MOBILITY 2017

The Seventh International Conference on Mobile Services, Resources, and Users

ISBN: 978-1-61208-568-5

June 25 - 29, 2017

Venice, Italy

MOBILITY 2017 Editors

Josef Noll, Basic Internet Foundation #Basic4All and University of Oslo/ITS, Norway
Khalil El-Khatib, University of Ontario Institute of Technology, Canada

MOBILITY 2017

Forward

The Seventh International Conference on Mobile Services, Resources, and Users (MOBILITY 2017), held between June 25-29, 2017 in Venice, Italy, continued a series of events dedicated to mobility-at-large, dealing with challenges raised by mobile services and applications considering user, device and service mobility.

Users increasingly rely on devices in different mobile scenarios and situations. "Everything is mobile", and mobility is now ubiquitous. Services are supported in mobile environments, through smart devices and enabling software. While there are well known mobile services, the extension to mobile communities and on-demand mobility requires appropriate mobile radios, middleware and interfacing. Mobility management becomes more complex, but is essential for every business. Mobile wireless communications, including vehicular technologies bring new requirements for ad hoc networking, topology control and interface standardization.

The conference had the following tracks:

- Mobile architectures, mechanisms, protocols
- Challenges in mobile environments

We take here the opportunity to warmly thank all the members of the MOBILITY 2017 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to MOBILITY 2017. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

We also gratefully thank the members of the MOBILITY 2017 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that MOBILITY 2017 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the field of mobile services, resources, and users. We also hope that Venice, Italy provided a pleasant environment during the conference and everyone saved some time to enjoy the unique charm of the city.

MOBILITY 2017 Chairs

MOBILITY Steering Committee
Przemyslaw Pochec, University of New Brunswick, Canada
Rainer Wasinger, University of Tasmania, Australia
Carlo Vallati, University of Pisa, Italy
Masayuki Murata, Osaka University Suita, Japan
Sara de Freitas, Murdoch University, Australia

MOBILITY 2017
Committee

Table of Contents

Improving Quality on Native and Cross-Platform Mobile Applications

Rudy Bisiaux, Mikael Desertot, Sylvain Lecomte, Joachim Perchat, Dorian Petit
FRANCE
name.firstname@univ-valenciennes.fr

***Abstract*—This paper discusses quality markers for a mobile application, both during conception and development, to propose the most suitable way to validate them automatically. We focus on native and cross-platform applications, as well as component based development. To achieve this, we rely both on research papers of the domain, and on our partnership with the Keyneosoft company. An industrial expertise is useful to identify real problems encountered with quality processing for mobile applications.**

Keywords–Mobile, Quality, Product Line.

I. Introduction

With more than 4 billion mobile devices around the world [5] and more than 5 million applications in the different stores [2], the mobile is omnipresent. But developing mobile applications means complying with several constraints and that comes at a cost [22]. In this paper, we highlight the differences between the implementation of classic apps (for Desktop, Web or server) and mobile apps. Afterwards, we define the quality of a mobile application [24] and determine some quality checkpoints. Finally, we describe a solution for validating these checkpoints, to be able to evaluate the overall quality of a mobile application. The objective is to reduce the cost of creating and maintaining these applications by addressing quality control in these two phases. A lot of time is wasted to rollback, hot-fix or replace parts of the application during conception, development or tests, if a minimum quality threshold is not reached. To do this, we introduce the mobile application development concept in Section II. We define the needs for mobile application development, software quality and software engineering in Section III. We detail our approach in Section IV. To finish we conclude in Section V.

II. Mobile development

Developing mobile applications is different from developing classic software even if some similarities exist (like conception, development, test or continuous integration). Two main divergent points are explained based both on the Keyneosoft experience and a survey about mobile applications development challenges [15].

A. Market constraints

A mobile application is produced and released only in a few weeks. This implies the creation of the application quickly and correctly from the beginning. Once deployed, there is no time left to correct mistakes. Moreover, the first release of a mobile application will contain only a few primary features. Afterwards, more and more features are added. The quality of the initial application and all its additional features have to be certified. An application with poor development quality will be more difficult to manage, and adding features will cause regressions.

The heterogeneity of mobile Operating Systems, even in the same family, is also a major problem. The behavior of an application can be different between two OS versions. An example is given by Android version 6.0, where an application needs runtime permissions to work whereas these permissions were not mandatory on previous versions of this OS [11]. To reach all the market stores (Android, iOS) with an application, we have to multiply the supported OS (different languages imply a higher cost of production). Some answers have been proposed, offering cross-platform solutions. Multiple cross-platforms frameworks exist, which are based on web development like Ionic [14] or cross compiling like COMMON [17] and Xamarin [6].

Another heterogeneity issue is due to the manufacturers who apply overlays on OS. Once again, the behaviour of the application can be disturbed by these overlays and the developer has to check for all of them. The last heterogeneity drawback concerns the device screen size. The user interface has to be clear and consistent regardless of the screen size. But with Android or iOS devices, managing the screen size has also a cost, even when using cross-platform solutions.

All these constraints are imposed by the market; they can not be changed but need to be considered when developing mobile applications.

B. Development constraints

To deal with the complexity of mobile applications, different kinds of designs are available. The most popular is the MVC (Model-View-Controler) pattern [19], but some technologies like Xamarin replace this design pattern with MVVM (Model-View-ViewModel) [20], where a view-controller replaces the current controller to notify the view. With these two kinds of designs, the application does not embed a lot of data. The data is usually extracted from a database or a Web service call. For remote sources, a mobile application needs to be connected to collect them. Then, the quality of these services can not be certified because they are externals. This identifies one of the most important challenges, namely, how to maintain the distributed quality over the business logic of mobile applications.

Another challenge is the integration of third party libraries, like when using social networks for connection/identification. These libraries contain uncontrollable code, which has a high risk for quality criteria. They are often interdependent from the main application, and do not follow the releases of the platform's update. When changing an obsolete library, the developer has to verify the impacted code in the application, especially since it has a strong dependency to it.

These constraints produce a lot of bugs, like code quality bugs or integration bugs. In addition, there is currently a context difference between the development of the application and its actual use after release. Indeed, the context like the number of users or the stability of the remote services can change the final quality.

In this paper, we present a suitable solution to take these constrains into account.

III. State of Art

In this section, we discuss the different aspects needed to qualify the mobile software process.

A. *Software product line*

To understand how mobile software is made, let us have a look at the Software Product Line (SPL) defined by the Software Engineering Institute (SEI) to manage and organize a software product processing [12]. This SPL describes the different steps of the process. At the beginning, the entry points are the client needs (functionalities), used at the conception phase to determine the different implementations technologies. But it also helps to define the way functionalities will be isolated in different modules, relying on the components standards. These technical and functional requirements are done by experts. The components designed are then produced during the implementation phase. Finally, the test phase intends to validate the different components and functionalities created during the implementation phase. Afterwards, the release phase is triggered to distribute the final product. Continuous integration is the usual way to automate these phases. In Figure 1, we detail the common use of continuous integration. With some tools, we can automate some parts of the software's process. An orchestrator can play defined jobs to control source repositories or source codes with different versions, compile the code, run tests and delivery the final product. A feedback of all these operations can be provided to developers and to managers. These processes are associated with management methods from the way to develop a software, to the product team management. The team management can impact the quality of the process so we have to consider it. The size of the team and the development' speed time leads us to Scrum management methods [18]. Indeed, this method is specially well suited to these features as said in [22].

B. *Software quality*

Our research mainly focuses on the quality in mobile applications. Quality is an important characteristic of software.

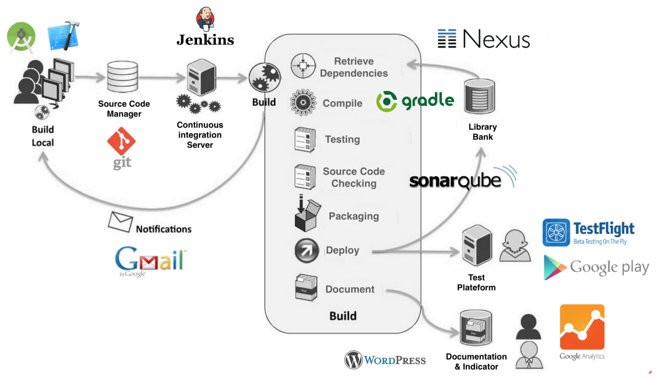
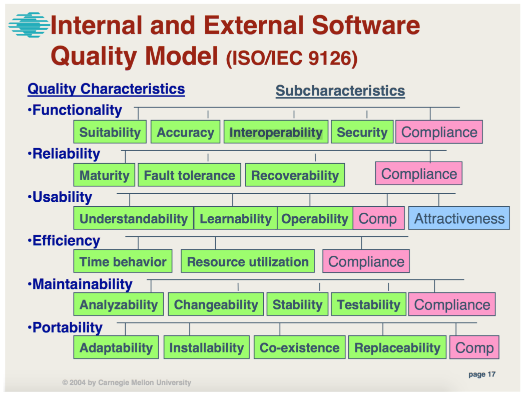


Figure 1: Continuous integration example

When many solutions exist for classic applications, the previous constraints are not embedded in these solutions. The need for a specific mobile quality model is real, as described in [22] or [24]. Software quality is a widespread subject already validated by different kinds of certifications.

Firstly, we have certifications based on the software company structure, like CMMI (Capability Maturity Model Integration)[21]. This approach evaluates the maturity of the company in order to determine its capability to produce quality software. This approach is not suitable for mobile application development because this kind of application is deployed quickly [15] and also because they are organization specific and our approach considers the source code level of the process. Secondly, we have production certifications, where the


Internal and External Software Quality Model (ISO/IEC 9126)

Quality Characteristics	Subcharacteristics				
•Functionality	Suitability	Accuracy	Interoperability	Security	Compliance
•Reliability	Maturity	Fault tolerance	Recoverability	Compliance	
•Usability	Understandability	Learnability	Operability	Comp	Attractiveness
•Efficiency	Time behavior	Resource utilization	Compliance		
•Maintainability	Analyzability	Changeability	Stability	Testability	Compliance
•Portability	Adaptability	Installability	Co-existence	Replaceability	Comp

© 2004 by Carnegie Mellon University

page 17

Figure 2: Square Software Quality criteria

most popular is Square from the ISO9126 certification [25]. The quality of the application is defined by rating different criteria, as described in Figure 2. In our approach, we are looking for validating these criteria by using mobile context checkpoints. When Square certification gives a criterion, we need to find a way to check it during mobile development. Some approaches define a mobile quality model. For example, Zahra [24] proposes to add data integrity notions to validate data consistency when the application is paused or stopped. But it is not enough. Because business logic distribution is strongly used in mobile software, the data integrity must be constant when the application interacts with other systems like servers. So, we have to generate checkpoints that match the quality criteria of the Square certification for every component in a mobile application. Finally, we have to generate checkpoints that match the quality of these components' implementation.

Based on our definition of the mobile development constraints II-B, we notice that every constraint can not match these criteria. The logic distribution as well as the high dependency to third party libraries are too specific and can not be associated with any criteria. For mobile software processes, we use two new criteria to define the quality of remote services, and the usability, quality and quality of integration, of a third party library.

All of these checkpoints need to be controlled so we need to monitor them and decide when we should process them. The next section will tackle these problems.

IV. Proposition

The first step to validate the checkpoints identified above, is defining the stages of a basic software product line [12] :

- Conception, to generate the applications architecture.
- Implementation, to produce the sources.
- And finally, testing, which is done by multiple actors.

But these stages are validated by testing the final product. Our approach is based on defining key points that have to be checked at each level of the software product line.

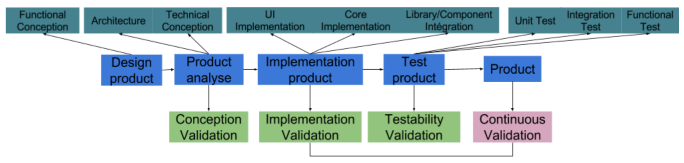


Figure 3: mobile software product line

Figure 3 shows a basic mobile software product line. We added, in green and purple, some steps for checkpoints validations. Thanks to them, the goal is to have, at the end of the process, a qualitative and sustainable mobile application to match the market constraints. These steps are going to be done simultaneously with the software product line ones, and automatically powered by continuous integration tools [7].

A. Conception validation

At first, we generate control points (based on the software engineering standards) for the conception phase of the application. This will allow to validate the usage of design patterns [23], libraries' dependencies and isolation of components [3]. This step will be done by experts. The organizational part of quality in a company is a key to permit these validations. Due to delay constraints imposed by the market, this validation could be time consuming. But agile project managements like Scrum provide time management solutions. An exemple of standard used in mobile development, is the component based development. A component [3] is a reusable self-working element. A mobile application is an aggregation of these components. Each component has its own properties and interfaces which allow other components to interact with him. The quality of this component has to be validated for all the constraints we saw above. As a component needs to work for every OS and version, it has to work against any environment and it can be modified without affecting the other components using it. When all the components are validated individually, and the integration of these components is validated too, we can validate the whole application.

B. Implementation validation

Then, we are looking for implementation validation, and this will be done in two ways. To guarantee that the implementation is matching the conception, we use different methods.

- Generate class diagram using UML (Unified Modeling Language) tools and compare them with the conception phase.
- Use pair programming to validate the implementation by different developers.
- Perform code revision, using version merge requests.

These approaches match with the organizational concept of quality. Afterwards, some differences persist between conception granularity and implementation caused by the technology environment. Because Android and Apple display guidelines to establish implementation standards, they also provide tools to check these rules. But these guidelines are not enough. For example, there is no guideline description to explain the best way to integrate a third-party library. The use of component based programming imposes some rules too. These new rules are suitable for mobile development and can easily be integrated in static analyse tools (like Android lint). They can also be easily integrated to a continuous integration platform. Furthermore, a component or library developer can embed these rules in their components or libraries. This analysis should be done on every code modification and automatically, based on different checkpoints we are going to formalize.

C. Test validation

Then, we check the validity of both unit and integration tests. As shown in our software product line with Figure 3, the tests can be split in three different domains, Unit, Integration or Functional. The objective here is to define proper tests, check implementations and finally run them over different devices, under different OS versions and different context of use (with or without network etc...). This allows us to validate that the software does what it intend to in each context. We have to define a severity threshold to reach for an application to be released. These tests have to be written manually, but can also be automatically recorded and played on several devices with different tools. Once again, component based engineering offers the possibility to embed tests with the component. The continuous integration platform should integrate these tools to automatically run tests.

D. Continuous validation

These different validation steps can certify the quality of the application. We need to ensure that the conception and the implementation are sustainable and that the tests are continuously validating. For example, if just one month after the release, Android releases a new version of its OS, some of our checkpoints can be in a wrong state and our application does not validate any more. We need a way for measuring this impact without releasing a new version of the application. To achieve this, trackers will be added to the application source code to verify the different checkpoints sustainability like in [16]. Once these track events are set, they retrieve information about any potential anomaly and could alert the developer. These notions are already used for crash reporting and runtime healing like [16] but are not used yet to do the same treatment we exposed. This method is used to keep an eye on implementation, when the final user is interacting with the application. Thanks to data callback, we can determine the sustainability of a suitable implementation for evolutions.

The validation system has to be flexible, as some checkpoints are more important than others and priorities may vary during all the process time (configurable severity level).

E. Literature overview

Some papers report challenges like user acceptance [13], highlighting quality criteria from user experience, like Application interface design, Application performance, battery efficiency, phone features and connectivity cost. In [4], Dehlinger and Dixon point out the differences between classic applications and mobile applications, affecting the engineering process. Criteria like mobile screen size heterogeneity, platform heterogeneity etc. are a challenge for developers. Some papers propose a new definition of quality criteria like [10] for a specific branch of mobile app, the M-commerce, by questioning the user. Franke et al. propose a framework automating some existing quality check [9] and to extract a quality model ISO9126 [8]. The tradeoff between speed development and quality is discussed by Hansen [1]. It shows that Agile development is the best suited for mobile development and that quality automatic tools can be used to reduce cost/tile/risk in mobile applications. But the time spent to set up the automation and to maintain it costs more than quality control by different ways for small projects.

F. Realization

To illustrate our approach, we use the case of the Network Http request. As said above, mobile software relies on network to retrieve data from servers and to do this they use http request. We need an exhaustive list of our checkpoints in this use case. These checkpoints are defined by using the specifications of an http request like body, header and error code, pairing with the use case of http request and the development skills. These checkpoints can be embedded in third party libraries. When these checkpoints pass, we can be sure that the application can use an http request call or a library without errors or degrading the quality. In the conception part we have to ensure that the Http request is made by only one component (singleton). For the implementation part, we have to check if every checkpoint is validated by parsing the source code. For instance, we have to be sure that when a POST request is sent, the body part is filled. To finish, we add trackers to monitor the quantity and the reason of rejected requests. We have defined every parameter that compose an HTTP request to generate checkpoints and validate all the parameters for an Http request.

V. Conclusion

Our goal is to increase the quality of mobile applications. To achieve this, we identify a mostly exhaustive list of quality checkpoints extended from the quality model ISO9126 to verify. This quality level will be improved and simplified by relying on component models, from common mobile application functionalities to the core components. These checkpoints have to be validated during all the software process (conception, implementation, test and release) to guarantee its sustainability. A checkpoint, the way to validate it and the actors are led by the phase involved. Moreover, a tracking system is added to monitor checkpoint validation, even after the application's release. For now, we are working on implementing and validating checkpoints over a simple distributed application in a software product line.

References

[1] Hansen Aaron. A mobile software quality framework. Lionbridge Technologies, 2007.

[2] AppFigure. Mobile application quantity on appstore by statrista. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2015.

[3] Xia Cai, Michael R Lyu, Kam-Fai Wong, and Roy Ko. Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 372–379. IEEE, 2000.

[4] Josh Dehlinger and Jeremy Dixon. Mobile application software engineering: Challenges and research directions. In *Workshop on Mobile Software Engineering*, volume 2, pages 29–32, 2011.

[5] DeviceFigure. Mobile device quantity in 2015 by statrista. http://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/, 2015.

[6] Jared Dickson. Xamarin mobile development. 2013.

[7] Paul M Duvall. *Continuous integration*. Pearson Education India, 2007.

[8] Dominik Franke, Stefan Kowalewski, and Carsten Weise. A mobile software quality model. In *Quality Software (QSIC), 2012 12th International Conference on*, pages 154–157. IEEE, 2012.

[9] Dominik Franke and Carsten Weise. Providing a software quality framework for testing of mobile applications. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 431–434. IEEE, 2011.

[10] John D Garofalakis, Antonia Stefani, Vasilios Stefanis, and Michalis Nik Xenos. Quality attributes of consumer-based m-commerce systems. In *ICE-B*, pages 130–136, 2007.

[11] Google. Android 6.0 change. https://developer.android.com, 2015.

[12] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, 41(4), 2008.

[13] Selim Ickin, Katarzyna Wac, Markus Fiedler, Lucjan Janowski, Jin-Hyuk Hong, and Anind K Dey. Factors influencing quality of experience of commonly used mobile applications. *IEEE Communications Magazine*, 50(4), 2012.

[14] ionic. Ionic home page. http://ionicframework.com/, 2016.

[15] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24. IEEE, 2013.

[16] Renaud Pawlak, Carlos Noguera, and Nicolas Petitprez. *Spoon: Program analysis and transformation in java*. PhD thesis, Inria, 2006.

[17] Joachim Perchat, Mikael Desertot, and Sylvain Lecomte. Common framework: A hybrid approach to integrate cross-platform components in mobile application. *Journal of Computer Science*, 10(11):2165, 2014.

[18] Ken Schwaber and Jeff Sutherland. The scrum guide (2013). *http://www. scrum. org/Scrum-Guides¿. Acessado em*, 16:18, 2013.

[19] Kishori Sharan. Model-view-controller pattern. In *Learn JavaFX 8*, pages 419–434. Springer, 2015.

[20] Artem Syromiatnikov and Danny Weyns. A journey through the land of model-view-design patterns. In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, pages 21–30. IEEE, 2014.

[21] CMMI Product Team. Cmmi for development, version 1.2. 2006.

[22] Anthony I Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 397–400. ACM, 2010.

[23] Pree Wolfgang. *Design patterns for object-oriented software development*. Reading, Mass.: Addison-Wesley, 1994.

[24] Sobia Zahra, Asra Khalid, and Ali Javed. An efficient and effective new generation objective quality model for mobile applications. *International Journal of Modern Education and Computer Science*, 5(4):36, 2013.

[25] Dave Zubrow. Software quality requirements and evaluation, the iso 25000 series. *Software Engineering Institute, Carnegie Mellon*, 2004.

Scalable Light-Weight Peer-to-Peer Risk Communication Framework for Critical Infrastructures Management

Titus Okathe, Khalil El-Khatib, Stephen Marsh and Shahram S. Heydari
Faculty of Business and IT
University of Ontario Institute of Technology
Oshawa, Canada
{Titus.okathe, Khalil.el-khatib,Stephen.marsh, shahram.heydari}@uoit.ca

Tim Storer
University of Glasgow
School of Computing Science
Glasgow, Scotland
timothy.storer@glasgow.ac.uk

***Abstract*—Critical infrastructures are growing in scale and complexity and are becoming increasingly interdependent on one-another. This paper argues that existing centralized methods in monitoring and management are unlikely to be sustainable as this trend continues. To address this challenge, this paper presents a complementary model of critical infrastructure monitoring, management and inter-infrastructure communication. The model leverages the advantages of a distributed peer-to-peer method of communication amongst artifacts within infrastructures to provide a scalable, flexible and light-weight means of communication, interaction, and awareness.**

Keywords- critical infrastructures; communication model; interdependence

I. Introduction

Situational awareness has always proven to be extremely important for the management and operation of any system, and especially in the case of critical infrastructures (CI) [1,2,3,4]. To build this situational awareness, operators of CIs collect data from their own systems as well as from other system operators. However, the growing independencies between CIs means that their interactions can be characterized as that of a system-of-systems in which no one entity has overall control or even a global view of the entire system [5]. As a consequence, each infrastructure owner is dependent on peer infrastructures to provide information about the status of facilities or services on which it is dependent for successful operation. These other operators can be from within the same sector, as in the case of the Union for the Co-ordination of Transmission of Electricity (UCTE) or from a different sector, as is the case with the EDXL [6].

Whilst existing frameworks for information exchange can assist with providing situational awareness for CI operators, there are still some problems that can hinder the task of building a more comprehensive picture of situations faced by operators. These include:

1. Data exchange between CI operator is always based on existing collaboration, and does not allow for spontaneous exchange;
2. Data exchange is always carried out at the infrastructure level (between command and control centers), and invariably does not allow individual nodes from one infrastructure to talk to a nodes in different infrastructure;
3. Data exchange still involves a human in the loop, usually the CI operator, to scan through the collected data, who, based on their understanding of the effect of the information on other infrastructures, decides whether or not to pass on the information to other operators;
4. Lack of a simple, common language to express risk or status information[7].

Critical infrastructures are increasing in scale, complexity and interdependence, magnifying these challenges. As a consequence, there is a need to develop flexible, scalable and autonomic mechanisms for exchanging information at appropriate levels of detail in a timely manner across infrastructures. In this paper we propose a risk/status communication framework that abstracts the detailed descriptions of pertinent risks as a *statement of infrastructure artifact comfort.* We explain how this model provides a light-weight means for effectively communicating risks at an appropriate level of abstraction across heterogeneous, legacy infrastructures.

