evaluation results by using a questionnaire. We created a questionnaire capturing key aspects of user experiences. The one-page questionnaire started with a brief instruction on how to start the app, and the following questions:

1. Which device did you test on?
2. How easy was it to understand what kind of functionality the app offers?
3. How easy was it to figure out what you needed to do to configure/get started with using the application?

4. How easy was it to use the app?
5. How easy did you think it was to navigate through the app?
6. Did you get sufficient information on what happened, e.g., if an error situation occurred?
8. What did you think of the user interface?
7. Were there any functionality that you felt was missing, something that you expected to be there?
8. This is a prototype that is to be further developed. Do you have any suggestions for possible improvements?
9. Other remarks:

Finally, we provided the application installed on one of the devices along with a questionnaire to the small group of testers and let them evaluate our app.

The main focus of this test was to see how simple and intuitive the application appeared to new users. We also wanted to see if any clear differences in design or behavior between the devices were found. If something did not look or function optimally, we wanted the testers to let us know, so we could make a note of these as potential improvements.

All three devices were tested in the GUI tests, and several testers ended up testing more than one device. The essential feedback and the commonalities between them are discussed below.

The first and the last three of the questions are omitted here. The first is excluded because all devices were tested, and some of the testers evaluated more than one device. No one answered the last question labeled “Other remarks”, and the two last questions are further discussed in the next section (since they talk about potential improvements).

The feedback from question two was very positive. They ranged from very easy to easy to understand what the functionality of PISA was. The menu was experienced as intuitive by all testers.

The evaluation of question three was somewhat varied. The configuration part was experienced as both easy and difficult. This is due to that all devices were configured in advance of the tests. If the app had not been properly set up, the configuration part of PISA seemed somewhat difficult. There were some aspects that were not enough explained, like the NFFI attributes in the settings page (the testers had no previous knowledge of the NFFI format and how to configure the system accordingly).

On question four, the testers answered that it was relatively easy to use the app, and that there were no huge difficulties of using it. However, the observation page was considered as a bit complicated. The different alternatives were not that easy to understand.

With navigation in question five, the testers seemed to think it was pretty easy. However, what they all missed was a “back” button. A “menu” button is present, but some did not even see that button at first. The “menu” button should either way be easier to push (especially on the iOS device, which is the smallest), and much more visible. The same goes for the “login” button. The “menu” button could also be named “home” since it takes the user back to the starting point of the application.

When it came to if the user was provided with enough information at all times (from question six), the feedback was very positive. They turned off their network connection at the devices, and they thought the information on what happened was very good during these situations, and generally otherwise also. The only thing noted, is that the information given sometimes was excessive and unnecessary since some of the information was duplicated in both pop-ups and the results text field.

Question seven, which asks how the tester feels about the user interface, got a lot of feedback. The page for reporting observations was described as "messy and chaotic". It should have more defined lines, and drop the alternative “send” buttons (there was one button for sending to CEI and another for sending to the server). It should have fewer buttons, because they could be combined. The way of choosing an image also felt poor, and should be improved. In retrospect, what we should have done was to allow configuration once for reporting destination (CEI or wrapper or both) and just offer a "send" button in the observations interface. This will be pursued as future work.

Questions eight and nine, when compiled to a list, provided us with the following points for improvement:

- Menu/login buttons should be enlarged and moved further away from each other.
- The “Settings” button should not be equivalent to the report buttons. It is important to distinguish between functionality and configurations.
- The observations page was perceived as a bit messy. Buttons could be combined, and automatically send correct alternative of observation according to which fields are that are filled in.
- The application should also have a “back” button.
- The prototype could be expanded to have video and audio functionality.
- If the app does not really do that much when the network is down, the user could be notified when he opens PISA that the network is down and that he should try again later, before the app closes.
- The “results” fields should have another color (e.g., gray), since it seems that it is possible for the user to edit these fields.
- The “results” field could also be dropped, since the pop-ups describe the same information.
- The settings page of PISA should be more informative, and have possible explanations of what the different terms mean.
- The observation page is very long. It should be much shorter, or have the possibility of a “scroll to top” button.

All in all we found this evaluation valuable. The results ranged from positive feedback to constructive criticism. The points suggestions for improvements will prove very useful in future work on the PISA prototype.

5 Concluding remarks

Our objective was to develop a handheld prototype application for reporting positions and observations. We focused on using consumer COTS products as our platform, targeting it for military use with our position and incident reporting app. We paid special attention to deployment on the tactical edge and DIL environments. Using COTS devices such as smartphones and tablets instead of military hardware, is preferred when possible due to the low cost.