The rest of this paper is organized as follows: Section 2 reviews existing work on inter-infrastructure communication and coordination. Section 3 presents the proposed peer-to-peer model and outlines different models of inter-infrastructure communication that can be adopted for different circumstances. Section 4 evaluates the proposed model and Section 5 draws conclusions and presents the next steps in the research.

II. Background

There is an extensive literature on the modeling, monitoring and management of critical infrastructures [8], protection tools, as well as mechanisms for facilitating effective information exchange [9]. Fundamentally, the purpose of exchanging information among critical infrastructures is to improve their reliability and safety. In light of this there has been research in the area of how to quantify risk; how to represent risk across infrastructures; and the development of suitable information architecture to support these mechanisms amongst heterogeneous systems.

Hu *et al.* [10] and Algirdas *et al.* [11] propose a framework for describing risk by looking at the concepts of dependability and security. The proposed framework combines the attributes of dependability and security and they include: availability, reliability, safety, integrity, maintainability, confidentiality, authenticity, and non-repudiation.

MICIE [12] has the objectives of (1) design of CI modeling techniques which can help in the modeling of the effects of undesired events occurring in a given CI on the Quality of Service (QoS) of its services as well as those of interdependent CIs, (2) design and implementation of an infrastructure for secure cross CI information sharing and mediation, and (3) design and implementation of a MICIE on-line risk prediction tool that encompasses the CI modeling. MICIE also uses a Service Quality Descriptor (SQD) data structure to exchange information between interdependent CI's [13].

Several research groups have investigated techniques for modeling infrastructure interdependencies, highlighting the challenge of presenting dependencies in a uniform manner. Beccuti *et al.* [14] described the CRUTIAL project which employs a Petri-net like approach to modeling systemic effects of individual dependency failures in multiple critical infrastructures. Klein [15] and Klein *et al.* [16] describe the Integrated Risk Reduction of Information-based Infrastructure Systems (IRRIIS) project, which integrates models from a variety of heterogeneous infrastructures in order to analyze their interdependencies. Several other techniques have also been applied to understanding dpendencies in critical infrastructures, such as systems-of-systems modeling [17] and agent based simulations [18].

Several research efforts are underway to facilitate effective and timely information exchange between CIs. The Critical Infrastructure Warning Information Network (CIWIN)[19] is part of the effort by the European Union (EU) to build a secure network for the exchange of critical infrastructure alerts and warnings among EU member states. CIWIN "…offers an efficient and rapid alternative to often time-consuming methods of searching for information, i.e. create a type of "one-stop-system" to obtain all relevant information on Critical infrastructures in the EU"[15]. Additionally, CIWIN "offers the possibility to Member States to communicate directly and upload information that they deem relevant". However, there have been concerns as to the relevance of such a platform given that many of the member states already have Rapid Alert Systems (RAS) of their own which can already perform the functions proposed by the CIWIN [20].

Separately, in [21], Flentge *et al.* present a language for exchanging information across CIs called the "Risk Management Language" (RML). RML is developed around the idea of analyzing CIs using the Implementation-Service-Effect Metamodel (ISE)[22]. RML is an XML based and is therefore extensible. It divides the messages exchange by CIs into three (3) groups:

– Information messages: used to provide information to the service consumer about the possibility of service degradations, as well as any time span and their location.
– Negotiation messages: used by the service provider and the service consumer to exchange and negotiate terms of service delivery.
– Administrative messages: used to control the message exchange.

RML has been tested within the context of the IRRIIS project [23]. Other techniques have also been proposed for extending this work to the autonomic management of interactions between and within infrastructures. Gustavsson and Ståhl described the work on applying self-healing techniques to critical infrastructures in the INTEGRAL project [24]. Hall-May *et al.* [25] and Krrüger *et al.* [26] have separately advocated the use of a service oriented architecture approach to integrating infrastructure management systems.

III. PROPOSED MODEL

This paper proposes a novel approach to critical infrastructure monitoring, management and inter-communication. The proposed model leverages a decentralized agent based, peer-to-peer architecture in which individual artifacts in different critical infrastructures are able to interact directly with others via a variety of communication models. This contrasts with conventional models of critical infrastructure inter-management, in which each entire infrastructure is treated as an agent, service or other computational entity and where communication only occurs between centralized control centers.

In our proposed model, an infrastructure is represented as a collection of agents, with each agent representing some artifact in an infrastructure. For example, consider a fictional modern city comprising many infrastructures such as:

- An electrical power supply infrastructure consisting of electricity consumers, generating facilities, sub-stations, pylons and cabling.
- The water supply consisting of pipes, reservoirs, filtration plants, pumps and water consumers.
- The road network, comprising road lanes, intersections, traffic signals and vehicles.
- The telecommunications network, comprising switches, servers, end-user communication devices, wireless and mobile network base stations and cabling.
- An underground railway network consisting of train sets, rail links and stations.

All of these infrastructures are interdependent on the state of each other. A water pump, for example may depend on power supplied by the electricity infrastructure. On the other hand, a nuclear power station may depend on a ready supply of water to act as a coolant.

In the proposed model, each of the artifacts (road lane, railway station, vehicle and so on) is represented as an agent. Figure 1 illustrates an example of the architecture for three infrastructures: power supply, telecommunications and transportation. Each infrastructure is shown as a circle containing a number of artifacts represented as agents.

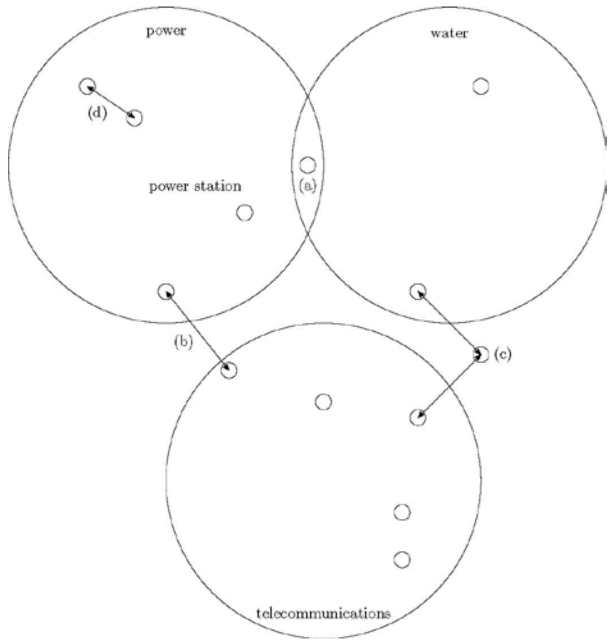


Figure 1. Peer to peer model of inter-infrastructure artifact communication.

We anticipate that agents within the same infrastructure will be able to communicate with each other directly, using a variety of specialized messages and protocols that suit the specific needs of the infrastructure. In particular, we assume that most infrastructures will continue to maintain a centralized control center that will receive status information communicated from infrastructure artifacts, as well as issue commands. However, the arrangement of communication between agents within an infrastructure is an infrastructure dependent issue and not considered further here. Each infrastructure has very different technical characteristics requiring tailored monitoring and management systems. Consequently, we presume that each infrastructure owner will adopt information and communication technologies for infrastructure monitoring, management and internal communication that suits their own needs. This allows each infrastructure owner to continue to use a heterogeneous range of legacy ICTs for infrastructure management as they see fit.

Separately, inter-infrastructure communication is enabled by permitting agents from different infrastructures to also communicate with each other. This allows the communication of information across infrastructure inter-dependencies directly between relevant infrastructure artifacts, rather than via central control centers. For example, a water pump in the water supply infrastructure can be informed of a pending shortfall in the power supply by a nearby power system artifact, giving it time to either reduce the amount of power it requires to operate (by moving to a more efficient but less capable mode, for example) or transition to a safe state for temporary shut-down.

A challenge here is the vast array of infrastructure status information that may be pertinent to a dependent infrastructure artifact. Deciding which information must be communicated and in what format so that it can be understood by a peer infrastructure. This diversity reflects the different physical systems that have to be managed in different infrastructures, and the priorities for measuring different characteristics. For example, water flow rates, reservoir levels and purity may be important characteristics for a water supply infrastructure. However, a nuclear power station may only need to be alerted if the level of water in its coolant reservoir drops below a certain critical minimal level.

As described in Section 2, several research efforts are underway to develop standardized means of communicating this information between infrastructures that may be managed using a heterogeneous range of ICTs; and Genc *et al*. describe the application of a service based publish and subscribe software architecture to the problem of information distribution[27]. However, neither of these approaches addresses the general need to provide a holistic over-view of the status of a critical infrastructure to peers in a flexible and scalable manner.

The model proposed in this paper employs a different approach, by *abstracting* the detailed status information that is specific to a particular infrastructure artifact as an overall sense of *comfort* in infrastructure artifact. The concept of computational comfort has previously been employed in the management of user-personal device interactions in order to provide a more flexible and context adaptable security environment. [28,29,30] In this previous work, a personal device (such as a smartphone or tablet) would continually evaluate its sense of comfort based on a range of factors, such as the user's actions, data accessed, connected services and networks, physical location and time. The device can then adjust its security posture as well as deter less desirable actions based on its overall sense of comfort. For example, a user accessing personal family photographs at home may enhance a device's sense of comfort (because this is a familiar activity). However, performing the same action in a public place or work environment may cause the device comfort level to drop. In this situation, the device would begin to resist (but not prevent) the users action in order to communicate the sense of discomfort as a warning that the actions may be inappropriate.

A range of factors may contribute to the sense of comfort of an agent in a critical infrastructure, depending on the nature of the underlying artifact. Some examples are:

- Flow rates on a reservoir supply pipe.
- Power fluctuations on an electrical line.
- Average vehicle speed on a road link.
- Congestion on a road link, or frequency of traffic signal changes at an intersection.

The computation of an individual infrastructure node's comfort is therefore specific to that node. However, the node (agent) can use the computed sense of comfort to communicate in an abstract manner about potential problems within the infrastructure to dependent artifacts (nodes) in other infrastructures.

The inter-infrastructure agent communication may occur according to several different models of interaction, depending on the relationship between the respective infrastructure owners and the nature of the underlying artifacts. The different models of communication are shown in Figure 1:

- A shared agent, with a presence in both of the communicating infrastructures, labeled (a) in Figure 1. A shared agent receives communications directly from agents in all of the infrastructures it resides in. This arrangement reflects a situation where two infrastructure owners have a significant amount of trust in one another. The agent's level of comfort is computed from the infrastructure specific factors of all the infrastructures it resides in.
- Direct agent to agent communication, labeled (b) in Figure 1. This represents a medium level of trust between two infrastructures. The agents are able to inform each other directly about their current level of comfort. This peering between agents represents a situation where one agent represents an artifact that is dependent on the performance of its peer.
- Communication mediated by an agreed independent third party, labeled (c) in Figure 1. This arrangement represents the lowest level of trust between two infrastructures. The agents in the peer infrastructures' communication is mediated by an agreed independent third party. This may be in order to prevent direct access between agents, for example, to permit anonymous communication of information, or to filter messages. This model is analogous to information security coordination centers that have been established in several jurisdictions for industry specific incident reporting.

The selection of the appropriate form of inter-agent communication is a design decision that will depend on the relationship between infrastructure owners and the nature of the underlying infrastructures. The use of shared agents between infrastructures allows for a closely integrated sense of comfort that allows agents in both infrastructures to respond directly to problems. However, this model assumes a willingness of infrastructure providers to 'share' control of artifacts within their infrastructures and may not be appropriate in all cases. Mediated communication can provide for information that is more limited and anonymous, but can make this information less useful (an agent may not be able to determine which peer is causing the mediator to report discomfort). The middle case provides for a compromise situation in which direct communication of comfort is permitted between certain peer artifacts in an infrastructure.

A final aspect of the proposed model is that agents in one infrastructure may also comprise a number of agents in a critical infrastructure themselves. In this situation, an aggregated agent presents an overall comfort level for the underlying infrastructure. In Figure 1, the power station agent in the power supply infrastructure could be a large complex system, comprising many supporting infrastructure artifacts. However, the overall status of the power station can be abstracted for the purpose of communication to peer artifacts in the power supply infrastructure.

IV. Evaluation of the Proposed Model

In the context of disaster management, Genc *et al*. have argued that the challenges in information distribution in critical infrastructures include **interoperability**, **timeliness**, **security**, **flexibility** and **adaptability**, due to the evolutionary nature of the set of participants [23]. Considering the proposed model against these criteria:

- **Interoperability**: the model imposes minimal new standards on the implementation of CI management systems. Each infrastructure is able to decide for itself which artifacts should be enabled to express comfort levels to peer artifacts. In addition, the computation of comfort levels for a given artifact is left to the infrastructure owner. This leverages the expertise in each infrastructure and minimizes the need for cross-infrastructure communication.
- **Timeliness**: The three models of artifact interaction described in Section 3 provide for real time (type *a* or *b*) or mediated communication (type *c*) as appropriate to the situation between two infrastructures. In addition, the communicated information is abstracted away from the details of the infrastructure, enabling infrastructure artifacts to respond rapidly to changing contextual information.
- **Security**: The different communication models proposed in Figure 1 allow an infrastructure owner to customize their interactions with peer infrastructures based on perceived security risks. Mediated communication can provide a firewall between infrastructures where there is a desire for indirect communication (for anonymity or confidentiality purposes (for example).
- **Flexibility**: The proposed model provides for considerable variation in adoption for infrastructure owners. A system architect is able to select which artifacts in an infrastructure act as agents able to express comfort, as well as selecting which agents they will communicate with in other infrastructures and in what way. In addition, the hierarchical composition of infrastructures allows an
- **Adaptability:** Depending on the situation, the model allows critical infrastructure providers to exchange information with whomever they deem important in the current situation, and without the need for lengthy relationship set-up process.

V. Conclusion and Future Work

We present a new, complementary model for the communication of infrastructure awareness within and between Critical Infrastructures. The model is lightweight, and uses the concept of *comfort*, itself a subjective measure of potential security or risk tolerance, to allow individual artifacts (nodes) within infrastructures, represented by autonomous agents, to make informed, self-aware judgments of ongoing real-time situations.

Currently, the model is abstract and has not been fully implemented, although we have implementations of infrastructure awareness and modeling using Esri's ArcGIS system. It is our intent to take this model and develop it into a working system for Critical Infrastructures, and couple it with our ongoing work in the area of Infrastructure Awareness and Augmentation.

REFERENCES

[1] Commission of the European Communities: Communication from the Commission on a European Programme for Critical Infrastructure Protection, COM(2006) 786 final, 2006.

[2] European Commission Information Society and Media Directorate-General: Availability and Robustness of Electronic Communications Infrastructures The ARECI Study Final report, March 2007.

[3] J. Yoon, S. Dunlap, J. Butts, M. Rice, and B. Ramsey, "Evaluating the readiness of cyber first responders responsible for critical infrastructure protection," International Journal of Critical Infrastructure Protection, 13 , 2016, pp.19-27.

[4] A. Farouk "Critical Infrastructure Protection in Developing Countries," Handbook of Research on Economic, Financial, and Industrial Impacts on Infrastructure Development. IGI Global, 2017, pp.23-39.

[5] J. Boardman and B. Sauser, "System of Systems: The Meaning of," the IEEE/SMC International Conference on System of Systems Engineering, 2006, pp. 118-123.

[6] M. Raymond, S. Webb, and P.I. Aymond, "Emergency Data Exchange Language (EDXL) Distribution Element," v. 1.0 OASIS Standard EDXL-DE v1.0, 1, May 2006.

[7] Union for the Co-ordination of Transmission of Electricity (UCTE): Final Report, System Disturbance, 2006.

[8] M. Alam and K.A. Shakil, "A decision matrix and monitoring based framework for infrastructure performance enhancement in a cloud based environment," Advances in Engineering and Technology Series, Elsevier 7, pp.147-153, 2014.

[9] J. Parajuli and K. E. Haynes. "Transportation Network Analysis in Nepal: A Step toward Critical Infrastructure Protection," 2016.

[10] J. Hu, P. Bertok and Z. Tari, "Taxonomy and Framework for Integrating Dependability and Security," Information Assurance: Dependability and Security in Networked Systems, Elsevier, 2008, pp. 149-170.

[11] A. Aizienis, J. -C Laprie, B. Randell, and C. Landdwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11-33.

[12] S. De Porcellinis, G. Oliva, S. Panzieri, and R. Setola, "A Holistic-Reductionistic Approach for Modeling Interdependencies," Critical Infrastructure Protection III, C. Palmer and S. Shenoi (Eds.), vol. 311, 2009, pp. 215-227, Springer AICT.

[13] M. Aubigny, C. Harped, and M. Castrucci "Risk ontology and service quality descriptor shared among interdependent critical infrastructures," Critical Information Infrastructures Security, Springer, 2011, pp. 157-160.

[14] M. Beccuti, G. Franceschinis, M. Kaâniche, and K. Kanoun, "Multi-level dependability modeling of interdependencies between the Electricity and Information Infrastructures," Int. Workshop on Critical Information Infrastructures Security (CRITIS09), volume 5508 of Springer LNCS, 2008, pp. 48-59, Frascati (Rome), Italy.

[15] R. Klein, "Information Modelling and Simulation in Large Dependent Critical Infrastructures. An Overview on the European Integrated Project IRRIIS," the 3rd International Workshop on Critical Information Infrastructures Security, CRITIS 2008, Rome, Italy, October 2008, LNCS 5508, Springer, Berlin.

[16] R. Klein, E. Rome, C. Beyel, R. Linnemann, W. Reinhardt, and A. Usov, "Information Modelling and Simulation in Large Interdependent Critical Infrastructures in IRRIIS," the 3rd International Workshop on Critical Information Infrastructures Security, CRITIS 2008, Rome, Italy, October 2008, LNCS 5508, Springer, Berlin.

[17] W. Tolone, "Making Sense of Complex Systems Through Integrated Modeling and Simulation," Advances in Information and Intelligent Systems, volume 251 of Studies in Computational Intelligence, Springer, 2009.

[18] E. Casalicchio, E. Galli, and S. Tucci, "Modeling and Simulation of Complex Interdependent Systems: A Federated Agent-Based Approach," CRITIS 2008, pp. 72-83.

[19] Commission of the European Communities, "European Commission," [Online]. Available: http://ec.europa.eu/governance/impact/commission_guidelines/docs/sec_2008_2701_ia_ciwin_en.pdf. [Accessed June 2017].

[20] Commission of the European Communities, "CIWIN," [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2006:0786:FIN:EN:PDF. [Accessed June 2017].

[21] F. Flentge, C. Beyel and E. Rome, "Towards a standardised cross-sector information exchange on present risk factors," Critical Information Infrastructure Security, Springer, 2008, pp. 349-360.

[22] F. Flentge and U. Beyer, "The ISE metamodel for critical infrastructure," Critical Infrastructure Protection, Springer, 2007, pp. 323-326.

[23] R. Klein, "The EU FP6 Integrated Project IRRIIS on Dependent Critical Infrastructures," Critical Information Infrastructures Security, Springer, 2011, pp. 26-42.

[24] R. Gustavsson and B. Ståhl "Self-healing and Resilient Critical Infrastructures," Critical Information Infrastructure Security, Third International Workshop, CRITIS 2008, Rome, Italy.

[25] M. Hall-May and M. Surridge, "Resilient Critical Infrastructure Management Using Service Oriented Architecture," CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems.

[26] I.H. Krrüger, M. Meisinger, M. Menarini, and S. Pasco, "Rapid Systems of Systems Integration - Combining an Architecture-Centric Approach with Enterprise Service Bus Infrastructure." Proc. IRI'06, IEEE Systems, Man, and Cybernetics Society, Sep. 2006, pp. 51-56.

[27] Z. Genc, F. Heidari, M.A. Oey, S.van Splunter, and F.M.T. Brazier, "Agent-Based Information Infrastructure for Disaster Management." Springer Berlin Heidelberg, 2013, pp.349-355.

[28] S. Marsh, P. Briggs, K. El-Khatib, B. Esfandiari, and J.A. Stewart, "Defining and investigating device comfort." Journal of Information Processing 19, 2011, pp. 231–252.

[29] S. Marsh, S. No¨el, T. Storer, Y. Wang, P., Briggs, L. Robart, J. Stewart, B. Esfandiari, K. El-Khatib, M.V. Bicakci, M.C. Dao, M. Cohen, and D.D. Silva, "Non-standards for trust: Foreground trust and second thoughts for mobile security." Proceedings STM 2011, Springer.

[30] T. Okathe, S.S.Heydari, V. Sood, O. Cole, and El-Khatib K, "Middleware For Heterogeneous Critical Infrastructue Networks Intercommunication," International Journal on Smart Sensing and Intelligent Systems, vol. 9.3, 2016 , pp. 1261-1286.

Reliable, Context-Aware and Energy-Efficient Architecture for Wireless Body Area Networks in Sports Applications

Richard Jaramillo, Alejandro Quintero, Steven Chamberland
Computer Engineering Departement
Polytechnic School of Montreal
Montréal, Canada
email: richard.jaramillo@ieee.org, alejandro.quintero@polymtl.ca, steven.chamberland@polymtl.ca

***Abstract*—Reliability, Context awareness and Energy efficiency are some of the most important requirements for Wireless Body Area Networks (WBANs). The proposed architecture for WBANs puts together a Media Access Control (MAC) protocol, a Transport Protocol and a Rate Control scheme proposed in our previous work. It proposes the use of three extra bits within each beacon period for indicating whether the hub is going to assign new slot Reallocations to the nodes, to request for lost packet Retransmissions from the nodes, to send Rate-control requests to the nodes, or any combination of them. Some experimental results are presented in order to validate the requirements and a comparison with other architectures is made using the main detected requirements for WBANs.**

Keywords–reliability; context awareness; energy efficiency; congestion control; loss recovery, WBAN architecture.

I. INTRODUCTION

Information collected from body sensors in a WBAN is sent to a hub or coordinator, which processes the information and can also perform other functions, such as managing body events, merging data from sensors, sensing other parameters, performing the functions of a user interface and bridging the WBAN to higher-level infrastructure and other stakeholders.

The IEEE Standard for Local and Metropolitan Area Networks - Part 15.6: Wireless Body Area Networks (2012) specifies short-range and wireless communications in the vicinity of, or inside a human body (although it is not limited to humans). It categorizes WBAN applications into medical and non-medical [1]. Some examples of medical applications might be: sleep staging, diabetes control and monitoring of cardiovascular diseases. Some examples of non-medical applications might be: entertainment, sports and military operations.

The design of a WBAN implies the tackle of several challenges. Among the most important challenges, we can find: (i) the energy efficiency of the whole network, which may require new MAC protocols, new routing protocols and new energy scavenging sources; (ii) the consideration of the impact of data loss, which may require additional measures in order to ensure the Quality of Service (QoS); (iii) the reliability of the network to grant accuracy and to guarantee on-time delivery of data; (iv) the context awareness for responding according to the current situation in the network.

The objective of this paper is to design an architecture for WBANs, which tries to tackle these main challenges. The architecture will be composed of three main components: (i) A context-aware and energy-efficient mechanism for providing QoS in WBANs; (ii) A reliable and energy-efficient mechanism to provide packet loss recovery and fairness in WBANs; and (iii) A context-aware rate control scheme to provide congestion control in WBANs.

Sports WBANs have a different behavior compared to other WBANs. In a Sports WBAN, most of the packets are periodic with a small portion of emergency packets that need to be delivered with low latency. A reliable, context-aware and energy-efficient architecture for WBANs used in sports applications is proposed, facing four challenges: energy efficiency, context awareness, QoS and reliability.

Several architectures for WBANs have been proposed recently in the literature. Some of them are mentioned in Section II. The requirements of the proposed architecture are summarized in Section III. Section IV presents all the protocols and schemes, the general and the node architectures, the global topology, the phases in the beacon period, and the node behavior. Some experimental results are presented in Section V for demonstrating the requirements accomplishment. A comparison of some architectures with the proposed architecture is made in the Section VI. Section VII presents the conclusion and future work.

II. RELATED WORK

Wang et al. [2] have proposed a configurable quantized compressed sensing (QCS) architecture, in which the sampling rate and quantization configuration are explored together for improving the energy efficiency. A rapid configuration algorithm has been developed to locate the optimal configuration of the sampling rate and the bit resolution, always monitoring low energy consumption, reducing the elapsed time while keeping an excellent efficiency and capacity. However, loss recovery was not mentioned.

In the following references, energy efficiency was not contemplated. Felisberto et al. [3] proposes a WBAN architecture to recognize human movement, identify human postures and to detect harmful activities in order to prevent risks. The architecture proposal comprises five basic components. (i) Sensor node – responsible for acquiring data of inertial and physiological sensors and transmitting them to the Coordinator node. (ii) Coordinator node – responsible for serving as a forwarder of data gathered by the Sensor nodes. (iii) Gateway node – it is the interface between the WBAN and the network that provides the Internet connection. (iv) Mobile node – an alternative interface used when the Gateway Node or Internet connection are not available. (v) Control center – responsible for the registration and post processing of the motion events sent by the Sensor nodes.

Almashaqbeh et al. [4] have proposed a Cloud-based real-time remote Health Monitoring System (CHMS) for tracking the health status of non-hospitalized patients while they perform their daily activities. The system tries to provide high QoS and to focus on connectivity-related issues between the patients and the global cloud. The CHMS design includes four basic components: Wireless routers, Gateways, WBANs and Medical staff.

Domingo [5] has proposed a context-aware service architecture for the integration of WBANs and social networks through the IP Multimedia Subsystem (IMS). In this architecture, multimedia services are accessed by the user from several wireless devices via an IP or cellular network based on the vital signs monitored in a WBAN. The architecture is divided into four layers: (i) Device layer – the sensors communicate with the gateway using ultra-wideband (UWB) or the IEEE 802.15.6 standard, and the gateway communicates with the monitoring station using Bluetooth or ZigBee. (ii) Access layer – responsible for the access of the monitoring stations to the radio channel. (iii) Control layer – it controls the authentication, routing and distribution of IMS traffic. (iv) Service layer - used to store data, execute applications, or provide services.

Wan et al. [6] have proposed a framework for a pervasive healthcare system with Mobile Cloud Computing (MCC) capabilities. This system is composed of four main components: (i) WBANs – which collect various vital signals, such as body temperature or heart rate information from wearable or implantable sensors. (ii) Wired/Wireless transmission. (iii) Cloud services – which possess powerful Virtual Machine resources, such as CPU, memory, and network bandwidth in order to provide all kinds of cloud services. (iv) Users – such as hospitals, clinics, researchers, and patients.

Kartsakli et al. [7] cites a remote monitoring scheme that provides ubiquitous connectivity for mobile patients. A patient-attached monitoring device collects the WBAN data, classifies them as high-priority (e.g., critical data such as blood pressure, pulse rate and heart rate) or normal priority (e.g., ECG signals) and forwards them towards the healthcare provider through a heterogeneous WiFi/WiMAX access communication network.

Kartsakli et al. [7] also cites a three-tier network architecture for the remote monitoring of elderly or chronic patients in their residence. The lower tier consists of two systems: (i) a patient-worn fabric belt, which integrates the medical sensors and is equipped with a Bluetooth transceiver; and (ii) the ambient wireless sensors that form a ZigBee network and are deployed in the patient's surroundings (e.g., in the patient's home or in a nursing house). In the middle tier, an ad hoc network of powerful mobile computing devices (e.g., laptops, PDAs, etc.) gathers the medical and ambient sensory data and forwards them to the higher tier. The middle-tier devices must have multiple network interfaces: Bluetooth and ZigBee to communicate with the lower tier and WLAN or cellular capabilities for connection with the higher layer. Finally, the higher tier is structured on the Internet and includes the application databases and servers that are accessed by the healthcare providers.

Kartsakli et al. [7] also cites a system architecture based on two independent subsystems for the monitoring and location tracking of patients within hospital environments. The healthcare monitoring subsystem consists of smart shirts with integrated medical sensors, each equipped with a wireless IEEE 802.15.4 module. The location subsystem has two components: (i) a deployment of wireless IEEE 802.15.4 nodes that are installed in known locations within the hospital infrastructure and broadcast periodic beacon frames; and (ii) IEEE 802.15.4 end devices, held by the patients, that collect signal strength information from the received beacons. Both subsystems transmit their respective data (i.e., medical sensory data and signal strength information) to a gateway through an IEEE 802.15.4-based ad hoc distribution network.

III. MAIN REQUIREMENTS

The selected architecture should offer to the hub and each node within the WBAN some specific advantages. The main requirements of the proposed architecture can be enumerated like:

- Energy Efficiency: to minimize the power consumption avoiding or mitigating collisions, idle listening, overhearing, and control packet overhead. The proposed architecture should decrease contention-based transmissions and increase the sleeping time for each node.
- Reliability: to assure the end-to-end packet delivery between the sensor nodes and the hub. The proposed architecture should allow the nodes to be able of sending all their emergency and normal packets to the hub.
- QoS: to have the ability to deliver packets with the least latency and the highest throughput. The proposed architecture should allow the trade-off between the energy efficiency and the desired reliability of the WBAN.
- Congestion Control: ability to control traffic in the WBAN in order to avoid packet collision and buffer overflow. The proposed architecture should decrease the packet loss due to the packet collision and both normal and emergency buffer overflow.
- Rate Control: ability to prevent the nodes from overwhelming the hub and the whole WBAN. The proposed architecture should allow the hub to control the packet rate of the sensor nodes in order to keep an average rate in the whole WBAN.
- Loss Detection: ability to detect the lost packets in the hub side and in each node side. The proposed architecture should provide the early detection of lost packets on both sides: the hub and the sensor nodes.

- Loss Recovery: ability to make the lost packet retransmission requests and to send the corresponding packet retransmissions. The proposed architecture should decrease the total number of lost packet through the retransmission of some of them.
- Fairness: ability to distribute the network resources equitably among all nodes of the WBAN. The proposed architecture should allow all nodes to get equal access to the network and give the corresponding priority to those nodes with emergency traffic and with high packet rate.
- Emergency Awareness: ability to respond to any emergency event in any node at any time. The proposed architecture should be able to detect early any emergency event and to give the corresponding priority to the emergency nodes.
- Context Awareness: ability to respond to any alert (high buffer, low battery, emergency) in any node at any time. The proposed architecture should be able to detect high buffer levels, low battery levels, and any emergency event into the nodes.

IV. PROPOSED ARCHITECTURE

This section explains the proposed architecture that gathers all the protocols and schemes presented in the previous work by Jaramillo et al. [8]-[10]. The section presents the proposed phases for each beacon period. Then, it summarizes all the protocols and schemes, depicts the general and the node architectures, and the global topology. Finally, the section explains the operation of the R's Indicator Bits (RIBs) scheme for each beacon period and the overall node behavior.

A. *Phases in the Beacon Period*

Figure 1 depicts the three proposed phases within the beacon period. The first phase is called Reallocation, Retransmission & Rate-control Phase (3RP). It is used by the hub for sending slot reallocations, retransmission requests and the Rate Control Factor (RCF) to all nodes. The second phase is Managed Access Phase (MAP). It is used by all nodes for sending normal and emergency traffic, always giving the highest priority to the emergency traffic. The third phase is called Special Contention Access Phase (SCAP). All nodes use SCAP for sending connection requests and additional normal and emergency traffic.

There is no contention during 3RP due to the use of the Time Division Multiple Access (TDMA) protocol. The hub uses all slots for sending slot reallocations and the RCF.

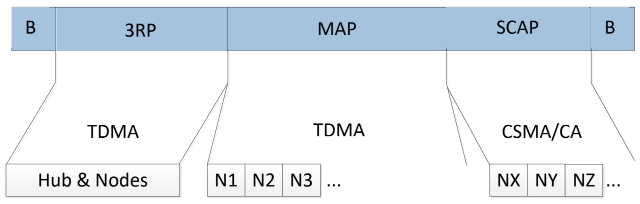


Figure 1. Phases in the Beacon Period.

The nodes only send lost packet retransmissions after they have received a lost packet retransmission request from the hub. MAP also uses the TDMA protocol for transmitting normal and emergency traffic. The use of Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) in SCAP implies contention-based transmission during this phase. At least one phase needs to offer contention to allow the unconnected nodes to connect to the WBAN.

B. *Protocols and Schemes*

The energy-efficient and emergency-aware MAC protocol for WBANs proposed by Jaramillo et al. [8] is based on the existing MAC protocol described in the IEEE 802.15.6 standard, but with some modifications in the access phases and the access methods for each beacon period in order to provide more emergency awareness while keeping energy efficiency. The proposed MAC protocol outperformed the IEEE 802.15.6 MAC protocol, the IEEE 802.15.4 MAC protocol and the Timeout-MAC (T-MAC) protocol in the percentage of emergency and normal packet loss and latency, while maintaining similar energy consumption as the IEEE 802.15.6 MAC protocol. In the MAC layer, the hub uses the Slot Reallocation Algorithm depicted in Figure 2 to create the slot reallocations for all nodes.

```
1: If Need Reallocations then
2:   Set Reallocation Indicator Bit ← 1
3:   For each Node in Reallocation_Buffer do
4:     Assign New Reallocation to Node
5:   End For each
6:   Assign Default Allocations to remaining
     nodes
7:   Send Slot Reallocations in the next 3RP
8: End If
```

Figure 2. Slot Reallocation Algorithm.

The reliable transport protocol based on loss recovery and fairness for Sports WBANs proposed by Jaramillo et al. [9] is a cross-layer design that detects out-of-sequence packets and requests retransmission of some lost packets. The hub and each node in the WBAN detect lost packets and make the requests and retransmissions during 3RP. The hub calculates the Fairness Index as the ratio between the number of lost packets and the total number of received packets. The hub uses the Fairness Index to prioritize the request creation in order to provide fairness between all the nodes in the WBAN. It outperformed the IEEE 802.15.6 Standard in the percentage of the packet loss, while maintaining similar energy consumption. The Packet Loss Detection Algorithm is used by the hub and the nodes and it is depicted in Figure 3 and Figure 4, respectively. The hub and the nodes detect lost packets in the MAC layer.

```
If sequenceNumber > lastSequenceNumber + 1 then
  Set Retransmission Indicator Bit ← 1
  Set i ← lastSequenceNumber + 1
  While i < sequenceNumber do
    Create Lost Packet Retransmission Request
    i ← i + 1
  End While
  Send Retransmissions Requests in the next 3RP
End If
```

Figure 3. Packet Loss Detection in the Hub.

```
If Transmissions + Fails = Maximum_Tries or
Current_Packet does not fit in Buffer then
  Set Retransmission Indicator Bit ← 1
  Push Current_Packet in Packet_Lost_Buffer
  Prepare to send Retransmissions in the next
  3RP
End If
```

Figure 4. Packet Loss Detection in the Node.

The rate control scheme for congestion control in Sports WBANs proposed by Jaramillo et al. [10] is context-aware and responses to emergency events in any node, reducing the normal traffic rate. When an emergency event occurs in the WBAN, the hub has to calculate RCF and communicate it to all nodes in the network in order to keep the same average rate of traffic during the entire emergency event. The proposed solution improved the performance of the IEEE 802.15.6 Standard. The Rate Control Scheme depicted in Figure 5 is used by the hub in the MAC layer to calculate the RCF for all nodes, and it is used by the nodes in the application layer for applying the RCF sent by the hub.

```
If Emergency Alert detected then
  Set Rate-Control Indicator Bit ← 1
  Push Current_Node in EmergencyBuffer
  Calculate RCF for all remaining nodes
  For each node not in EmergencyBuffer do
    Create Rate Control Request
  End For each
  Send Rate Control Requests in the next 3RP
End If
```

Figure 5. Congestion Control in the Hub.

C. General Architecture

Figure 6 depicts the proposed general architecture. The hub and the sensors are composed of five modules. (i) Sensing Module (SM) – in charge of sensing body information and detecting alerts into the packets. (ii) Memory Module (MM) – in charge of storing sensing data and lost packets for future retransmissions. (iii) Battery Module (BM) – in charge of detecting low battery levels. (iv) Processing Module (PM) – in charge of processing body information, creation of slot reallocations, detecting lost packets, sending requested lost packets, and processing the RCF for congestion control.

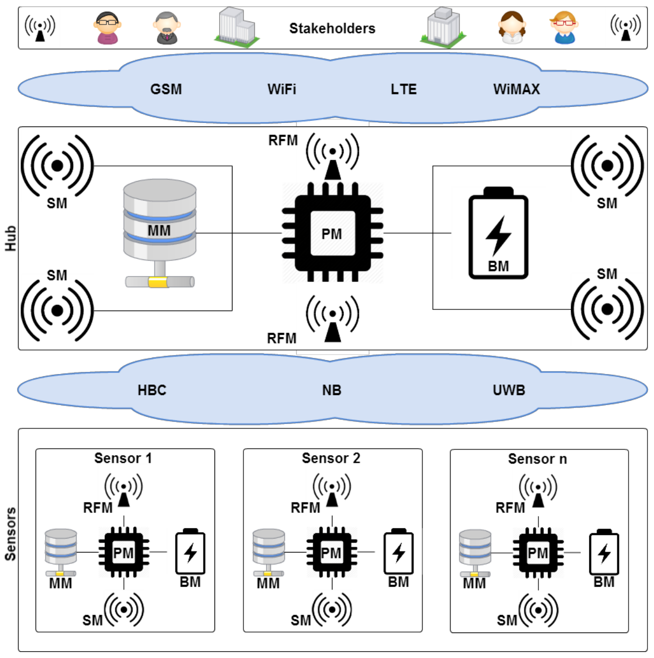


Figure 6. General Architecture.

(v) Radio-Frequency Module (RFM) – in charge of the transmission of the body information between the nodes and the hub (via either Human Body Communication - HBC, Narrow Band - NB, or Ultra-Wideband - UWB). It also manages the communication between the hub and the coach devices, data centers, and other stakeholders (via either Global System for Mobile communication - GSM, Long-Term Evolution - LTE, Wi-Fi, or Worldwide Interoperability for Microwave Access - WiMAX).

D. Node Architecture

Figure 7 depicts the node architecture with its five modules. (i) The SM supports the Slot Reallocation Algorithm with the detection of the alert type into each packet. (ii) The MM supports the Slot Reallocation Algorithm with the detection of future emergency buffer overflow, and supports the Lost Packet Retransmission Algorithm with the buffering of lost packet for future retransmissions. (iii) The BM supports the Slot Reallocation Algorithm with the detection of low battery levels. (iv) The PM supports the Slot Reallocation Algorithm, the Rate Control Scheme with the processing of the RCF, and supports both the Packet Loss Detection Algorithm and the Lost Packet Retransmission Algorithm with the creation of lost packet retransmission requests and the sending of lost packet retransmissions. (v) The RFM supports the Slot Reallocation Algorithm, the Rate Control Scheme and the Packet Loss Detection Algorithm, with the sending of slot reallocations, the RCF, and lost packet retransmissions, respectively.

E. Topologies

The proposed WBAN topology is always a star topology. In this way, the use of a special routing protocol is not needed. The hub must always be in the center.

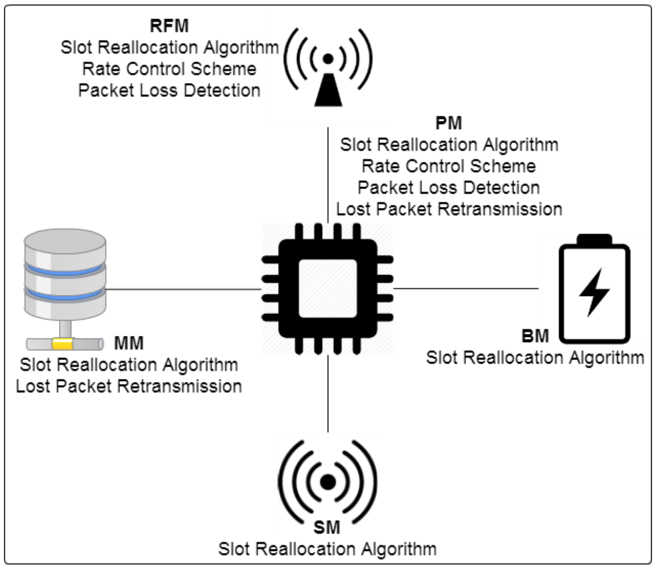

RFM
Slot Reallocation Algorithm
Rate Control Scheme
Packet Loss Detection
PM
Slot Reallocation Algorithm
Rate Control Scheme
Packet Loss Detection
Lost Packet Retransmission
MM
Slot Reallocation Algorithm
Lost Packet Retransmission
BM
Slot Reallocation Algorithm
SM
Slot Reallocation Algorithm

Figure 7. Node Architecture.

The sensor nodes must be around the hub. The hub is in the right hip. There are two sensor nodes over the wrists, two sensor nodes over the ankles and one last sensor node over the chest. Each sensor node has a direct wireless connection with the hub, and there are not relaying nodes for the packet routing. With this direct connection of sensor nodes to the hub, the information takes the least possible delay in transmission. Besides, the failure of a single node does not compromise the remaining nodes.

Figure 8 depicts the global topology for the proposed architecture. After gathering and processing the body information from nodes within the WBAN star topology, the hub can send this information both directly to a coach device (via Wi-Fi, GSM, LTE) or to other stakeholders (via GSM, LTE, Wi-Fi, WiMAX). The coach device can perform additional processing to help the coach to improve the training plan of the sportsman. The stakeholders can see the processed information into the data centers to improve research in training protocols of athletes, and deficiency detection.

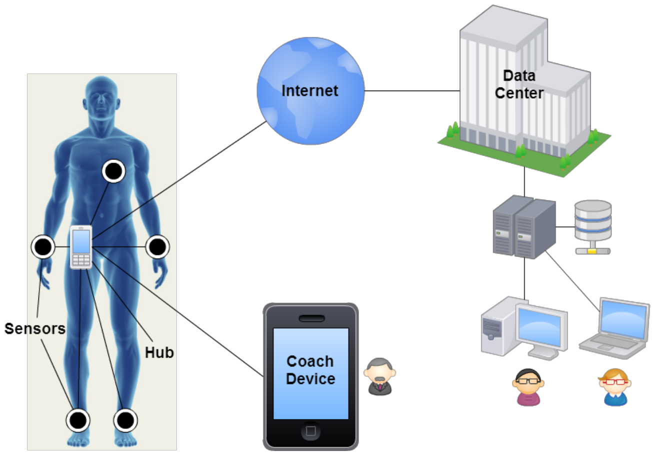


Figure 8. Global Topology.

F. RIBs in the Beacon

Table I shows the configuration of indicator bits into each beacon for the proposed architecture. Only three extra bits were used within each beacon for indicating whether the hub is going to assign slot Reallocations, to request packet Retransmissions, to send Rate-control requests, or any combination of them. When an emergency event happens in any node, the hub receives an alert from the node and it can decide whether the congestion control is necessary and use the Rate-control Indicator Bit (Third Bit). If there are many lost packets detected by the hub, it may use the Retransmission Indicator Bit (Second Bit). The Reallocation Indicator Bit (First Bit) is always used by the hub after an emergency event occurs in any node.

TABLE I. INDICATOR BITS INTO EACH BEACON

Type	3rd Bit	2nd Bit	1st Bit
None	0	0	0
Reallocation	0	0	1
Retransmission	0	1	0
Reallocation & Retransmission	0	1	1
Rate-control	1	0	0
Reallocation & Rate-control	1	0	1
Retransmission & Rate-control	1	1	0
Reallocation, Retransmission & Rate-control	1	1	1

The value of 1 (one) in the Reallocation Indicator Bit, means that the hub is going to send slot reallocations to all nodes in the current beacon period. The value 1 (one) in the Retransmission Indicator Bit means that the hub has detected lost packets and it is going to send packet retransmission requests in the current beacon period. The value 1 (one) in the Rate-control Indicator Bit means that the hub has received an emergency alert and it is going to send a RCF to control the rate inside all normal nodes in the current beacon period. This strategy allows to easily extend the number of indicator bits for new hub behaviors.

G. Nodes Behavior

Figure 9 summarizes the behavior of each node for the proposed architecture. After the beacon reception, the node has two options: it evaluates the RIBs if it is connected or it must wait until SCAP if it is unconnected. When it is unconnected, the node will always send its connection request during the next SCAP.

During 3RP phase (the yellow color zone), the node has previously evaluated the RIBs, then, the node can listen to slot reallocations sent from the hub, or listen to the RCF sent from the hub, or listen to lost packet retransmission requests sent from the hub and finally resend the lost packets requested by the hub.

During MAP phase (the blue color zone), the node sends emergency and normal packets giving the highest priority to the emergency traffic. The node uses its own assigned slots and it might use the remaining slots at the end of MAP if needed. If there were slot reallocations for this beacon period, the node will use the new slot reallocation received, otherwise, it will use the original slot relocation received when it connected to the WBAN for the first time.