The application needs to be platform independent to avoid placing restrictions on use. However, allowing use of COTS consumer products raises issues with trust. Hence, we allowed information to flow from the device to a central server in order to help create improved situational awareness for the decision makers. Ideally, we want to share some of this information with the device's user as well. This is left for future work, where we aim to explore allowing some information to flow back to the device. Our idea is to attempt to couple this with the identity management and claims-based authentication and authorization specifications that are currently being evaluated for use in NATO's Federated Mission Networking (FMN) Implementation Plan (NFIP).

As of today we focus on improving system robustness, increasing the level of trust and other aspects of security. Another function that we think is worth pursuing is adding status indicators telling the user how “healty” the network is at any given time. Also, future work could investigate how using the application affects situational awareness and other measures of agility.

Acknowledgements

We would like to thank Prof. Dr. Carsten Griwodz for his valuable feedback and discussions that directly contributed to this work. Also, we would like to thank the SINETT project at FFI for all their support in conjunction with us interfacing with and using the CEI server. We couldn’t have done this without you. Finally, we would like to thank Johnny Johnsen for proofreading.

References

[1] F.T. Johnsen, T.H. Bloebaum, P.-P. Meiler, I. Owens, C. Barz, N. Jansen, “IST-118 – SOA recommendations for Disadvantaged Grids in the Tactical Domain”, 18th ICCRTS, Alexandria, VA, USA, June 2013.

[2] F. Dandashi, J. Higginson, J. Hughes, W. Narvaez, M. Sabbouh, S. Semy, and B. Yost. “Tactical Edge Characterization Framework”, MITRE Technical Report MTR070331, McLean, VA, USA, 2007.

[3] S. Simanta, D. Plakosh, and E. Morris. “Web services for handheld tactical systems”, IEEE International Systems Conference (SysCon), Montreal, QC, Canada, April 2011.

[4] M. Crewes, M. Asche, G. Singh, and J. H. Gibson. “SPARCCS – Smartphone-Assisted Readiness, Command and Control System”, 17th International Command and Control Research and Technology Symposium (ICCRTS) - “Operationalizing C2 Agility”, Fairfax, VA, USA, 2012.

[5] J. B. Evans, B. J. Ewy, M. T. Swink, S. G. Pennington, D. J. Siquieros, and S. L. Earp. “TIGR: The Tactical Ground Reporting System”, Institute of Electrical and Electronics Engineers (IEEE) Communications Magazine, USA, 2013.

[6] B. K. Reitan, M. Fidjeland, H. Hafnor, and R. Darisiro. “Approaching the mobile complex - In search of new ways of doing things”, 17th International Command and Control Research and Technology Symposium (ICCRTS) - “Operationalizing C2 Agility”, Fairfax, VA, USA, 2012.

[7] L. H. Karlsen and B. K. Reitan. “CEI - et sosialt taktisk rapporteringssystem: Teknisk beskrivelse av Android klient for smarttelefon og nettbrettstøttet til CEI-systemet”, FFI technical report 2014/00526, Kjeller, Norway, 2014.

[8] A. Welin, K. Johannessen, S. Olimstad, M. A. Krog, and L. Sandvik. "Cross-Platform Mobile Development", Bachelor’s thesis, Oslo and Akershus University College of Applied Sciences, May 2012.

[9] Adobe Systems Inc. “Phonegap Documentation - Platform Support”, http://docs.phonegap.com/en/3.3.0/guide_support_index.md.html#Platform%20Support

[10] Adobe Systems Inc. “Phonegap - Supported Features”, http://phonegap.com/about/feature/

[11] K. Lund, F. T. Johnsen, T. H. Bloebaum, and E. Skjervold. “SOA Pilot 2011 - service Infrastructure”, FFI technical report 2012/02235, Kjeller, Norway, 2012.

[12] A. Dixon and J. Henning. “Nett Warrior gets new end-user device”, WWW.ARMY.MIL, http://www.army.mil/article/107811/, 2013.

[13] M. Gillen, J. Loyall, K. Usbeck, K. Hanlon, A. Scally, J. Sterling, et al., “Beyond line-of-sight information dissemination for Force Protection”, IEEE MILITARY COMMUNICATIONS CONFERENCE - MILCOM 2012, Orlando, Florida, USA, 2012.

[14] K. L. Leonard and D. Cray, “US Army's Nett Warrior Map Engine Trade Study”, U.S. Army Geospatial Center, Systems Acquisition Support Directorate, Alexandria, VA, USA, 2014.

[15] D. Carpenter, “An Approach To Command and Control Using Emerging Technologies”, 18th International Command and Control Research and Technology Symposium (ICCRTS), Alexandria, Virginia, USA, 2013.

[16] M. Pajusalu. “The Evaluation of User Interface Aesthetics”, Master’s thesis, Tallinn University, Institute of Informatics, Spring 2012.