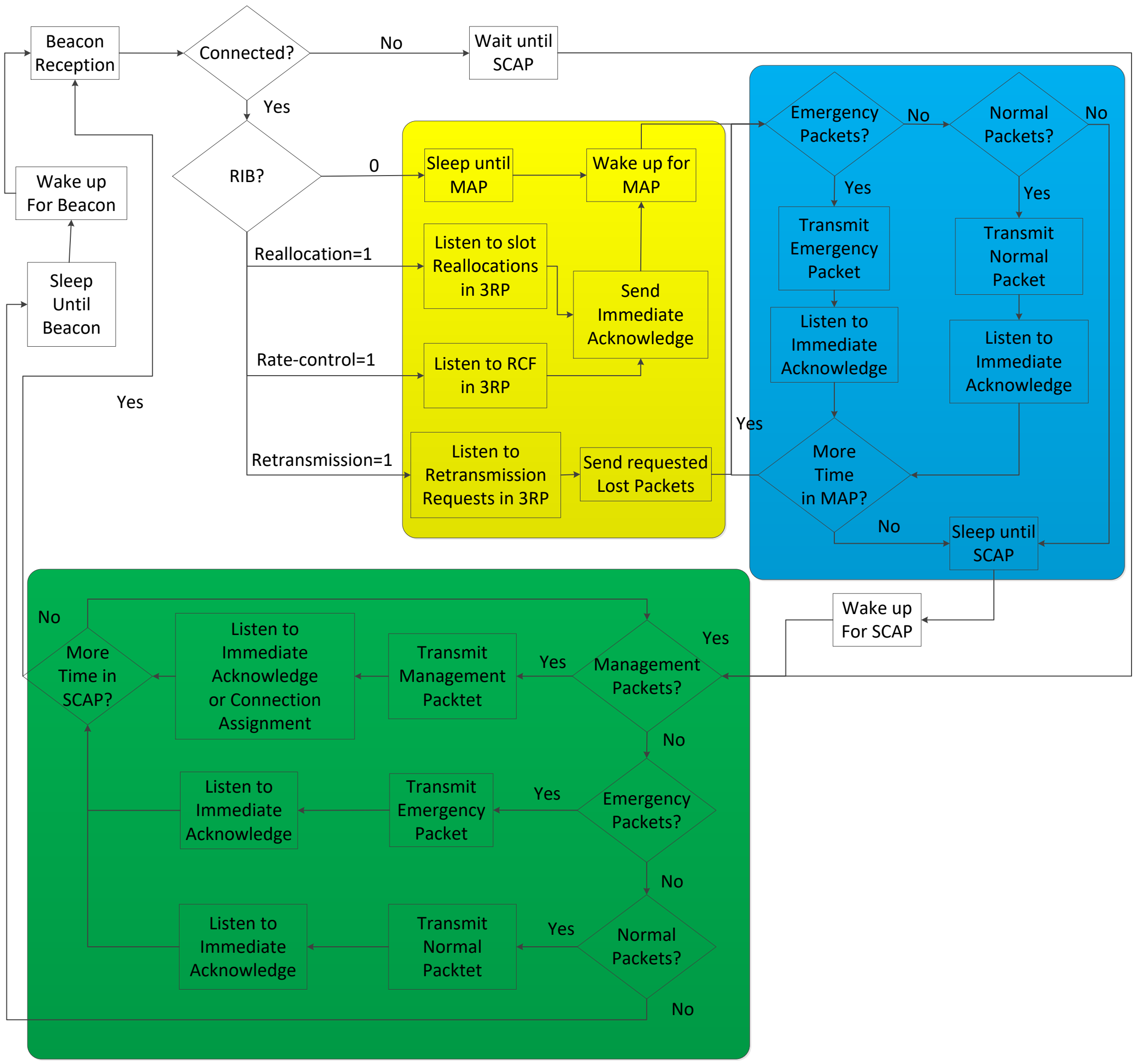


Figure 9. Overall Behavior of the Nodes.

Finally, during SCAP phase (the green color zone), the node sends management packets (e.g., connection requests) and additional emergency and normal traffic. The priority from the highest to the lowest for the traffic during SCAP is: (1) Management packets, (2) Emergency traffic, and (3) Normal traffic.

V. EXPERIMENTAL RESULTS

The MAC protocol, the Transport protocol and the Rate Control schema were compared against the IEEE 802.15.6 Standard, the IEEE 802.15.4 Standard and the T-MAC protocol. Castalia was chosen as the simulator because it offers a very good implementation for the IEEE 802.15.6 MAC protocol [11]. The performance of the architecture was evaluated using three parameters: (i) the packet loss; (ii) the Energy Waste Index (EWI), which was calculated as the ratio between the total percentage of lost packets and the average consumed energy. The lower the EWI, the better the energy effectiveness of the proposed solution; and (iii) the latency of the normal and the emergency traffic.

In some simulations, the number of emergency events was changed incrementally from one to five. Each emergency event had the duration of five seconds. The simulation time was 300s, the packet rate was 20pkt/s, and the number of nodes was 10. The percentage of emergency packet loss when the number of emergency events was changed in the WBAN is depicted in Figure 10. The proposed solution showed the lowest percentage of emergency packet loss (almost 0% no matter the total number of emergency events) because of the Slot Reallocation Algorithm and the Packet Loss Detection Algorithms. The behavior of the IEEE 802.15.4 MAC protocol and T-MAC protocol with the emergency traffic when we increased the number of emergency events in the WBAN, demonstrates why we should not use WSN MAC protocols directly on WBANs. As the emergency events were not generated at the same time, the behavior of all protocols stayed almost the same when we change the number of emergency events in the WBAN.

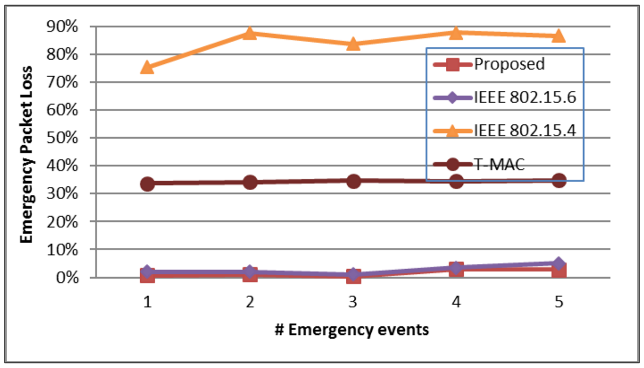


Figure 10. Emergency Packet Loss vs Number of Emergency Events.

For other simulations, the simulation time was changed incrementally from 400s to 2000s. The packet rate was 20pkt/s, and the number of nodes was ten. There were three emergency events with the duration of five seconds each. The Energy Waste Index when the simulation time was changed in the WBAN is depicted in Figure 11. The proposed solution showed the best Energy Waste Index for emergency traffic (almost 0). The IEEE 802.15.4 MAC protocol showed the worst Energy Waste Index because of its poor performance with the emergency packet loss (more than 80%). The Figure 11 shows how the IEEE 802.15.4 and T-MAC protocols improve the Energy Waste Index with the increase of the simulation time, but this is due to the increase of the average energy consumption.

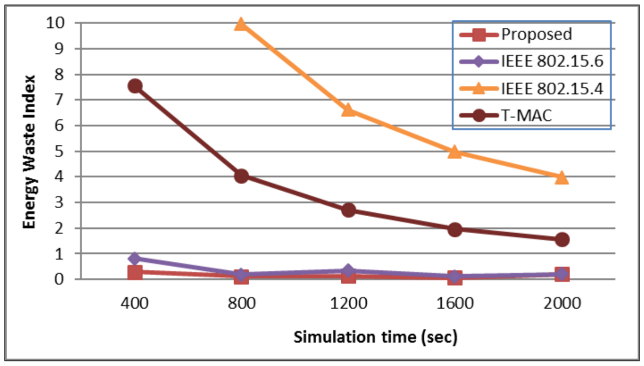


Figure 11. Energy Waste Index vs Simulation Time.

In the final simulations, the simulation time was 300s, the number of nodes was six, the packet rate was 20pkt/s and there was one emergency event at t=150s with the duration of five seconds. The latency distribution for emergency packets and normal packets is depicted in Figure 12 and Figure 13, respectively. The number of emergency packets with the lowest latency (between 0 and 100 milliseconds) in the proposed solution was much higher than the other three MAC protocols. This is due to the Slot Reallocation Algorithm, the lack of contention for emergency traffic during the MAP phase, and besides, the additional contention phase (SCAP) for emergency and normal traffic into each beacon period.

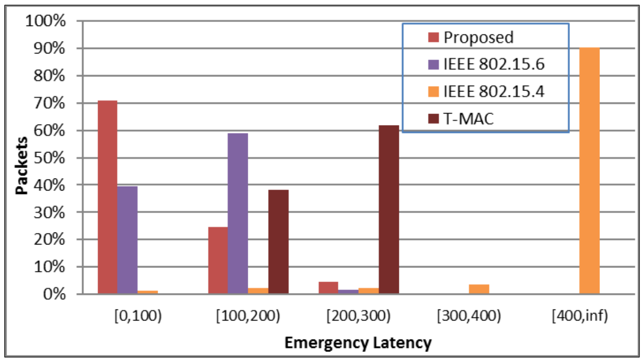


Figure 12. Latency for Emergency Traffic.

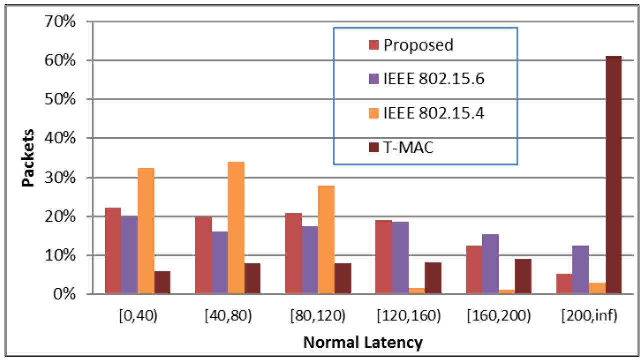


Figure 13. Latency for Normal Traffic.

The number of normal packets with low latency (between 0 and 160 milliseconds) in the proposed MAC protocol was higher than the other MAC protocols, excepting the IEEE 802.15.4 MAC protocol, because of the poor average performance of the latter with emergency traffic. With the IEEE 802.15.4 MAC protocol, almost 90% of the emergency traffic was delivered with the latency of more than 400 milliseconds, while almost 90% of the normal traffic was delivered with the latency of fewer than 120 milliseconds.

VI. ARCHITECTURES COMPARISON

The Table II presents a comparison of the proposed architecture with some architectures published recently and described in Section II. In order to make the comparison with other architectures, the main requirements of all the architectures were used. The first architecture used for the comparison was a Context-Aware Service Architecture for the Integration of WBANs and Social Networks through the IMS presented by Domingo [5]. The second architecture was an Un-Obstructive WBAN for Efficient Movement Monitoring presented by Felisberto et al. [3]. The third architecture was a Cloud-Enabled WBAN for Pervasive Healthcare presented by Wan et al. [6]. The fourth architecture was a QoS-Aware Health Monitoring System using Cloud-Based WBANs presented by Almashaqbeh et al. [4]. The final architecture was a Configurable Energy-Efficient Compressed Sensing Architecture with its application on WBANs presented by Wang et al. [2].

TABLE II. ARCHITECTURES COMPARISON

Requirement	**Proposed Architecture**	**Context-Aware Service Architecture [5]**	**Un-Obstructive Architecture - Movement Monitoring [3]**	**Cloud-Enabled Architecture - Pervasive Healthcare [6]**	**QoS-Aware Health Monitoring System [4]**	**Compressed Sensing Architecture [2]**
Energy Efficiency	Yes	No	Yes	Yes	No	Yes
Reliability	Yes	No	No	Yes	No	No
QoS	Yes	No	No	Yes	Yes	No
Congestion Control	Yes	No	No	No	Yes	No
Rate Control	Yes	No	No	No	No	Yes
Loss Detection	Yes	No	No	No	Yes	No
Loss Recovery	Yes	No	No	No	No	No
Fairness	Yes	No	No	No	No	No
Emergency Awareness	Yes	Yes	Yes	Yes	No	No
Context Awareness	Yes	Yes	No	Yes	No	No
Security	No	No	No	Yes	No	No
Coexistence	No	No	No	No	Yes	No
Interference	No	No	No	No	Yes	No
Topology changes	No	No	Yes	No	No	No
Node Placement Optimization	No	No	Yes	No	No	No
Nodes Wearability	No	No	Yes	No	No	No
Energy Harvesting	No	No	Yes	No	No	No

VII. CONCLUSION AND FUTURE WORK

The main objective was to design a reliable, context-aware and energy-efficient architecture for WBANs, ensuring QoS and fairness in sports applications. This objective was achieved through the joint of some protocols, algorithms, schemes, and the proposition of a new hub and nodes architecture. The architecture is composed of: (i) an energy-efficient, context-aware and reliable MAC protocol; (ii) a reliable transport protocol based on loss-recovery and fairness; and (iii) a context-aware rate control scheme for congestion control in WBANs.

The architecture was compared with other architectures using the main requirements of all the proposed architectures. While some architectures focused on challenges like coexistence, interference, topology changes, node placement optimization, nodes wearability, and energy harvesting, the proposed architecture is the only one focused on energy efficiency, reliability, QoS, congestion control, rate control, loss detection, loss recovery, fairness, emergency awareness, and context awareness, all at the same time.

Future work includes working in additional challenges to design WBANs like high-security mechanisms and topology changes support. Besides, the development of new energy-efficient routing protocols taking advantage of the coexistence of other WBANs in the vicinity. This work could be extended to enhance the quality of life of children, ill and elderly people.

REFERENCES

[1] IEEE Standard for Local and metropolitan area networks - Part 15.6: Wireless Body Area Networks," in IEEE Std 802.15.6-2012 , vol., no., pp.1-271, Feb. 29 2012

[2] A. Wang, F. Lin, Z. Jin, and W. Xu, "A Configurable Energy-Efficient Compressed Sensing Architecture With Its Application on Body Sensor Networks," in IEEE Transactions on Industrial Informatics, vol. 12, no. 1, pp. 15-27, Feb. 2016.

[3] F. Felisberto, N. Costa, F. Fdez-Riverola, and A. Pereira, “Unobstructive Body Area Networks (BAN) for efficient movement monitoring”. Sensors. 2012 Sep 13;12(9):12473-12488.

[4] G. Almashaqbeh, T. Hayajneh, A. V. Vasilakos, and B. J. Mohd, “QoS-aware health monitoring system using cloud-based WBANs”. Journal of medical systems. 2014 Oct 1;38(10):121.

[5] M. C. Domingo, "A context-aware service architecture for the integration of body sensor networks and social networks through the IP multimedia subsystem," in IEEE Communications Magazine, vol. 49, no. 1, pp. 102-108, January 2011.

[6] J. Wan, C. Zou, S. Ullah, C. F. Lai, M. Zhou, and X. Wang, "Cloud-enabled wireless body area networks for pervasive healthcare," in IEEE Network, vol. 27, no. 5, pp. 56-61, September-October 2013.

[7] E. Kartsakli, A. S. Lalos, A. Antonopoulos, S. Tennina, M. D. Renzo, L. Alonso, and C. Verikoukis, “A survey on M2M systems for mHealth: a wireless communications perspective”. Sensors 14, no. 10 (2014): 18009-18052.

[8] R. Jaramillo, A. Quintero, and S. Chamberland, "Energy-efficient MAC protocol for Wireless Body Area Networks," 2015 International Conference and Workshop on Computing and Communication (IEMCON), Vancouver, BC, 2015, pp. 1-5.

[9] R. Jaramillo, A. Quintero, and S. Chamberland, "Reliable Transport Protocol Based on Loss-Recovery and Fairness for Wireless Body Area Networks," 2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), Washington, DC, 2016, pp. 18-23.

[10] R. Jaramillo, A. Quintero, and S. Chamberland, "Rate control scheme for congestion control in wireless body area networks," 2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), New York, NY, USA, 2016, pp. 1-6.

[11] NICTA. (2013). Castalia Wireless Sensor Network Simulator. Retrieved from http://castalia.npc.nicta.com.au/

Wireless Communications in Railway Systems

Anna Lina Ruscelli*, Gabriele Cecchetti*, Andrea Sgambelluri*, Filippo Cugini†,
Alessio Giorgetti*, Francesco Paolucci*, Silvia Fichera*, Piero Castoldi*,
*Scuola Superiore S. Anna, Pisa, Italy
Email: {a.ruscelli | g.cecchetti | a.sgambelluri | a.giorgetti | fr.paolucci | s.fichera | p.castoldi}@santannapisa.it
†CNIT, Pisa, Italy
Email: f.cugini@cnit.it

***Abstract*—Railway management systems are based on a centralized structure where the Central Post centrally manages all the components, trains and railways, with the aim to guarantee the safety of the service and the efficiency of the network capacity. This architecture requires an extended exchange of information between management units, monitoring systems and actuators. In general, the communication is based on wired links that ensure required performance, but present also some drawbacks. For instance, copper cable-based links are affected by cables thefts or can limit the type and the amount of information that can be sent due to the capacity of involved technologies. In all these scenarios, the introduction of a wireless link can improve the safety, the performance and the flexibility of the communications. In this paper, the use of wireless communications as backup or extension of the pre-existent wired links is deepened. Trackside and on-board communications, as well as European Rail Traffic Management System and EURORADIO protocol are studied analyzing the issues related to wired links and illustrating how the use of wireless communications can face off their drawbacks.**

Keywords*—Railway systems; wireless communications; trackside systems; on-board systems; signalling; ERTMS; EURORADIO; RadioInfill; MRP.

I. Introduction

Railways are complex systems composed by infrastructure, vehicles, and all the elements required to make these components work together efficiently and safely. Vehicles for freight and passengers transit on the railways that are a complicated network of connections where they have to be synchronized in order to avoid accidents. Furthermore, the transport system has to be managed efficiently in order to increase the system capacity in terms of number of convoys travelling on the network. Both these requirements, safety and efficiency, require and required along the time to monitor and manage the trains transit and the railways. This is the motivation of the complex and highly populated system composed by trackside and on-board equipments used by the signalling system to monitor the state of infrastructures and convoys, to dispatch and actuate command. In particular, a huge variety of devices performing important functions are distributed along a railway. Examples are railway switches boxes that command switches, equipment that monitor the trains status (bush temperature detector) and the trains transit (axle counter, track circuit), light signals used to communicate with the train drivers, components that allow the communication between the ground management system (Computer-Based Interlocking) to the on-board management system (Lineside Electronic Equipment and Encoder), etc. Furthermore, both on-board and trackside devices communicate with the centralized management units through some intermediate management and information points. Indeed, the trains management has a centralized structure where a central unit, the Central Post, has the task of manage the network infrastructure and the trains. This unit distributes its commands by means of intermediate points, the Peripheral Posts, in general corresponding to the trains stations. At their turn the Peripheral Posts dispatch and elaborate the command received by the Central Posts to the trackside and on-board equipments by means of the *Computer-Based Interlocking* (CBI) system. Moreover Peripheral Posts and CBI receive and elaborate the information collected by trackside and on-board equipments and send the derived information to the Central Posts in order to update the management system. This centralized architecture allows to organically and safely manage the complex railways system reaching all the devices distributed along the railways.

Reflecting and following the historical evolution of the railway systems dragged by the modifications of the involved technologies, most of these equipments are evolved starting from simple mechanical devices, to electromechanical, to electric-digital components, see for instance the first CBI, where the logic where implemented by a mechanical leverage, or old railway switches that were manually operated. Furthermore, these components are not isolated entities but they communicate together in order to exploit their functions. Until now, most of the communications are based on wired links, where the information derived by the monitoring systems are dispatched to the central management unit and the commands set by the last one are sent to the actuators spread on-board of trains or trackside. In dependency of the type of message and of the involved devices these communications are based on different protocols. However, as previously said, wired links based, for instance, on copper cables, or optical fibers, are used. Through the right choice of transmission technology and protocols and thanks to the appropriate setting of transmission parameters, wired links ensure required performance but present also some drawbacks. For instance, copper cable-based links are affected by cables thefts due to the high monetary value of copper [1]. This is a severe damage, especially in the case of links used for critical communications since their interruption can seriously jeopardize safety trains transit. Furthermore, in some cases, such as on-board communications, wired links limit the type of information that can be sent due to the capacity of involved technologies. In all these scenarios the introduction of a wireless link can improve the safety, the performance and the flexibility of the communications. In this paper, the use of wireless communications as backup or extension of the pre-existent wired links, in dependency of the application, is deepened. Both trackside and on-board communications are studied analyzing the issues related to wired links and illustrating how the use of wireless communications can face off their drawbacks. Some meaningful case studies will be provided in order to corroborate the proposal along with the highlight of some open issues related to the introduction of wireless links.

The rest of the paper is composed by the mentioned study that is exposed in Section II, whereas in Section III some conclusions are drawn.

II. Introducing wireless communications in railway systems

The railway management deals with a capillary structure where railway lines are populated by a huge amount of devices suitable to monitor the transit and the status of the train and by actuators used to manage railway lines. They communicate with the Central Post, responsible to centrally manage the whole infrastructure, through the Peripheral Posts. Furthermore, on board of convoys monitoring and actuators collect the information about the state of the train and execute the received commands. In general, the communication is based on wired links that, in dependency of the applications, can result not sufficient to support new functionalities, for instance video surveillance or on-board entertainment, or are subject to critical damages or copper cable thefts that seriously jeopardize the critical management applications. Thus the evolution of railway systems has to consider new solutions suitable to face off these challenges. A possible solution is the adoption of wireless communications both as a backup of the wired one, for instance to ensure the service until the system is recovered, or as integration to the wired one in order to allow new services or to improve the pre-existent ones.

In the following, some use cases about the introduction of wireless communications in the railway domain are illustrated in order to highlight the potentialities of this approach.

A. Communications between wayside equipment and Peripheral Post

Copper or fiber optic cables typically support communications between trackside devices and accidental or intentional damage of copper cables can cause unavailability of the corresponding link. Wireless communication can be introduced as a backup of the wired link. This require to analyze the communication requirements in terms of bandwidth, delay, packets loss strictly related to the particular application. To overcome the problem of possible interference, a Spread Spectrum technology can be Zang2005Zang2005chosen as possible alternative to narrowband technologies [2]. Indeed, thanks to its robustness against noise and intrusion, this technology is suitable to provide the required reliable communication level. Beyond the precise design and setting of the wireless link that has to be apt to provide the required Quality of Service, further investigations require analyzing the compatibility of legacy communication interfaces of the involved railway devices, often based on proprietary connectors, with the proposed wireless devices that in general have standardized interfaces. In this case, either a re-engineering of the devices is required, either an interface adapter has to be designed for this connection.

A considered use case is the link between Eurobalise and Lineside Electronic Unit (LEU). Switched Eurobalise communicates with the corresponding LEU through the C interface, which is continuously powered by the LEU and it is connected to the physical wired link for the exchange of telegrams, powering and other information about the train transit. In order to use wireless links for their communications, an interface adapter is required to be connected to the C interface on one side and to the Ethernet interface of the wireless device on the other side, as illustrated in Figure 1. The same apply on the corresponding interface of the LEU.

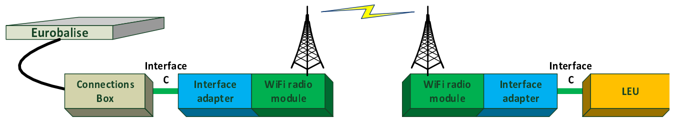


Figure 1. Eurobalise - LEU case of study.

The adapter mechanically interconnects the proprietary input-output connector of the C interface to the standard interface of the Wi-Fi module. It adapts electrical features of the cable signals to the corresponding ones of the serial or Ethernet standard and preserves timing requirements. Creating wireless link requires a careful design in order to connect in a point-to-point topology the two Wi-Fi devices for the wireless transmission between Eurobalise and LEU that implies to consider different aspects. Since Eurobalise and LEU are placed in fixed locations, the position of the directional antenna of each Wi-Fi device has to be chosen carefully. Each antenna has to be placed far not more than few meters from the corresponding Wi-Fi equipment to minimize signal losses through the cable and should be placed in Line of Sight of trackside poles where the antennas are mounted, at a height compatible with other railways elements eventually present. In case of obstacles and, in general, in presence of Non Line of Sight, it is necessary to insert one or more bridge-repeaters that turn around the obstacle. Obviously, the radio link has to meet the requirements about bandwidth, frame loss and delay, preserving the connection between Eurobalise and LEU. In particular, the link budget, i.e., the algebraic sum of all gains and losses of each component of the radio system has to be taken into account since it allows to decide if it is necessary to act on transmission power or antenna gains to obtain the desired performance. Experimental results shown that the wireless link can meet the mentioned requirements providing a backup connection suitable in case of hard damage of the wired one due to accidental or intentional

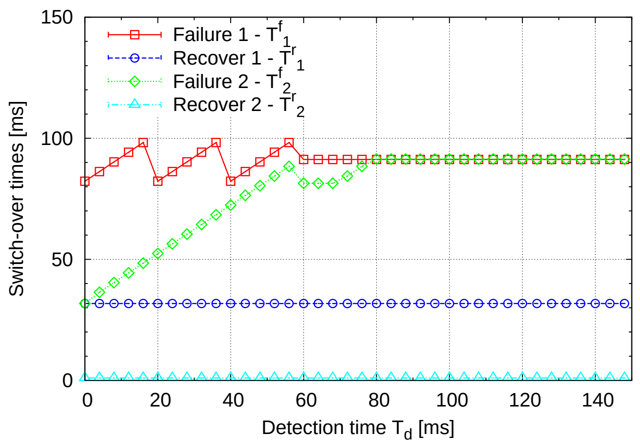


Figure 2. Switch-over time of MRP. vs detection time

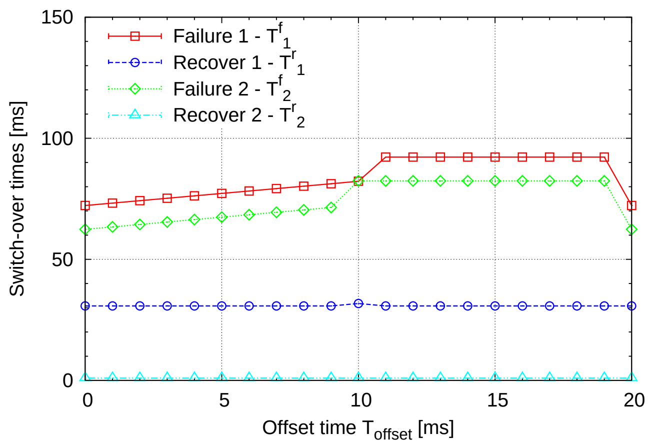


Figure 3. Switch-over time of MRP. vs offset time

damage (theft or physical cut), improving the system fault tolerance.

The completely reduction/elimination of copper cables implies further open issues, related to the presence of power cables. In this case the approach is completely different with respect to communication cables, since the requirements involve the continuous power provisioning. For instance, solution based on energy harvesting could overcome this challenge [3].

B. Railway control networks and communications between Peripheral Post and Central Post

Ethernet is penetrating railway communication networks mainly for its simplicity and cost effectiveness but also thanks to several emerging Industrial Ethernet solutions that improve the Ethernet standard. Specifically, Industrial Ethernet solutions often include proprietary redundancy management protocols for the automatic handling of failures on ring topologies. The *International Electrotechnical Commission* (IEC) has published IEC 62439 in 2010 [4] including the specification of a standard redundancy management protocol, i.e., the *Media Redundancy Protocol* (MRP), that attracted the attention of most network equipment vendors. With the settings specified in IEC 62439, MRP guarantees a worst case recovery time of 30 ms in rings composed of up to 50 switches, and can support multi-ring topologies guaranteeing similar performance. MRP is currently used in various network segments of RFI railway communication serving the control of high-speed high-capacity train. In [5], two factors are identified (i.e., offset time and the physical detection time) that jointly affect the MRP performance. Their impact on the recovery time has been consistently evaluated with an analytical approach, with simulations (see Figure 2 and Figure 3), and by means of experimental measurements. Obtained results confirmed that, in all the considered scenarios, the switch-over is performed within the target time declared in IEC 62439.

In this scope, wireless communications can be used as a backup of PVS wired communications between Peripheral Post and Central Post, in most of the system performed by EURORADIO standard protocol or by proprietary national protocols, such as Italian *Protocollo Vitale Standard* (PVS). This intervention could have a huge impact involving important communications, whereas the adoption of wireless communications, for instance based on IEEE 802.11ac protocol, can easily provide the required levels of Quality of Service in terms of bit data rate and security.

C. On-board LAN

Another important challenge is the introduction of wireless communications on the Local Area Network on the train (i.e., on-board LAN). Typically on-board LANs on the trains are implemented using commercial layer 2 switches interconnected by means of copper cables traversing the several carriages. This solution is able to provide adequate bandwidth (e.g., 100 Mbps or 1 Gbps) on the LAN, but introduces some rigidity in the dynamic re-combination of carriages. The utilization of a wireless bridge among adjacent carriages could facilitate dynamic re-arrangement of trains carriages while guaranteeing adequate network performance. The main problems to be addressed are the maximum supported bitrate, and the required integration of the wireless devices in the failure recovery mechanisms typically supported by the on-board LAN. Regarding the supported bit rate traffic bandwidth of up to 200 Mbps can be supported with very cheap hardware (e.g., about hundred dollars each wireless bridge). Regarding the integration of the recovery techniques, properly designed scripts should be implemented and deployed on the wireless devices so that failures of the local interfaces can be announced to the rest of the on-board LAN to properly recover the affected traffic flows.

D. ERTMS

European Rail Traffic Management System (ERTMS) [6], [7] is the European reference management system suitable to homogeneously manage different national trains when they cross national boundaries. It aims to overcome the limits of the diverse national management systems, each one based on different communication protocols, system architectures, trackside and on-board components. This heterogeneity implies that when a train travels across different countries the on-board equipment has to be able to interface to different signaling systems. This problem is generally overcame by substituting the locomotive, where the on board system is placed, or equipping the locomotive with all the different equipment corresponding to the different crossed nations. Obviously, this approach is not flexible and efficient, impacting on travel time and railway capacity. In this context ERTMS aims to homogenize the signaling systems by the introduction of a unique management system. Furthermore, its goal is to improve the railway efficiency by a progressive substitution of wired communications with wireless ones. This will allow a reduction of trackside devices and the introduction of further and improved functionalities, as will be described in the following. The introduction of ERTMS theoretically follows three different steps, starting from Level 1, compatible with pre-existent systems based on exclusively wired communications, to Level 2, see Figure 4, where wireless communications are side by side to the wired ones, to the Level 3, where only wireless links are used.

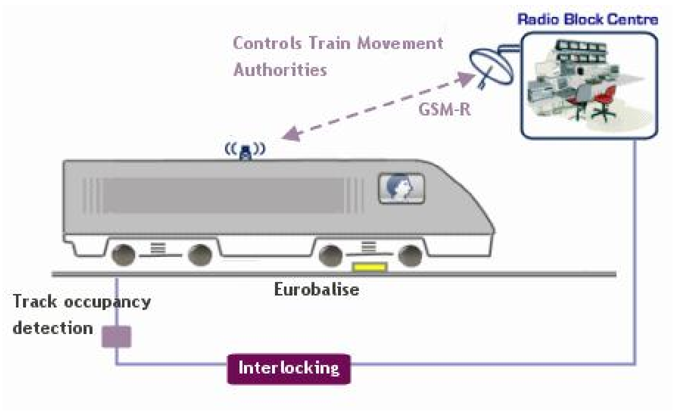

Radio Block Centre
Controls Train Movement Authorities
GSM-R
Track occupancy detection
Eurobalise
Interlocking

Figure 4. ERTMS Level 2

1) EURORADIO: ERTMS defines as secure communication protocol EURORADIO [8] based on an open communication network such as *Global System for Mobile Communications âĂŞ Railway* (GSM-R). It is based on a layered architecture and all the layers are executed onboard of the train to enable the communication. In particular, a *Safety Functional Module* (SFM) and a *Communication Functional Module* (CFM) respectively deal with safety transmission functionalities and communication system functionalities.

In [2], an implementation of EURORADIO is presented, developed with open-source tools for better portability. It is based on software stack of different layers that form a hierarchy of functionalities starting from the physical hardware components (Modem GSM-R) to the user interfaces at the software application level (Radio-Infill Application), see Figure 5.

EURORADIO layers communicate together by means of API. Each layer receives information from the layer above,

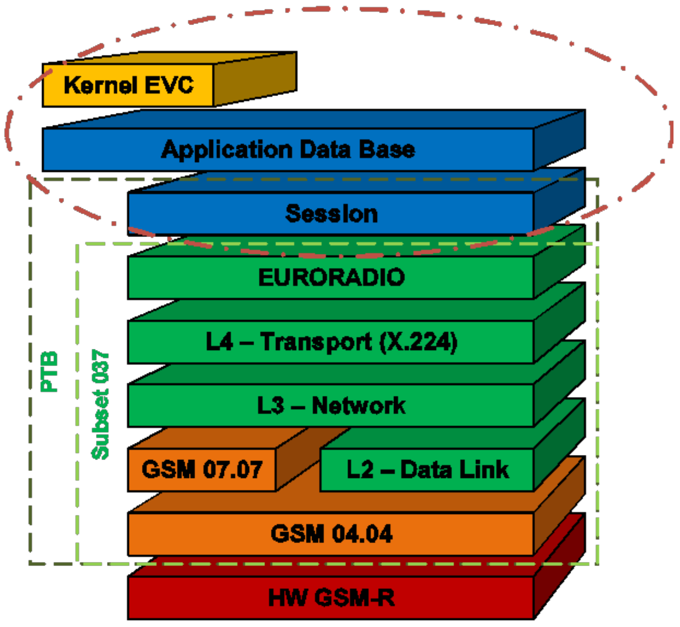


Figure 5. EURORADIO protocol.

processes and transfers that to the layer below, adding its own encapsulation information (header).

EURORADIO is the basis for ERTMS Level 2 and Level 3 but it allows to improve the behavior also of ERTMS Level 1, where the trackside-traiborne communication (communication between the ground subsystem SST and the onboard subsystem SSB) is through Eurobalises based on a duplication of information. This communication is discontinuous being Eurobalises placed in fixed and meaningful positions along the railways lines. Furthermore, the train driver can modify the train speed only after these information points. This makes the speed curve not optimal, due to discontinuous accelerations and decelerations, and the trains circulation not efficient. However, the introduction of the RadioInfill function is suitable to provide a compromise between the use of ERTMS Level 1 and the continuous communications.

2) RadioInfill: RadioInfill function supported by EURORADIO in ERTMS Level 1 at the application layer faces off the lack of responsiveness typical of discontinuous ATP systems. In ERTMS Level 1 the signaling information delivered to the train driver is based on information points, for instance light signals used to deliver stop and go information to the train driver. A scheme based on duplicated information is used for safety reasons. Double information points convey a single type information: one notice point as an advice, and one protection point as a confirmation. RadioInfill allows an early release of the running restriction before the next information point, as shown in Figure 6. The early release function is used when routes conditions are further exchanged and a train deceleration is no more necessary and to manage deceleration in proximity of stations. Some implementation of RadioInfill are based on pre-existent signaling system sending coded electric signals through the track circuit or on dedicated

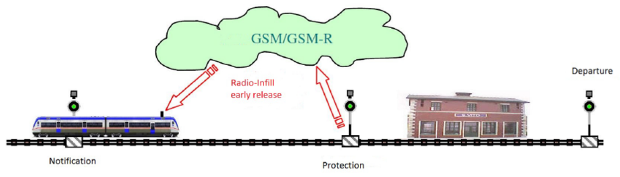


Figure 6. The early release of RadioInfill function.

trackside components (Euroloop). In [2], an implementation of EURORADIO protocol and of RadioInfill is described. The experimentation shown as this function can reduce the travel time and improve the power saving reducing the number of braking.

3) ERTMS Level 3: As mentioned, the distinctive element of ERTMS Level 3 is the exclusive use of wireless communications for the exchange of information between train and ground system, see Figure 7.

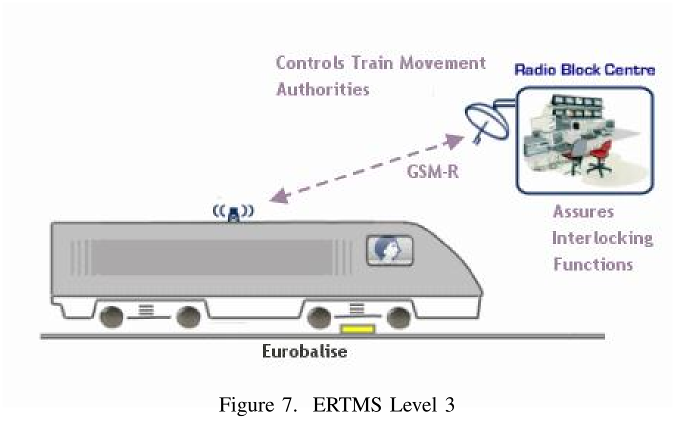

Controls Train Movement Authorities
Radio Block Centre
GSM-R
Assures Interlocking Functions
Eurobalise

Figure 7. ERTMS Level 3

This allows the reduction of trackside signaling devices with a consequent reduction in costs and maintenance load. Furthermore, the speed curve is continuous improving energy consumption efficiency and passenger travel comfort. At the present time there are no implementations of ERTMS Level 3 but railway operators and European national authorities are actively considering the evolution of their systems in order to implement its functionalities, see Norway or Finland, and some nations of the East of Europe are evaluating to update their national management systems directly to ERTMS Level 3, pushing forward the evolution and avoiding the huge implementation of the consecutive different levels, starting from Level 1.

4) Moving Blocks: The goal is to improve the efficiency of rail traffic management and the quality of service offered by increasing the capacity of the line. For this purpose, ERTMS Level 3 is based on the use of the Moving Block concept to manage trains on the same line [9]. *Moving Block Signaling* (MBS) is an intelligent control system where safety zones around the train are defined that can not be crossed by adjacent trains. Specifically, MBS exceeds the Fixed Block limits where the line is divided into fixed length blocks determined based on the braking capacity of the train in worst case conditions, taking into account the speed allowed in the line, and delimited by signals. According to this system, a train can access a block only if its next one is free, so the distance between two trains on the same line is more than one block [10] [11]. This results in an accumulation of braking times and excessive spacing between the trains, affecting the density of trains on the line. To increase the capacity of the line, i.e., the number of trains on it, MBS introduces "mobile" blocks that are no longer delimited by long distance signals and whose length is not fixed but determined by the safety distance needed to completely stop the train.

According to this method, the moving block is determined by the position of the train and the safety distance from it, and no other signaling equipment is needed being managed by the ERTMS control system. Particularly in Moving Block, the train is modeled with a safe-envelope consisting of the sum of its length, a rear safety margin that takes into account the distance of rollback, uncertainty in determining the position of the train and spacing with the next train, and a frontal security margin which, in turn, takes into account the uncertainty in determining the train head and the distance traveled during the maximum permissible time interval in which ground-to-earth communication can be interrupted [12]. Thus, the minimum distance of a train from the preceding one, the *Limit of Movement Authority* (LMA), is dependent on the position of the train ahead and its specific braking and speed properties. This interdiction space moves along the line to proceed to the next train. This reduces the length of the moving blocks and, consequently, the distance between the trains, allowing to increase the capacity of the line.

III. Conclusion and future work

In this paper, some use cases about the introduction of wireless communications as backup or integration to the wired ones in the railway domain are illustrated. Trackside and on-board communications, as well as ERTMS systems and EURORADIO protocol are studied analyzing the issues related to wired links and illustrating how the use of wireless communications can face off their drawbacks.

Despite the use of wireless communications in most cases is a challenge especially considering the strict safety and service requirements but, from the other hand it opens new field of application and feeds the evolution of the railway systems.

Future works will be focused on the management of the train integrity. Until now, this function is guaranteed by the train inauguration process and by the monitoring of the train and of its queue in particular by means of the *Train Communication Network* (TCN). Furthermore, it can be integrated by the use of wireless communications, for instance, monitoring the round trip time between the head and the queue of the train.

Further application of wireless communication that deserves to be deepened is the "cloudification" of railway management and services which can be based on wireless links as backup connections.

References

[1] H. of Commons Transport Committee, “Cable theft on the railway - Fourteenth Report of Session 2010-2012,” House of Commons, Tech. Rep., 2012.

[2] G. Cecchetti, A. L. Ruscelli, A. Sgambelluri, F. Cugini, and P. Castoldi, “Wireless Spread Spectrum for trackside railways communication systems,” in *11th World Congress on Railway Research WCRR*, 2016. [Online]. Available: https://www.sparkrail.org/Lists/Records/DispForm.aspx?ID=23559

[3] A. L. Ruscelli, G. Cecchetti, and P. Castoldi, “Energy Harvesting for Trackside Railways Communications,” in *11th World Congress on Railway Research WCRR*, 2016. [Online]. Available: https://www.sparkrail.org/Lists/Records/DispForm.aspx?ID=23560

[4] “Iec 62439,” https://webstore.iec.ch/publication/24447, (Retrieved: May, 2017).

[5] A. Giorgetti, F. Cugini, F. Paolucci, L. Valcarenghi, and A. e. a. Pistone.

[6] “Directive DC 2001/16/ec,” http://eur-lex.europa.eu/smartapi/cgi/sga_doc?smartapi!celexplus!prod!DocNumber&lg=en&type_doc=Directive&an_doc=2001&nu_doc=16, (Retrieved: May, 2017).

[7] “Directive DC 96/48/ec,” http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31996L0048:it:HTML, (Retrieved: May, 2017).

[8] S.-. ERTMS/ETCS, “Subset-037 ERTMS/ETCS - class 1 EURORADIO functional interface specifications,” www.era.europa.eu, UNISIG, jul 2005.

[9] H. Takeuchi, C. J. Goodman, and S. Sone, “Moving block signalling dynamics: performance measures and re-starting queued electric trains,” vol. 150, no. 4, July 2003, pp. 483–492.

[10] Z. Li-yan, L. Ping, J. Li-min, and Y. Feng-yan, “Study on the simulation for train operation adjustment under moving block,” in *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, Sept 2005, pp. 351–356.

[11] N. A. Zafar, “Formal model for moving block railway interlocking system based on un-directed topology,” in *2006 International Conference on Emerging Technologies*, Nov 2006, pp. 217–223.

[12] L. Xu, X. Zhao, Y. Tao, Q. Zhang, and X. Liu, “Optimization of train headway in moving block based on a particle swarm optimization algorithm,” in *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, Dec 2014, pp. 931–935.

Assessment of Data Storage Strategies Using the Mobile Cross-Platform Tool Cordova

Gilles Callebaut, Lieven De Strycker

KU Leuven
DraMCo research group
Department of Electrical Engineering
Technology Campus Ghent, Gebroeders De Smetstraat 1
Email: firstname.lastname@kuleuven.be

Michiel Willocx, Vincent Naessens, Jan Vossaert

KU Leuven
MSEC, imec-DistriNet
Technology Campus Ghent, Gebroeders De Smetstraat 1
Email: firstname.lastname@cs.kuleuven.be

Abstract—**The mobile world is fragmented by a variety of mobile platforms, e.g., Android, iOS and Windows Phone. While native applications can fully exploit the features of a particular mobile platform, limited or no code can be shared between the different implementations. Cross-platform tools (CPTs) allow developers to target multiple platforms using a single codebase. These tools provide general interfaces on top of the native APIs. Apart from the performance impact, this additional layer may also result in the suboptimal use of native APIs. This paper analyses the impact of this abstraction layer using a data storage case study. Both the performance overhead and API coverage is discussed. Based on the analysis, an extension to the cross-platform storage API is proposed and implemented.**

Keywords–Cross-platform tools; data storage; performance analysis; API coverage; Apache Cordova/Phonegap.

I. Introduction

An increasing number of service providers are making their services available via the smartphone. Mobile applications are used to attract new users and support existing users more efficiently. Service providers want to reach as many users as possible with their mobile services. However, making services available on all mobile platforms is very costly due to the fragmentation of the mobile market. Developing native applications for each platform drastically increases the development costs. While native applications can fully exploit the features of a particular mobile platform, limited or no code can be shared between the different implementations. Each platform requires dedicated tools and different programming languages (e.g., Objective-C, C# and Java). Also, maintenance (e.g., updates or bug fixes) can be very costly. Hence, application developers are confronted with huge challenges. A promising alternative are mobile cross-platform tools (CPTs). A significant part of the code base is shared between the implementations for the different platforms. Further, many cross-platform tools such as Cordova use client-side Web programming languages to implement the application logic, supporting programmers with a Web background.

Although several cross-platform tools became more mature during the last few years, some scepticism towards CPTs remains. For many developers, the limited access to native device features (i.e. sensors and other platform APIs) remains an obstacle. In many cases, the developer is forced to use a limited set of the native APIs, or to use a work-around –which often involves native code– to achieve the desired functionality. This paper specifically tackles the use case of data storage APIs in Cordova. Cordova is one of the most used CPTs [22, 23]. It is a Web-to-native wrapper, allowing the developer to bundle Web apps into standalone applications.

Contribution. The contribution of this paper is threefold. First, four types of data storage strategies are distinguished in the setting of mobile applications. The support for each strategy using both native and Cordova development is analysed and compared. Second, based on this analysis a new Cordova plugin that extends the Cordova Storage API coverage is designed and developed. Finally, the security and performance of the different native and Cordova storage mechanisms is evaluated for both the Android and iOS platform.

The remainder of this paper is structured as follows. Section II points to related work. Section III discusses the inner workings of Cordova applications, followed by an overview of data storage strategies and their API coverage in Cordova and native applications. The design and implementation of NativeStorage, a new Cordova storage plugin, is presented in Section IV. Section V presents a security and performance evaluation of the Cordova and native storage mechanisms. The final section presents the conclusions and points to future work.

II. Related Work

Many studies compare CPTs based on a quantitative assessment. For instance, Rösler et al. [26] and Dalmasso et al. [18] evaluate the behavioral performance of cross-platform applications using parameters such as start-up time and memory consumption. Willocx et al. [30] extend this research and include more CPTs and criteria (e.g., CPU usage and battery usage) in the comparison. Further, Ciman and Gaggi [17] focus specifically on the energy consumption related to accessing sensors in cross-platform mobile applications. These studies are conducted using an implementation of the same application in a set of cross-platform tools and with the native development tools. This methodology provides useful insights in the overall performance overhead of using CPTs. Other research focuses on evaluating the performance of specific functional components. For instance, Zhuang et al. [31] evaluate the performance of the Cordova SQlite plugin for data storage. The work presented in this paper generalizes this work by providing an overview and performance analysis of the different data storage mechanisms available in Cordova, and comparing the performance with native components.

Several other studies focus on the evaluation of cross-platform tools based on qualitative criteria. For instance, Heitkötter et al. [19] use criteria such as development environment, maintainability, speed/cost of development and user-perceived application performance. The user-perceived performance is analyzed further in [20], based on user ratings and comments on cross-platform apps in the Google Play Store. The API coverage (e.g., geolocation and storage) of cross-platform tools is discussed in [24]. It is complementary with the work presented in this paper, which specifically focuses on the API coverage, performance and security related to data storage.

III. Data Storage In Cordova

A. Cordova Framework

A typical Cordova application consists of three important components: the application source, the WebView and plugins, as depicted in Figure 1.

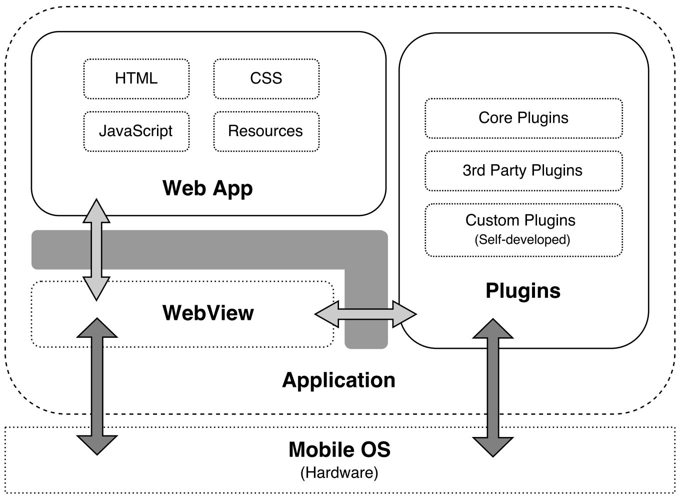


Figure 1. Structure of a Cordova application. Light grey arrows represent JavaScript calls, darker grey arrows represent native calls. The Cordova framework is illustrated by the grey area.

Cordova applications are, similar to Web apps, developed in client-side Web languages (i.e. HTML, CSS and JavaScript). Typically, developers use JavaScript frameworks such as Ionic and Sencha, which facilitate the development of mobile-friendly UIs.

The application code is loaded in a chromeless WebView. By default, Cordova applications use the WebView bundled with the operating system. An alternative is to include the Crosswalk WebView [13]. The Crosswalk WebView provides uniform behaviour and interfaces between different (versions of) operating systems.

Cordova developers have two options for accessing device resources: the HTML5 APIs provided by the WebView and plugins. Despite the continuously growing HTML5 functionality [11] and the introduction of Progressive Web Apps [8], the JavaScript APIs provided by the WebView are not –yet– sufficient for the majority of applications. They do not provide full access to the diverse resources of the mobile device, such as sensors (e.g., accelerometer, gyroscope) and functionality provided by other applications installed on the device (e.g., contacts, maps, Facebook login). Plugins allow JavaScript code to access native APIs by using a JavaScript bridge between the Web code and the underlying operating system. Plugins consist of both JavaScript code and native code (i.e. Java for Android, Objective-C and recently Swift for iOS). The JavaScript code provides the interface to the developer. The native source code implements the functionality of the plugin and is compiled when building the application. The Cordova framework provides the JavaScript bridge that enables communication between JavaScript and native components. For each platform, Cordova supports several bridging mechansims. At runtime, Cordova selects a bridging mechanism. When an error occurs, it switches to another mechanism. Independent of the selected bridging mechanism, the data requires several conversion steps before and after crossing the bridge. Commonly used functionality such as GPS are provided by Cordova as *core plugins*. Additional functionality is provided by over 1000 third-party plugins, which are freely available in the Cordova plugin store [12].

B. Storage API Coverage

This work focuses on data storage mechanisms in Cordova applications. Four types of data storage strategies are distinguished: files, databases, persistent variables and sensitive data. Databases are used to store multiple objects of the same structure. Besides data storage, databases also provide methods to conveniently search and manipulate records. File storage can be used to store a diverse set of information such as audio, video and binary data. Persistent variables are stored as key-value pairs. It is often used to store settings and preferences. Sensitive data (e.g., passwords, keys, certificates) are typically handled separately from other types of data. Mobile operating systems provide dedicated mechanisms that increase the security of sensitive data storage.

The remainder of this section discusses the storage APIs available in Cordova and native Android/iOS. A summary of the results is shown in Table I.

TABLE I. Storage API Coverage

	Cordova	Android	iOS
Databases	WebSQL IndexedDB SQLite Plugin	SQLite	SQLite
Files	Cordova File Plugin	java.io	NSData
Persistent Variables	LocalStorage	Shared Prefs	NSUserDefaults Property Lists
Sensitive Data	SecureStorage Plugin	KeyStore KeyChain	Keychain

1) Databases: Android and iOS provide a native interface for the **SQLite** library. Cordova supports several mechanisms to access database functionality from the application. First, the developer can use the database interface provided by the WebView. Both the native and CrossWalk WebViews provide two types of database APIs: **WebSQL** and **IndexedDB**.

Although WebSQL is still commonly used, it is officially deprecated and thus no longer actively supported [10]. Second, developers can access the native database APIs via the **SQLite Plugin** [6].

2) Files: In Android, the file storage API is provided by the **java.io** package, in iOS this is included in **NSData**. Cordova provides a core plugin for File operation, namely **Cordova File Plugin** (cordova-plugin-file) [4]. Files are referenced via URLs which support using platform-independent references such as *application_folder*.

3) Persistent Variables: In Android, storing and accessing persistent variables is supported via **SharedPreferences**. It allows developers to store primitive data types (e.g., booleans, integers, strings). iOS developers have two options to store persistent variables: **NSUserDefaults** and **Property Lists**. NSUserDefaults has a similar behaviour to SharedPreferences in Android. Property Lists offer more flexibility by allowing storage of more complex data structures and specification of the storage location. Cordova applications can use the **LocalStorage** API provided by the Android and iOS WebView. Although it provides a simple API, developers should be aware of several disadvantages. First, LocalStorage only supports storage of strings. More complex data structures need to be serialized and deserialized by the developer. Second, LocalStorage is known [1] to perform poorly on large data sets and has a maximum storage capacity of 5MB.

4) Sensitive Data: Android provides two mechanisms to store credentials: the **KeyChain** and the **KeyStore**. A KeyStore is bound to one specific application. Applications can not access credentials in KeyStores bound to other applications. If credentials need to be shared between applications, the KeyChain should be used. The user is asked for permission when an application attempts to access credentials in the KeyChain. Credential storage on iOS is provided by the **Keychain**. Credentials added to the Keychain are, by default, app private, but can be shared between applications from the same publisher. Cordova developers can use the credential storage mechanisms provided by Android and iOS via the **SecureStorage** (cordova-plugin-secure-storage) [7] plugin.

IV. NATIVESTORAGE PLUGIN

An important limitation of using HTML5 APIs (e.g., IndexedDB and LocalStorage) to store data in Cordova applications is that both on Android and iOS the cache of the WebView can be cleared when, for instance, the system is low on memory. This section presents NativeStorage, a Cordova plugin for persistent data object storage, mitigating the limitations of the HTML5 storage mechanisms.

A. Requirements

The requirements of the plugin are listed below:

R_1	Persistent and sufficient storage
R_2	Storage of both primitive data types and objects
R_3	Support for Android and iOS
R_4	App private storage
R_5	Responsive APIs
R_6	A user-friendly API

```
// coarse grained API
NativeStorage.setItem("reference_to_value",<value>,
     <success-callback>, <error-callback>);
NativeStorage.getItem("reference_to_value",<success-
     callback>, <error-callback>);
NativeStorage.remove("reference_to_value",<success-
     callback>, <error-callback>);
NativeStorage.clear(<success-callback>, <error-
     callback>);
```

Listing 1. NativeStorage – Coarse-grained API

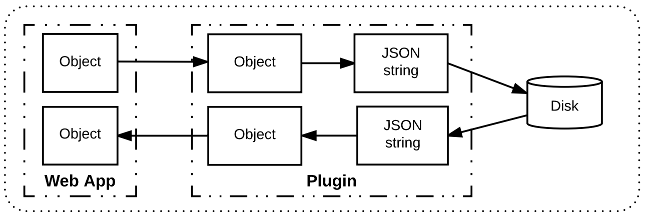
```
// fine grained API
NativeStorage.put<type>("reference_to_value",<value>,
      <success-callback>, <error-callback>);
NativeStorage.get<type>("reference_to_value",<
     success-callback>, <error-callback>);
NativeStorage.remove("reference_to_value",<success-
     callback>, <error-callback>);
```

Listing 2. NativeStorage – Fine-grained API

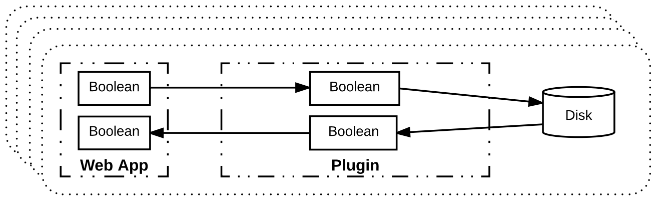
B. Realisation

The plugin consist of JavaScript and native code. The JavaScript API provides the interface to application developers. The native side handles the storage of variables using native platform APIs.

NativeStorage provides two sets of JavaScript APIs, a fine-grained and a coarse-grained API, which are both asynchronous and non-blocking. The coarse grained API (Figure 2a) provides a type-independent interface, variables are automatically converted to JSON objects via the JSON interfaces provided by the WebView and passed as string variables to the native side. When a value is retrieved, the WebView is used to convert the string back to an object. The fine-grained API (Figure 2b) provides a separate implementation for the different JavaScript types. On the native side, the variables are stored via SharedPreferences in Android and NSUserDefaults in iOS.



(a) Coarse-grained API



(b) Fine-grained API

Figure 2. NativeStorage API

C. *Evaluation*

The plugin is evaluated based on the previously listed requirements.

Persistent storage is provided via the native storage mechanisms. The documentation of the used native mechanisms doesn't state a limitation on the storage capacity. Hence, as opposed to LocalStorage, the storage capacity is only limited by the available memory on the device, satisfying R_1.

The native part of the plugin is developed for both Android and iOS. These mobile operating systems have a combined market share of 99% [15]. The used native storage mechanisms were introducted in iOS 2.0 and Android 1.0. The plugin, hence, provides support for virtually all version of these platforms used in practice, satisfying R_3.

The plugin uses NSUserdefaults and SharedPreferences to store the data in app-private locations, ensuring that the variables can not be accessed from outside the application. This satisfies R_4.

The APIs are implemented using an asynchronous non-blocking strategy, facilitating the development of responsive applications (conform R_5).

Web developers are familiar with duck typing used in languages such as JavaScript. These types of languages often have APIs that don't distinguish between data types. The coarse-grained API provides such a storage mechanism. This API is shown in Listing 6. Not all Cordova developers, however, have a Web background. Therefore, a fine-grained API (Listing 5) is provided for developers who are more comfortable with a statically typed language, satisfying R_6 and R_2. Using both the coarse- and fine-grained API, the different JavaScript data types can be stored. Developers, however, need to be aware that the object storage relies on the JSON interface of the WebView to convert the object to a JSON string representation. The WebView, for instance, does not support the conversion of circular data structures. These types of objects, hence, need to be serialized by the developer before they can be stored.

Since its release to Github [16] and NPM [5] the plugin has been adopted by many Cordova application developers. We've registered over 4000 downloads per month. Furthermore, the plugin is part of the 4% most downloaded packages on NPM. The plugin has been adopted in Ionic Native (Ionic 2) [3] and the Telerik plugin marketplace [9]. Telerik verifies that plugins are maintained and documented, thereby ensuring a certain quality.

V. Evaluation

The evaluation of the data storage mechanisms consists of two parts: a quantitative performance analysis and a security evaluation.

A. *Performance*

Developers want to be aware of the potential performance impact of using a CPT for mobile app development [21]. This section evaluates the performance of the different storage mechanisms for Cordova applications and compares the results with the native alternatives. Each storage strategy is tested by deploying a simple native and Cordova test application that intensively uses the selected storage strategy on an Android and iOS device. For Android the Nexus 6 running Android 6 was used, for iOS the IPhone 6 running iOS 9 was used. The test application communicates the test results via timing logs that are captured via Xcode for iOS and Android Studio for Android. The experiments were run sufficient times to ensure the measurements adequately reflect the performance of the tested storage mechanisms.

1) Databases:

a) Test Application: The database test application executes 300 basic CRUD operations (i.e. 100 x create, 100 x read, 50 x delete and 50 x read) of objects containing two string variables. The performance is determined by means of measuring the total duration of all the transactions. This test has been executed using the SQLite (native and Cordova), WebSQL (Cordova) and IndexedDB (Cordova) mechanisms.

b) Results and Comparison: The results are presented in Table II. The mechanism for retrieving values by means of an index clearly results in a better performance compared to the SQL-based mechanisms. This analysis shows that IndexedDB provides an efficient way of storing and retrieving small objects. WebSQL –provided by the WebView– acts as a wrapper around SQLite. This is illustrated by the performance overhead associated with this mechanism. The deprecation of the specification/development stop could also have contributed to the performance penalty. The SQLite plugin suffers from a performance overhead caused by the interposition of the Cordova framework/bridge and has consequently a noticeable performance overhead. The performance overhead introduced by the Cordova bridge is discussed in more detail in the following section.

TABLE II. Ratio of database execution time to the native (SQLite) operation duration (in %). *In iOS IndexedDB is only supported as of iOS 10.

	Android	iOS
	Nexus 6	*iPhone 6*
SQLite (Native)	100	100
IndexedDB	6.94	12.47*
WebSQL	153	128
SQLite Plugin	133	116

2) Files:

a) Test Application: The test application distinguishes between read and write operations. Each operation is tested using different file sizes, ranging from small files ($\sim 1\,\mathrm{kB}$) to larger files ($\sim 10\,\mathrm{MB}$). The performance of small files provides a baseline for file access. The performance of the read and write operations itself can be determined via the results of the large files. This test has been conducted ten times for each file size. The read and write operations consist of different steps on Android and iOS. Both the duration of the individual steps and the entire operation (i.e. read/write) is measured via timestamps. The application's memory footprint is measured via Instruments tool (Activity Monitor) in Xcode and via Memory Monitor in Android Studio.

b) Results and Comparison: The results of the timing analysis on Android and iOS is presented in Figure 3 and 4, respectively. In both Android and iOS a significant performance difference between the native and the Cordova mechanism can be observed. R/W operations via the file plugin take longer compared to the native mechanisms. On top of a performance overhead, Cordova also comes with a higher memory consumption, especially in iOS (Figure 5).

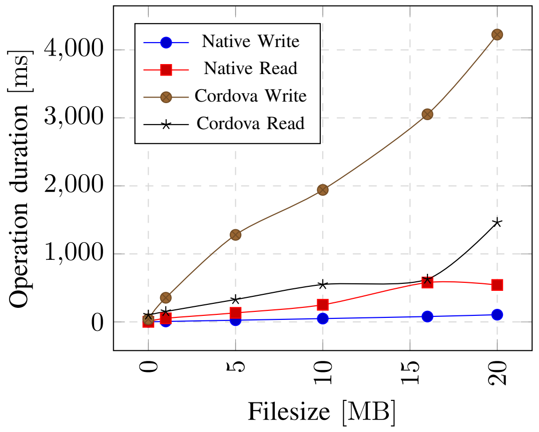


Figure 3. Duration of file operations in Android

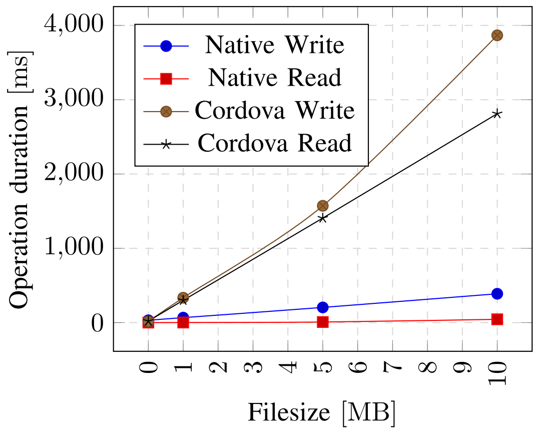


Figure 4. Duration of file operations in iOS

Speed. Tables III and IV give a fine-grained overview of the different operations executed during respectively a file read and write using the Cordova platform on Android. Tables V and VI provide the results for iOS. Before data can be sent over the Cordova bridge, it needs to be converted to a string. This can create significant overhead when large binary files such as images needs to be manipulated. Before they are sent over the bridge, the binary data is converted to a Base64 string. On Android, this is illustrated in the *Processing file* component of Table IV. Sending the data over the bridge also comprises a significant part of the overhead (i.e. *Sending over bridge*, from Table III). For small files, the overhead originates for the most part from resolving the platform-independent URL to a local path and retrieving meta-data. Similar observations can be made based on the iOS results.

TABLE III. EXECUTION TIME OF COMPONENTS ASSOCIATED WITH A READ OPERATION IN CORDOVA ANDROID (FILE PLUGIN). THE PROCEDURE "SENDING OVER BRIDGE" CONSISTS OF ENCODING, SENDING AND DECODING MESSAGES FROM THE JAVASCRIPT SIDE TO THE NATIVE SIDE.

Component	**Duration** [1 MB]		**Duration** [20 MB]	
	(ms)	(% total)	(ms)	(% total)
Resolve to local URL	58	**46**	59	7.56
Native reading	20	16	366	**47**
Sending over bridge	28	22	339	**43**
Total	126		780	

TABLE IV. EXECUTION TIME OF COMPONENTS ASSOCIATED WITH A WRITE OPERATION IN CORDOVA ANDROID (FILE PLUGIN). THE PROCEDURE "PROCESSING FILE" CONVERTS THE BYTES –AS AN ARRAYBUFFER– TO A STRING ARRAY. THE "EXECUTE CALL DELAY" REPRESENTS THE DELAY BETWEEN THE WRITE COMMAND EXECUTED IN JAVASCRIPT AND THE EXECUTION AT THE NATIVE SIDE.

Component	**Duration** [1 MB]		**Duration** [20 MB]	
	(ms)	(% total)	(ms)	(% total)
Processing file	108	**65**	1290	**56**
Execute call delay	38	**23**	632	**28**
Writing	20	12	369	16
Total	166		2291	

Memory. In iOS, applications manipulating large files will require large amounts of memory. This is illustrated in Figure 5. As shown, reading and writing a 10 MB file results in 400 MB of allocated memory. Reading and writing files larger than 10 MB can result in unstable behavior on iOS due to the large memory requirements. A solution for developers is to split large file operations in different steps.

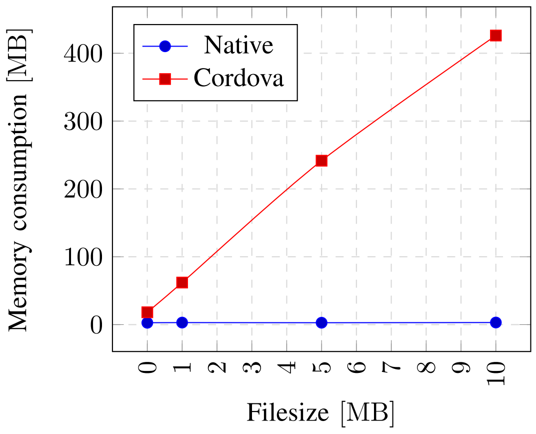


Figure 5. Memory consumption as a result of file operations in iOS

c) Conclusion: File storage on Apache Cordova comes with a number of limitation in terms of performance. This is a result of the Cordova framework/bridge technology. Allowing binary data to pass over the Cordova bridge could significantly improve the performance of plugins that perform operations on binary data. For instance, in [25] an bridging technology is

TABLE V. PERFORMANCE READ COMPONENTS IN CORDOVA IOS

Component	**Duration** [1 MB] (ms)	(% total)	**Duration** [10 MB] (ms)	(% total)
Resolve to local URL	11	3.56	16	0.6
Native reading	13.98	4.52	70	2.47
Arguments to JSONArray	202.77	**65.62**	2037.93	**71.88**
Sending over bridge	59.93	**19.39**	587.19	**20.71**
Total	309		2835	

TABLE VI. PERFORMANCE WRITE COMPONENTS IN CORDOVA IOS

Component	**Duration** [1 MB] (ms)	(% total)	**Duration** [10 MB] (ms)	(% total)
Processing file	266	**97**	2614	**96**
Native writing	7	3	96	4
Total	273		2710	

presented that allows access to native device APIs in HTML5 applications via WebSockets and HTTP servers, supporting the use of binary data.

3) Persistent variables:

a) Test Application: The performance is examined via storing and retrieving string values. The total duration of storing and retrieving a thousand variables is measured. The average storage and retrieval time is used to compare the different storage mechanisms. The Cordova mechanisms are LocalStorage and NativeStorage. These are compared to NSUserDefaults (iOS), Property Lists (iOS) and SharedPreferences (Android).

b) Results and Comparison: All mechanisms have an execution time under 1 ms, with the exception of NativeStorage and Property Lists. The set operation takes around 1.9 ms, the get operation takes less than 1 ms. NativeStorage is the only mechanism which uses the Cordova bridge and framework, introducing a certain overhead. However, the NativeStorage API is asynchronous, hence, developers can continue processing while the value is being stored. The listed measurements include the time until the callback is fired. Property Lists load an entire file in an array, after which individual parameters can be read. As a consequence, the performance of the get operation, which takes 9.83 ms, is worse compared to the native alternatives. SharedPreferences and NSUserDefaults also load all parameters in memory, but this is done during the initialisation phase of the application, which is not incorporated in the measurements.

B. Security

On both Android and iOS the security of storage mechanisms strongly depends on the storage location and the platform's backup mechanisms. Data stored inside the sandbox of the application is only accessible by the application. However, the backup mechanisms used in iOS and Android can result in the exposure of sensitive data [27, 29, 28], or potentially exhausting the limited cloud storage capacity. On iOS, this can result in the rejection of the application (conform the Data Storage Guidelines [14]). On Android, data stored inside the application sandbox (e.g., the WebView's storage) is included if a backup is taken. The Backup API of Android can be used to explicitly blacklist data that should not be backed up. On iOS, whether or not a file is included in the backup depends on the folder in which it is stored. For instance, by default, Cordova stores the WebView's data in a folder that allows backups. This behavior can, however, be changed by modifying a Cordova parameter.

1) Databases: All database mechanisms are by default private to the application and can be backed up on both mobile platforms, with the exception of the SQLite plugin in iOS. The plugin initially followed the default behaviour, but as a security measure the default storage location of the plugin in iOS was changed to a directory which is not backed up. This SQLite plugin also has an encrypted alternative, i.e. **cordova-sqlcipher-adapter**. This alternative provides a native interface to SQLCipher, encrypting SQLite databases via a user-supplied password.

2) Files: In iOS files are protected by a protection class. Each of these classes corresponds to different security properties. As of iOS 7, all files are by default encrypted individually until first user authentication. The file plugin doesn't allow changing this default behaviour. Native, each file can be secured using a protection class best suited for the security requirements of that file. The plugin allows the developer to choose between folders that are public/private and backup-enabled/disabled. However, on Android backup-disabled locations can be accessed by other applications.

3) Persistent variables: All persistent variable storage mechanisms are private to the application and included in backups on both mobile platforms, with the exception of Property List. Property lists can be stored in arbitrary locations, and can be backed up depending on the specified location.

4) Sensitive Data: The Secure Storage plugin provides storage of sensitive data on Android and iOS. On iOS, the plugin uses the SAMKeychain [2] plugin which provides an API for the native iOS Keychain. The plugin allows app-global static configuration of the KeyChain items' accessibility. This could entail a security risk, as it does not allow fine-grained protection of individual items. When a user backs up iPhone data, the Keychain data is backed up but the secrets in the Keychain remain encrypted with a phone-specific key in the backup. The Android KeyChain only allows storage of private keys. Hence, for storing other tokens such as passwords or JWT tokens, an additional encryption layer is used. The plugin generates a key that is stored in the KeyChain and used to encrypt/decrypt sensitive data. The KeyChain on Android is not included in backups.

VI. CONCLUSIONS

This paper presented an assessment of data storage strategies using the mobile cross-platform tool Cordova. An in-depth analysis was performed on the API coverage of the available data storage mechanisms in Cordova and Native applications. Based on the analysis, an additional Cordova storage plugin was developed that improves the storage of persistent variables.

Furthermore, the performance and security of the available storage mechanisms were evaluated. Our performance analysis shows that using the Cordova bridge comes with a significant performance penalty. Hence, the WebView's JavaScript API should be used when possible. However, apart from performance, other parameters such as functionality and security can have an impact on the selection of the storage mechanism.

Databases. If access to a full fledged SQL database is required, the SQLite plugin should be used. However, in most mobile applications, the functionality provided by the significantly faster IndexedDB interface of the WebView is sufficient.

Variables. As described in Sections IV and V, it is recommended to use NativeStorage for storing persistent variables, since LocalStorage does not guarantee persistence over longer periods of time. This type of storage is often used to store preferences. Preferences are typically only accessed once or twice during the life cycle of the application. Hence, the performance overhead of NativeStorage does not have a significant impact on de performance of the application.

Files. The WebView does not provide a file storage API. Hence, developers have to use the core plugin, Cordova File Plugin (`cordova-plugin-file`).

Sensitive data. The security analysis presented in Section V-B shows that plugins such as SecureStorage offer increased security compared to the WebView's JavaScript API because they benefit from the platform's native secure storage APIs. It is therefore recommended to use a plugin such as SecureStorage to store sensitive data.

Future work on this topic can include an in-depth analysis of the CrossWalk WebView. Currently, Cordova applications suffer from a major performance penalty every time the JavaScript bridge is accessed. CrossWalk has its own plugin mechanism, which could show better performance than Cordova plugins.

REFERENCES

[1] Cordova storage documentation. URL https://cordova.apache.org/docs/en/latest/cordova/storage/storage.html.
[2] Samkeychain. URL https://github.com/soffes/SSKeychain.
[3] Nativestorage in the ionic framework documentation. URL http://ionicframework.com/docs/v2/native/nativestorage/.
[4] Cordova file plugin npm website, . URL https://www.npmjs.com/package/cordova-plugin-file.
[5] Nativestorage plugin npm website, . URL https://www.npmjs.com/package/cordova-plugin-nativestorage.
[6] Sqlite plugin npm website, . URL https://www.npmjs.com/package/cordova-sqlite-storage.
[7] Securestorage plugin npm website, . URL https://www.npmjs.com/package/cordova-plugin-secure-storage.
[8] Progressive web apps. URL https://developers.google.com/web/progressive-web-apps/.
[9] Cordova plugins in the telerik marketplace. URL http://plugins.telerik.com/cordova.
[10] Web sql database documentation. URL https://dev.w3.org/html5/webdatabase/.
[11] Can i use ... ? URL http://caniuse.com/.
[12] Cordova plugins website. URL https://cordova.apache.org/plugins/.
[13] Crosswalk website. URL https://crosswalk-project.org.
[14] ios data storage guidelines. URL https://developer.apple.com/icloud/documentation/data-storage/index.html.
[15] Smartphone os market share, q2 2016. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2015. access date: 20/10/2016.
[16] Cordova plugin nativestorage, 2016. URL https://github.com/TheCocoaProject/cordova-plugin-nativestorage.
[17] Matteo Ciman and Ombretta Gaggi. Evaluating impact of cross-platform frameworks in energy consumption of mobile applications. In *WEBIST (1)*, pages 423–431, 2014.
[18] Isabelle Dalmasso, Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. Survey, comparison and evaluation of cross platform mobile application development tools. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pages 323–328. IEEE, 2013.
[19] Henning Heitkötter, Sebastian Hanschke, and Tim A Majchrzak. Evaluating cross-platform development approaches for mobile applications. In *International Conference on Web Information Systems and Technologies*, pages 120–138. Springer, 2012.
[20] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. End users' perception of hybrid mobile apps in the google play store. In *2015 IEEE International Conference on Mobile Services*, pages 25–32. IEEE, 2015.
[21] Vision Mobile. Cross-platform developer tools 2012, bridging the worlds of mobile apps and the web, 2012. access date: 13/04/2016.
[22] Vision Mobile. Cross-platform tools 2015, 2015. URL http://www.visionmobile.com/product/cross-platform-tools-2015/. access date: 13/04/2016.
[23] Vision Mobile. Developer economics state of the developer nation q1 2016, 2016. URL http://www.visionmobile.com/product/developer-economics-state-of-developer-nation-q1-2016/. access date: 13/04/2016.
[24] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. Comparison of cross-platform mobile development tools. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*, pages 179–186. IEEE, 2012.
[25] Arno Puder, Nikolai Tillmann, and MichałMoskal. Exposing native device apis to web apps. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, MOBILESoft 2014, pages 18–26, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2878-4. doi: 10.1145/2593902.2593908. URL http://doi.acm.org/10.1145/2593902.2593908.
[26] Florian Rösler, André Nitze, and Andreas Schmietendorf. Towards a mobile application performance benchmark. In *International Conference on Internet and Web Applications and Services*, volume 9, pages 55–59, 2014.
[27] Peter Teufl, Thomas Zefferer, and Christof Stromberger. Mobile device encryption systems. In *28th IFIP TC-11 SEC 2013 International Information Security and Privacy Conference*, pages 203 – 216, 2013.
[28] Peter Teufl, Thomas Zefferer, Christof Stromberger, and Christoph Hechenblaikner. ios encryption systems - deploying ios devices in security-critical environments. In *SECRYPT*, pages 170 – 182, 2013.
[29] Peter Teufl, Andreas Gregor Fitzek, Daniel Hein, Alexander Marsalek, Alexander Oprisnik, and Thomas Zefferer. Android encryption systems. In *International Conference*

on Privacy & Security in Mobile Systems, 2014. in press.

[30] Michiel Willocx, Jan Vossaert, and Vincent Naessens. Comparing performance parameters of mobile app development strategies. In *Proceedings of the International Workshop on Mobile Software Engineering and Systems*, pages 38–47. ACM, 2016.

[31] Yanyan Zhuang, Jennifer Baldwin, Laura Antunna, Yagiz Onat Yazir, Sudhakar Ganti, and Yvonne Coady. Tradeoffs in cross platform solutions for mobile assistive technology. In *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, pages 330–335. IEEE, 2013.

An Analysis of Mobile Application Update Strategies via Cordova

Cristiano Inácio Lemes, Michiel Willocx and Vincent Naessens
Faculty of Engineering Technology, MSEC, imec-DistriNet,
KU Leuven, Technology Campus Ghent,
Gebroeders Desmetstraat 1, 9000 Ghent, Belgium
{inaciolemes.lemes,*surname.name*}@kuleuven.be

Marco Vieira
CISUC – Centre for Informatics and Systems
FCTUC – University of Coimbra
3030-290 Coimbra, Portugal
mvieira@dei.uc.pt

***Abstract*—The demand for mobile apps is increasing steadily. To maximize revenue in Business to Customer (B2C) settings, multiple platforms and devices must be supported, which leads to increased development cost. Mobile App Cross Platform Tools (CPTs) tackle this problem as they allow to deploy and run a single codebase on multiple platforms. Cordova is a popular framework for cross-platform development. Multiple plugins support often recurring concerns. One prototypical example is update plugins. This paper focuses on the assessment of update plugins in Cordova, supporting the distribution of code updates. The paper evaluates the most commonly employed ones and compares them against traditional version updates with respect to security and relevant quality parameters. We show that improvident plugin selection and bad developer practices may seriously undermine the security and quality of the mobile apps.**

Keywords–Cross-Platform Tools; Security.

I. Introduction

The power of mobile devices such as smartphones and tablets has begun to rival personal computers over the last decade. The number of available devices has been increasing and, simultaneously, the amount of apps available in app stores has grown significantly, ranging from business-critical (e.g., banking) to social media [1], [2], [3], [4]. From the operating systems currently available on the market, two players have a substantial market share: Android and iOS. App developers must at least support those platforms to reach a large number of end users, although this fragmentation places a significant burden on the overall development cost. In practice, app development companies must acquire expertise in both iOS and Android development, and the development cycle must be undertaken for two distinct platforms.

Mobile App Cross-Platform Tools (CPTs) allow to deploy a single codebase on multiple platforms, and a wide variety of CPTs exist today [5]. Cordova [6] – formerly known as Phonegap – is very popular and applies web-to-native technology. This CPT enables the development of mobile apps using JavaScript, HTML5 and CSS3 instead of using the native development language [7], [8]. The app code is wrapped into a stand-alone application integrated with a Webview. Cordova offers a set of plugins for accessing device sensors like cameras, calendar, and GPS. Each plugin is split into two parts: (i) the native code which accesses the device feature and (ii) the web code that creates an interface between the application and the native code.

Update plugins are frequently used in Cordova, as they support updating application code without having to upload a new version to the platform's app store (this strategy only enables the updating of webcode, as native code cannot be changed via any channel except the app store). Although multiple update plugins exist and may therefore be integrated within a Cordova app, different approaches are employed internally to support web code updates.

This paper assesses and compares the major update plugins considering their functional behaviour and impact on qualitative properties. We selected major security, user experience, and performance criteria of the Cordova update plugins based on in-depth interaction with Small and medium-sized enterprises (SMEs) [9] The selection of the most appropriate plugin is demonstrated as being crucial to increase the overall app quality and that not all plugins are equally trustworthy. For example, a bad selection strategy can undermine an app's overall security. Finally, the paper compares the Cordova update strategies against more traditional Android version updates. This paper shows that hot code updates offer a viable solution for minor updates and bug fixes to applications. However, large version updates that introduce new permissions or add new plugins require the user to go through the OS supplied application update process. Moreover, using plugins for remote code updates can potentially introduce additional vulnerabilities. Note that the scope of this work is updating strategies on the Android platform. Android was selected for this research because many update plugins only have Android support. Nevertheless, most ideas presented in this work also apply for iOS, as described in Section VI.

The remainder of this paper is structured as follows. Section II discusses background and related work. A classification of code update strategies in the context of mobile apps in addition to an overview of Cordova plugins selected for this study are detailed in Section III. Section IV lists the criteria used when assessing the update strategies. The evaluation of each strategy and a comparison among all strategies is provided in Section V. Section VI reflects on the results. Finally, conclusions are drawn in Section VII.

II. Background and Related Work

Web apps allow servers to host their code and clients can launch the code in their browser by typing in the correct URL. This is a straightforward means of offering content or services to smartphone or tablet users by way of a single codebase. The code consists of web technologies like HTML, CSS and Javascript and may be easily updated without app store intervention. Also, end users are not required to install an additional app. Although this approach is highly flexible, it is also associated with drawbacks. First, the availability and responsiveness of the application is dependent upon the Internet connection of the mobile device. Second, mobile browsers do not support advanced access to hardware sensors and data, which constrains functionality and negatively impacts user experience.

Hybrid approaches [10] are applied in many CPTs and aim at combining the advantages of web technologies and native functionalities. This requires a native web container embedded in the application to allow web code to be executed on multiple platforms. Native capabilities are accessible via a Javascript bridge and native code execution enables access to device sensors. A hybrid application must be included in the app store and subsequently installed on the user's device. Cordova is the most popular hybrid approach available nowadays [7].

Recent research has focused on assessing and comparing different CPTs in both a qualitative and quantitative way. Heitkötter et al. [6] evaluate four CPT strategies and compare them against native app development. Their work focuses on two major characteristics: *infrastructural support* and *development*. The latter covers an analysis of all development cycle steps. Assessments are completed using a scale from 1 (*very good*) to 6 (*very poor*). Rieger and Majchrzak [11] build further on this work and propose extensions and revisions for evaluations CPTs. They compared two CPTs and applied their assessment to multiple devices. The evaluation criteria was split into four groups: *infrastructure*, *development*, *app*, and *usage*. A *weight* was also assigned to each criterion for each CPT. A detailed evaluation of CPT performance was realized by Willocx et al. [12]. Ten CPTs were assessed and compared against native development using multiple performance criteria. The assessment was performed on multiple iOS, Android and Windows Phone devices. Other studies focus on selecting the most feasible CPT for a specific application or set of applications [7], [8], [13]. All these contributions focus on usability and performance. However, update strategies in particular and plugins in general are not evaluated from a research perspective.

Many mobile applications rely on the Internet to download or upload content. This makes them potentially vulnerable to both passive and active attacks [14]. De Ryck et al. [15] analyze such network attacks. Vashisht et al. [16] propose splitting mobile threats into three categories: *application-based threats* are vulnerabilities concerning applications installed on the device; *web-based threats* expose security issues in the mobile browser and applications which download content from the Internet; and *network-based threats* originate from the mobile or local wireless network. In practice, mobile applications that access the Internet are potentially vulnerable to any of these threats.

Other research focuses on assessing the security of major mobile operating systems. La Polla et al. [17] offer an overview of mobile threats and vulnerabilities before presenting possible solutions to such threats. Peijnenburg [18] studied security considerations concerning Android. Bhardwaj et al. [19] present an in-depth security comparison between Android and iOS.

This paper evaluates and compares alternative *Update Strategies* for mobile applications developed with *PhoneGap/Cordova*. It focuses on the strengths and constraints related to security, user experience and performance of multiple Cordova update plugins. Moreover, we compare our findings to traditional version updates in Android.

III. Update Strategies Overview

Mobile apps may be upgraded after they are published in an app store. Multiple reasons can trigger code modifications, including the addition of new features, the modification of graphical user interface, and fixing defects, for example. In practice, code update strategies can be classified according to three categories: *installing* a new version, *storing* new mobile code in a client side dedicated folder, and *loading* updated code from a web server. These approaches are discussed below in greater detail. Our analysis focuses on plugins that are available at the official Cordova plugin store [20].

a) Installing updates: The developer submits a new installation file to a server and the end users may upgrade the app by installing this file on their devices.

- *Google Play* [21] is the default store for Android applications, but other marketplaces exist [4]. APK files are submitted to the store and published shortly after human revision has taken place.
- Although the Cordova framework does not support modifications in the native code of the application, a dedicated plugin *Cordova-plugin-app-update* enables updating the application by executing an installation file. The installation file can reside at any server selected by the developer.

b) Storing modified web code: The developer submits a set of files containing app code to a server, after which the app may download and store them in the device. Cordova uses a dedicated *www* folder to store, amongst others, business logic and user views. New, updated web code, cannot be stored in this folder because it is read-only. Hence, update plugins store updates outside of the applications. Four frequently used plugins apply this strategy:

- The most frequently downloaded Cordova update plugin according to the plugin store is *cordova-plugin-meteor-webapp*. This is a new version of *meteor-cordova-update-plugin* that fixes some bugs while introducing new features such as performing a rollback to the previous version when the new version is unstable. Any server can host code updates.
- *Cordova-plugin-code-push* is developed by Microsoft and is the second most frequently downloaded update plugin belonging to this category. Each new app release is stored on a server hosted by Microsoft. *CodePush CLI* [22] is a tool that helps developers managing app updates.
- The *cordova-hot-code-push-plugin* is the third most frequently downloaded plugin. Updates are hosted on an Amazon server. *Cordova Hot Code Push CLI* [23] assists developers on managing new releases of the application.
- Finally, *cordova-plugin-dynamic-update* does not employ any mandatory server, so developers can host the update on any server. This plugin only implements the most basic functionalities i.e. downloading new content from the Internet and pointing the WebView to the new files.

c) Loading remote code updates at runtime: This strategy downloads remote code each time the app is running. The user launches the application using a browser that connects to the server from which the application is loaded.

IV. Evaluation Criteria

Two major stakeholders may be identified in the context of the update process, namely the *developer* and the *end user*. The developer is responsible for upgrading a mobile app and submitting the upgrade to the server. The end user retrieves new releases when they are available on the server and integrates it in the mobile app. At least two components (or hardware platforms) are involved in a upgrade operation: the *server* and the *mobile*. The server stores new application releases sent by the developer, after which they may be downloaded to a mobile device. This is based on two main operations: *submit* and *retrieve*. In practice, developers submit new app code to the server, whereas end users retrieve the update some time afterwards. Both the server and the mobile device need Internet access to perform these operations.

This section lists the set of properties that are used to compare alternative update strategies. It resulted from compilation of feedback offered by mobile app developers in the scope of a technology transfer project CrossMoS [9].They are split in two groups: *Security Criteria* and *Quality Criteria*. The former concerns the security aspect of the update process, while the latter focuses on user experience and performance.

A. Security Criteria

Figure 1 shows the interaction between the stakeholders and the components involved in the update process, as well as security concerns and their relation to the update process.

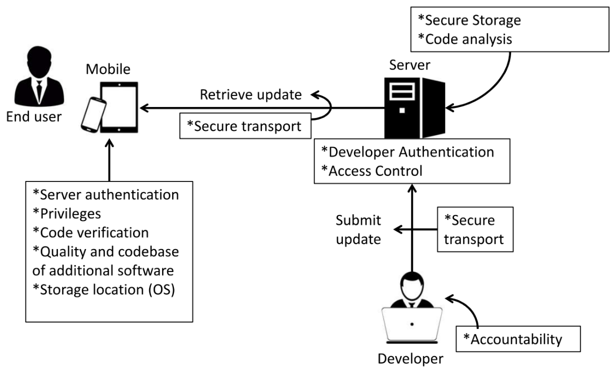


Figure 1: Security Parameters

Secure transport is a major security concern during both the submission and retrieval phase. Inappropriate communication properties can compromise mobile app security and even the device itself after the upgrade. An attacker should not be able to modify the code in transit, typically attempted by Man-In-the-Middle (MiTM) attacks (see details in the Section VI).

Developer authentication is an important security concern during the submission phase. It discourages misbehaving entities from uploading malicious code. Depending on the specific authentication strategy used, multiple updates may be linked to the same (pseudonymous) user, or even linked to an identifiable person or organization.

Accountability goes one step further than authentication and requires a stakeholder to be held accountable by a dispute handler (such as a law enforcement entity) for malicious behaviour. This study focus particularly on the accountability of developers themselves. More specifically, we evaluate whether developers can be held accountable for the submission of malicious code to the server. Appropriate authentication can already identify the individual behind a malicious submission, but only signed code can be probably linked to a physical entity.

Access control measures are implemented on the server side and restrict the possible actions that may be taken by stakeholders when accessing the server. Only authenticated developers and, maybe, collaborators or employees within the same organization, can modify application code (potentially with different access restrictions).

Code analysis/verification can be performed by the update server to check for malicious behaviour in the submitted code. Malicious code potentially exploits privileges (or permissions) already granted to apps and likely leaks sensitive personal data stored in the mobile device to an attacker. Moreover, it can disturb the correct functioning by draining the battery or stealing money in the background. The update server should check whether the code performs any malign tasks before making it available for download.

Secure storage implies that no third party or process running on the server can modify the app code without the consent of the developer. Insecure storage potentially results in malware attacks such as those aforementioned.

Server authentication should be mandatory during the retrieval phase. Mobile devices must be capable of verifying the authenticity of the server hosting the code updates to avoid malicious code being downloaded in the context of the client app.

Our assessment also focuses on how the update strategy deals with *privileges* (or permissions), as the amount of sensitive personal data that can be leaked by an app is highly dependent upon the privileges it is granted. Therefore it is crucial to implement measures that maintain the number of privileges granted under control. This property assesses what permissions are granted to the app if the update plugin is installed.

The overall security of the update strategy also depends on the *quality and codebase of the additional software* that must be installed to support updates. Some update strategies rely solely upon OS software components, while others require the installation of additional software plugins that may be composed by integrating third party software libraries. Integrating them can also increase the number of permissions that are required to run the app. In practice, both plugins and the software libraries on which they rely may contain vulnerabilities. In its turn this can increase the impact of successful attacks both from privacy as well as security point of view.

The *storage location* of the mobile device may also have an impact on the security status of the update strategy. Code can either be stored in shared memory or solely be accessible by the app itself, which obviously provides different levels of security.

B. Quality Criteria

The set of quality criteria is depicted in Figure 2. They primarily focus on user experience, performance, and overall code quality.

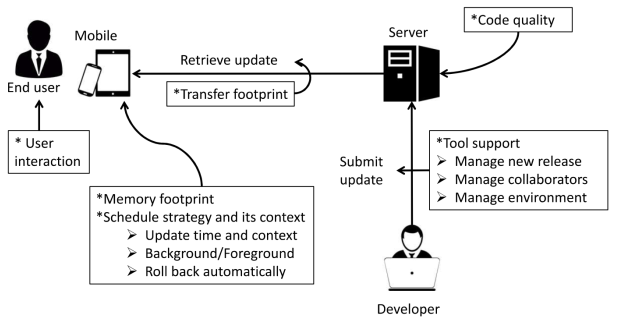


Figure 2: Quality Parameters

For each update strategy, the extent to which *user interaction* is required is evaluated. Either the update can be performed fully transparently, or consent must be explicitly given by the user. If user interaction is required, the type and quality of information returned to the end user is evaluated. Similar, the properties of the *schedule strategy and its context* are evaluated. Such properties can influence user experience. First, the update time and context may differ between alternative approaches. For instance, large (automatic) updates over an xG network are associated with substantial costs and consume unnecessary bandwidth, especially in a roaming context. Second, the update process can either lock the application until finished, or allow the user to use the application during the update. Finally, the possibility of rolling back unstable updates is investigated. This may be required if an update results in crashes on certain devices.

Memory and transfer footprint define the amount of storage required to store both the plugin and updated code and the bandwidth needed to retrieve the update, respectively. Compressed code can be transferred more efficiently and hence has less impact on the time required to install the update than non-compressed code. Similarly, downloading either a complete new version or an increment with respect to the previous version may be needed.

Code quality is often impacted by developer practices and may potentially result in performance decreases or even app crashes after installation. We define code quality in this work from the user's point of view, and evaluate if the mechanism's characteristics meet the user's needs [24]. Code analysis performed by update servers before the update is actually made available and helps avoiding (un)intentionally introduced bugs.

The final quality criterion focuses on the *tool support* available for the update process, like support associated with the management of new releases, enabling collaboration, or tools for covering the entire development cycle (such as design, implementation, and testing).

V. RESULTS AND COMPARISON

This section details the results of the assessment of the update strategies introduced in Section III. First, a general overview of each strategy is provided, after which the security and quality criteria are discussed. In the remainder of this work, the update strategies are denoted as shown in Table I.

Table I: UPDATE STRATEGY NOTATIONS

Update Strategy	Notation
Installation	I
Google Play	I1
Cordova-plugin-app-update	I2
Storage	S
Cordova-plugin-code-push	S1
Cordova-hot-code-push-plugin	S2
Cordova-plugin-meteor-webapp	S3
Cordova-plugin-dynamic-update	S4
Loading	L

A. Overview of update strategies

1) Installation (I): New application code can be stored in the commercial store managed by the OS provider or in another marketplace selected by the developer. Software on the mobile can poll whether a new version is available or new version notifications can be sent to the mobile. Within this class, an entire new version is typically submitted by developers and can subsequently be downloaded.

Developers submit new versions of an app to the standard *Google Play* (I1) store after having assigned it a higher version number. Google reviews the application code using two different approaches. First, an automatic malware scan is performed. Second, a human reviewer investigates possible violations with Google policy rules [25]. Two strategies can be adopted to retrieve a software update: apps are either updated automatically by the Google Play application or the update process is manually controlled by the owner of the device. Note that user consent is always asked when new permissions are required. To save bandwidth, only files differing from the previous version installed on the device are downloaded [26]. Finally, the *cordova-plugin-app-update* (I2) supports the installation of new executables without visiting the default app store, even though the OS does not support this behavior by default and even discourages it. For this, device owners must modify OS configuration settings and the execution of installation files from unknown sources must be allowed. The content of the file is not reviewed, unlike the Google Play strategy. When the app initializes, the plugin checks the version number of the installed app against that of the version available at the update server. A new APK is installed if an update is available.

2) Storage (S): Native mobile apps are typically installed in a read-only folder, which means that only the OS may alter its content. Cordova uses this folder only to store the core files of mobile apps (the native code), including the Webview. This Webview can be pointed to a folder with write permissions and the Web code is then stored and updated via this folder. Any updates to core files can only be performed by submitting and installing a new version. Each plugin follows a slightly different strategy, as discussed next.

The *cordova-plugin-code-push* (S1) checks for updates at a dedicated server and returns a URL from which the latest update can be downloaded. The user must give their consent after which a ZIP file containing the new web code can be downloaded.

The *cordova-hot-code-push-plugin* (S2) checks for updates by reading a configuration file from the server containing a hash of all web code files. The plugin identifies the files that were changed and only updates those files. When the application is launched for the first time, all web code files are copied in the writable *www* folder. The *cordova-plugin-meteor-webapp* (S3) employs a similar approach.

Finally, the *cordova-plugin-dynamic-update* (S4) downloads the entire content of the *www* folder each time an update is available. The previous update of the application is overwritten.

3) Loading (L): App updates occur almost instantly. The app is launched using a browser that loads the content from the server and the only requirement is for the developer to submit the new code to the server. The primary disadvantage of this strategy is that the application only works when an Internet connection is available. In the worst case scenario, the mobile device must contact the server every time the user requests a new page.

B. Security parameter assessment

Malicious individuals may exploit vulnerabilities and thereby compromise the security of mobile app updates. Bad design decisions as well as weak implementations can result in security vulnerabilities. Table II summarizes the results of the analysis of the security concerns listed in the previous when assessing update strategies.

Table II: EVALUATION OF STRATEGIES ACCORDING TO SECURITY PARAMETERS

Criteria/Strategy	I1	I2	S1	S2	S3	S4	L
Submit update							
Accountability	yes	no	no	no	no	no	no
Secure transport (submit)	yes	no	yes	yes	no	no	no
Developer Authentication	yes	no	yes	yes	no	no	no
Access control	yes	no	yes	yes	no	no	no
Secure storage (server)	yes	no	yes	yes	no	no	no
Code analysis (server)	yes	no	no	no	no	no	no
Retrieve update							
Secure transport (retrieve)							
HTTPS forced	yes	no	yes	yes	no	no	no
SSL Certificates checked	yes	yes	no	yes	yes	yes	yes
Hash check after download	yes	no	yes	yes	no	no	-
Server Authentication	yes	no	no	no	no	no	no
Privileges	no	yes	yes	yes	yes	yes	no
Code verification (device)	yes	no	no	no	no	no	-
Additional software	no	yes	yes	no	no	no	-
Secure storage (device)	yes	yes	yes	yes	yes	yes	-

Two approaches are possible for setting up an update server. First, the update server can be fixed in the update plugin. This means that each developer that uses the plugin must push updates to that third-party server. The server often represents a reliable Cloud platform (examples of such include Android Play Store or Microsoft Azure) associated with a series of advantages like high availability and small security vulnerability due to regular software updates. Second, some plugins enable developers to select their own storage server. Therefore, security is highly dependent upon both the reputation of the developer and selected server. The plugins that fix a specific update server (I1, S1, and S2) also provide a secure communication channel between the developer and the server. Other strategies are dependent upon the server hosting code updates. Storage security also relies on the trustworthiness of the server that hosts the updates. Similarly, some plugins rely on existing back-end infrastructure and, hence, benefit from the access control procedures offered by the used infrastructure. Only the owner (and possibly collaborators) can modify code within a particular application. The quality of other plugins depends on the particular update server selected. Therefore, if the developer is free to choose an update server, they must select one that has good security practices. Note that the end user is unaware of the selected server.

For the security of the end user, it is also important that the update reaches the device securely. The Android Play store is tightly integrated with the underlying operating system and offers all required security measures, as a secure HTTPS connection is strictly used for all communication.

Secure communication is not always guaranteed when using a plugin approach. In fact, some plugins do not force the use of HTTPS on the developer. S1 and S2 are an exception to this. They respectively make use of the Amazon and the Microsoft Azure platform to distribute updates, and therefore always make use of HTTPS. In the other cases, the responsibility to use HTTPS lies with the app developer.

When using an HTTPS connection, it is mandatory to check the SSL certificates before accepting communication. Plugins I2, S2 and S4 have custom native code for downloading the update. They make use of Android's `HTTPUrlConnection`, `UrlConnection` and `HTTPClient`, respectively, and are therefore secure when HTTPS is used. Android provides the necessary checks to secure the connection once an HTTPS domain is detected in the URL (for more information [27], [28]) Plugin S1 contains no native code to download the update but instead automatically includes the *file-transfer-plugin* in the application when installing the update plugin, and contains Web code to call the plugin to download the update. For developers, it is important to know that the *file-transfer-plugin*'s `download()` method contains an "AllowAllHosts" parameter. When this parameter is set to true, it overrides Android's HostnameVerifier's [29] `verify()` method to always return true. In this case, SSL certificates are blindly accepted without any checks, which raises severe security threats. Plugin S1 sets this parameter to true and is therefore not secure. Developers using this plugin should manually change this variable to false in the its web code in order to ensure secure communication. Plugin S3 contains no code for the actual download of the update. The straight-forward way to download the update in this case is to include the *file-transfer-plugin* in the application manually. Some example code provided by the developers of the plugin also applies this strategy. In this example, no AllowAllHosts parameter is specified, thus the default value is false. Hence, developers can use this example to provide secure communication.

After downloading the update, I1, S1 and S2 make use of a hashing function to determine if nothing was changed during the download. This acts as an additional measure against MiTM attacks, and also ensures that no communication errors occurred.

When loading web content in the Webview or a browser (L), the security depends only on whether HTTPS is used or not. When HTTPS is used, the communication is handled by the Webview and is therefore secure.

Strategy I1 supports developer accountability. It relies on the OS platform builder and requires one to update the submission via the Android store, and developers must sign the code before submitting it to the server. This procedure verifies if the files were submitted by a particular developer (or organization). In addition, Play store examines the code submitted by the developer before publishing the application on its store. In practice, it looks for malicious code and violation of system rules in the application files. If an unacceptable security or privacy threat is discovered the application is not published, the developer can be judged and their profile removed from the store. Other storage platforms do not explicitly mention code revision. Except for the Google Play Store downloads and updates [30], no other strategy applies client side code verification, despite there being some tools and methods currently available to execute it. Such tools could mitigate threats introduced by selecting unreliable update servers.

The Google play store (I1) is the only update mechanism that requires the user to authenticate to the update server. Android requires the user to be logged in before allowing downloading or updating applications.

The OS coordinates code updates within strategy I1. It saves new files in a folder that cannot be overwritten (unless a new code update is available) by the application or possible malware. All other strategies retrieve updates stored in memory only accessible by the app itself. This means that other, potentially malicious apps, cannot access or modify that code. Hence, once updates are installed on the mobile device, the security depends upon the trustworthiness of the OS version.

Update plugins often require extra privileges to support their tasks. Two permissions that every Cordova plugin requires are `internet` and `write_external_storage` permission. The former allows apps to open network sockets and is enabled by default in the Cordova framework; the latter allows apps to write content to external storage on the device. These permissions can be abused by locally installed untrusted code which downloads malicious content from the Internet before writing it to the device and altering the application to make it use this malicious content. The `mount_unmount_filesystems` permission is used by I2 to manage the file systems for removable storage. S2 also requires the `access_network_state` and `access_wifi_state` permissions for accessing information concerning network status.

Additional Cordova plugins (also called dependencies) are installed automatically when strategies I2 or S1 are applied. I2 installs an additional Cordova plugin for accessing the OS version. S1 relies on a set of seven other plugins that handle sending notifications to users, accessing configuration information from the device, handle runtime permissions, file management, among others. Note that installing such plugins weakens the privacy properties of the app and weak implementations may ultimately lead to vulnerabilities that can, in turn, compromise app security. Although S3 does not automatically introduce any dependencies, it requires additional plugins to work properly.

C. *Quality parameters assessment*

Table III provides an overview of the quality related parameters for each update strategy. Strategies S3, S4, and L neither require user confirmation nor notification of the end user regarding code updates. All other strategies ask the user to confirm the update. I1 also provides feedback to the user; I2, S1, and S2 do not inform the user about the scope of the updates, although certain code updates may negatively impact the user's privacy or security (as learned from the security assessment).

Table III: EVALUATION OF STRATEGIES ACCORDING TO QUALITY PARAMETERS

Criteria/Strategy	I1	I2	S1	S2	S3	S4	L
User interaction	yes	yes	yes	yes	no	no	no
Schedule strategy and its context							
Update time and context	req.	any	any	any	any	any	any
Lock the application	yes	yes	no	no	no	no	-
Roll back automatically	yes	no	yes	yes	yes	no	yes
Memory and Transfer footprint	++	++	+	-	-	+	++
Code Quality Verification	yes	no	no	no	no	no	no
Tool Support	yes	no	yes	yes	no	no	no
Manage new release	yes	-	yes	yes	-	-	-
Manage collaborators	yes	-	yes	no	-	-	-
Manage environment	yes	-	yes	yes	-	-	-

I1 supports automatic updates of non-active applications. Other approaches typically check whether an update is available when it launches or resumes, automatically. Alternatively, the developers may provide a button in the app to begin the update process. Strategy L receives updates when the page is (re)loaded. Cordova update strategies S_x do not enforce any constraints on running applications. In contrast, I1 and I2 lock the application until the update process has ended.

Strategy I2 and S4 do not support roll backs to a previous valid version in case of crashes caused by the update. When such a situation arises the end users must reinstall the application. All other strategies perform roll backs automatically when an error occurs. Crashes in strategy I1 occur rarely, due to heavy code revision by Google and the reporting of unexpected behavior by the end-users [25]. This strategy guarantees that a stable version of the application is available on the store until the developer provides a valid new version.

A large amount of code and data is potentially downloaded during the update process. Strategies I1 and S2 support contextual constraints related to updates. For instance, these strategies permit developers to specify that updates may only occur over WiFi. Other strategies do not support this control and, hence, the update can be downloaded via the mobile network operator, thus potentially incurring additional costs.

Most strategies do not ensure code quality thresholds. Indeed, only strategy I1 performs code quality analysis to ensure stable app behaviour. Thus, other strategies may lead to poorly implemented features, wasted processing time and memory. Such situations can result in bad user experiences and the misuse of the application given the resource constraints of the device.

Strategies I1, S1, and S2 offer tool support to aid developers when building a new application release. All these tools provide a feature to manage releases and make it easy to uploade new releases to the server, and make them available for download. The developer must change the version number of the application in strategy I1, which also allows automatic code signing before upload. In contrast, the tool used by strategies S1 and S2 changes the version number automatically during the upload process. These tools also calculate the hash of the files in the current version, enabling mobile devices to download only the altered files, and hence decreasing bandwidth. I1 and S1 also support collaboration within a team of developers via *developer profiles*. The application owner is responsible for adding and removing collaborators and may also identify who submits particular code updates. Strategy S2 does not offer this feature and only the owner may upload releases on the server. All tools provide a production and test environment. The latter enables the developer to simulate real world situations.

VI. Evaluation and Reflection

Especially in the case of Cordova applications, man-in-the-middle attacks on code transfers impose severe risks for the security of the end user. Attackers can, for example, insert malicious code in JavaScript files while in transit. The impact of such a MiTM attack strongly depends on the capabilities and privileges of the application. For example, many Cordova applications include several plugins to provide desired functionality such as access to the GPS, contacts, pictures and the camera. Malicious code, injected by attackers, also has access to the API's of these plugins, and can use them to steal personal information. In order to demonstrate the mechanisms and the risks of a MiTM attack, the next paragraph describes an example setup of such attack. It demonstrates a straightforward, realistic setup for executing MiTM attacks on regular HTTP connections.

The setup is displayed in Figure 3 and consists of two workstations, connected to the same network, and a mobile device.

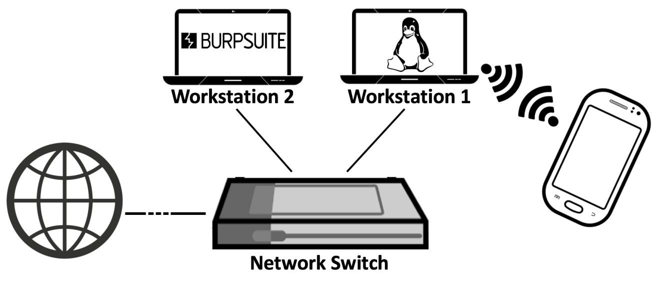


Figure 3: MiTM demonstration setup.

- **Workstation 1** contains a classic Linux distribution. This workstation is set up to create its own wireless network, thereby acting as a public wireless access point. All incoming wireless regular communication (wlan0) is redirected to workstation 2.
- **Workstation 2** runs a proxy tool. BurpSuite [31] was used in our setup. It listens for the traffic forwarded by workstation 1. An attacker can now read and alter all requests and responses.
- The **mobile device** is connected to the Internet via the wireless network provided by Workstation 1 and contains the application under attack. All HTTP traffic on the mobile device is now routed transparently through both workstations.

This attack describes the situation when a user accesses a public access point. Traffic can be eavesdropped and altered without the user noticing. For the sake of clarity, workstations 1 and 2 are displayed as two separate entities. Note that the functionality of both workstations can be combined into a single one. MiTM attacks on unsecure SSL connections (e.g. certificate checks are lacking) work similarly to the demonstrated setup, but require an SSL stripping step in the proxy (Workstation 2).

Our assessment mainly focused on the Android operating system. Nevertheless, most ideas presented in the paper also apply for iOS. In iOS, it usually takes several days before an application is approved and available the app store, hereby making hot code updates even more valuable than in Android. On the other hand, iOS developers have strict limitations for what a hot code update can change in an application [32]. The features and functionality have to be consistent with the intended and advertised purpose of the application as submitted to the App Store. Developers who do not comply with these rules can lose their access to the App Store.Not all Cordova plugins described in this work are available for iOS. Naturally, *Cordova-plugin-app-update* (I2) provides no iOS implementation because it relies on downloading and installing an APK file. Also, *cordova-plugin-dynamic-update* (I4) does not provide an iOS implementation. However, the security risks discussed also apply for iOS. For example, if developers do not secure the communication with the update server, the attack described above can also be executed on iOS applications.

Independently of the targeted operating system, developers have to be aware that plugins might contain unsafe behavior and ignore the best development practices for the platform. Hence, the source code should always be checked before including it in applications. In Android, examples of this are introducing too many permissions and overriding the security mechanisms of the platform, as described in Section V. iOS implementations of plugins can contain similar insecure behavior. For example, the *cordova-plugin-meteor-webapp* plugin disables Apple App Transport Security, introduced in iOS 9 [33].

VII. Conclusion

This paper presented an analysis of hot code updates in Cordova, and compared them against traditional updates through the app store and the loading strategy. Two sets of parameters were discussed in this work. The first group of parameters focused on the security aspect of the updating mechanisms. One of the major concerns when using hot code updates is the secure storage of the update and transition of the update to the device. Some plugins are developed for one specific type of server (e.g. Amazon, Microsoft Azure). In this case, the developer can rely on these mature platforms for securely hosting the update (secure storage on the server and only use HTTPS for communication), and has access to additional tools for managing the update. Other plugins do not offer this complete solution. Hence, the developer is responsible for implementing a secure server to host the update. Besides the server, the plugin is also responsible for secure communication. Developers should check the implementation before installing it in an application. Plugin developers should rely on the underlying platform's APIs as much as possible in order to ensure secure communication.

The second group of parameters focused on the functionality and the usability. The conclusion here is that, in many cases, hot code updates offer a viable solution for minor updates and bug fixes to an application. Based on the needs of the developer and the application, different plugins can be selected. Some plugins offer a simple but limited API, and are therefore quick and easy to integrate in applications. The disadvantage is that these plugins do not offer support for user interaction such as asking permission from the user and giving the user a visual overview of the progress. Other plugins offer a more advanced API to the developer and allow more customization. Hence, these plugins are also more complex to implement for the developer.

REFERENCES

[1] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in Proceedings of the 10th international conference on Mobile systems, applications, and services. ACM, 2012, pp. 113–126.

[2] AppBrain. Number of android applications. [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps [retrieved: October, 2016]

[3] C. Reese Bomhold, "Educational use of smart phone technology: A survey of mobile phone application use by undergraduate university students," Program, vol. 47, no. 4, 2013, pp. 424–436.

[4] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. P. Markatos, and T. Karagiannis, "Rise of the planet of the apps: A systematic study of the mobile app ecosystem," in Proceedings of the 2013 conference on Internet measurement conference. ACM, 2013, pp. 277–290.

[5] S. Amatya and A. Kurti, "Cross-platform mobile development: challenges and opportunities," in ICT Innovations 2013. Springer, 2014, pp. 219–229.

[6] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, Evaluating Cross-Platform Development Approaches for Mobile Applications. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 120 – 138.

[7] P. R. de Andrade, A. B. Albuquerque, O. F. Frota, R. V. Silveira, and F. A. da Silva, "Cross platform app: a comparative study," arXiv preprint arXiv:1503.03511, 2015.

[8] B. R. Mahesh, M. B. Kumar, R. Manoharan, M. Somasundaram, and S. Karthikeyan, "Portability of mobile applications using phonegap: A case study," in Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), International Conference on. IET, 2012, pp. 1–6.

[9] K. Leuven. Crossmos. [Online]. Available: https://www.msec.be/crossmos/ [retrieved: June, 2017]

[10] M. Latif, Y. Lakhrissi, E. H. Nfaoui, and N. Es-Sbai, "Cross platform approach for mobile application development: A survey," in 2016 International Conference on Information Technology for Organizations Development (IT4OD). IEEE, 2016, pp. 1–5.

[11] C. Rieger and T. A. Majchrzak, "Weighted evaluation framework for cross-platform app development approaches," in EuroSymposium on Systems Analysis and Design. Springer, 2016, pp. 18–39.

[12] M. Willocx, J. Vossaert, and V. Naessens, "Comparing performance parameters of mobile app development strategies," in Proceedings of the International Workshop on Mobile Software Engineering and Systems. ACM, 2016, pp. 38–47.

[13] A. Pazirandeh and E. Vorobyeva, "Evaluation of cross-platform tools for mobile development," 2015.

[14] G. S. Kearns, "Countering mobile device threats: A mobile device security model," Journal of Forensic & Investigative Accounting, vol. 8, no. 1, 2016.

[15] P. De Ryck, L. Desmet, F. Piessens, and M. Johns, Primer on client-side web security. Springer, 2014.

[16] S. Vashisht, S. Gupta, D. Singh, and A. Mudgal, "Emerging threats in mobile communication system," in Innovation and Challenges in Cyber Security (ICICCS-INBUSH), 2016 International Conference on. IEEE, 2016, pp. 41–44.

[17] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," IEEE communications surveys & tutorials, vol. 15, no. 1, 2013, pp. 446–471.

[18] F. Peijnenburg, "Security in android apps," 2013.

[19] A. Bhardwaj, K. Pandey, and R. Chopra, "Android and ios security-an analysis and comparison report," Int'l J. Info. Sec. & Cybercrime, vol. 5, 2016, p. 30.

[20] A. Cordova. Plugin search - apache cordova. [Online]. Available: http://cordova.apache.org/plugins/ [retrieved: June, 2017]

[21] Google. Google play. [Online]. Available: https://play.google.com [retrieved: June, 2017]

[22] Microsoft. Codepush. [Online]. Available: http://microsoft.github.io/code-push/ [retrieved: June, 2017]

[23] Github. Github - nordnet/cordova-hot-code-push-cli. [Online]. Available: https://github.com/nordnet/cordova-hot-code-push-cli [retrieved: July, 2016]

[24] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," IEEE software, vol. 13, no. 1, 1996, pp. 12–21.

[25] S. Perez. App submissions on google play now reviewed by staff, will include age-based ratings. TechCrunch. [retrieved: March, 2015]

[26] A. Morris. Improvements for smaller app downloads on google play. Android Developers Blog. [retrieved: July, 2016]

[27] Google. Httpurlconnection - android developers. [Online]. Available: https://developer.android.com/reference/java/net/HttpURLConnection.html [retrieved: June, 2017]

[28] ——. Security with https and ssl - android developers. [Online]. Available: https://developer.android.com/training/articles/security-ssl.html#HttpsExample [retrieved: June, 2017]

[29] ——. Hostnameverifier - android developers. [Online]. Available: https://developer.android.com/reference/javax/net/ssl/HostnameVerifier.html [retrieved: June, 2017]

[30] T. N. Web. Google describes how android 4.2's app verification checks your downloads for malware. [retrieved: November, 2012]

[31] PortSwigger. Download burp suite - portswigger. [Online]. Available: https://portswigger.net/burp/download.html [retrieved: June, 2017]

[32] Apple. Apple developer program information. [Online]. Available: https://developer.apple.com/programs/information/Apple_Developer_Program_Information_8_12_15.pdf [retrieved: December, 2015]

[33] ——. ios 9.0. [Online]. Available: https://developer.apple.com/library/content/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html [retrieved: June, 2017]

A General System for Self Collecting Individual Data - Application to Medical Data

Michel Schneider[1], Suan Tay[1], Chloé Gay[2], Marinette Bouet[1], Emmanuel Coudeyre[2]

michel.schneider@isima.fr, suan.tay@gmail.com, marinette.bouet@uca.fr, {cgay, ecoudeyre}@chu-clermontferrand.fr

[1]LIMOS, Université Clermont Auvergne, France
[2]Service MPR, CHU, Université Clermont Auvergne, France

***Abstract*-In this paper, we propose a system to allow the self-collection of individual data through digital questionnaires and sensors. The self-collecting person uses a tablet or a smart phone and wears a watch that contains the sensors. The main feature of this system is to memorize in a uniform way the answers to the questionnaires and the values of the sensed parameters in order to facilitate their joint analysis. The system has been developed and tested to monitor osteoarthritis patients. It represents an essential element of the control loop: elaboration of recommendations – monitoring the execution of these recommendations - evaluation and readjustment of these recommendations. This system has been implemented in a generic form and can be used to monitor any patient at home or on the move outside. We explain, also, the extensions we are currently making to obtain a general and flexible system.**

Keywords-self collection; questionnaire; sensor; mobile device.

I. INTRODUCTION

The initial motivation for this work resulted from the need expressed by the medical profession to have an easy-to-use system for the self-collection of health data. This involves checking that the patient is following the recommendations made, measuring the results of these recommendations, and then analyzing the results to readjust the recommendations. For example, in the case of the osteoarthritic patients we studied, it is a question of collecting the information relating to physical activity, difficulties encountered in performing certain movements, taking medicines, etc.. Some information can only be collected by questionnaire, but other pieces of information can be conveniently collected via sensors (e.g., the number of steps performed in a day). There is therefore a real interest in associating the self-collection of personal data by digital questionnaires and by sensors.

Self-collection by digital questionnaires has long been considered in all areas. Various systems have been suggested to create questionnaires and enable online responses. The best known are Google Forms and Lime Survey [1]. Questionnaire collection raises problems of relevance that are discussed in [2]|3][5]. In the medical field, questionnaires have been validated to collect various pieces of information about a state of health (see, for example, [4] for evaluating personality, [6] for evaluating level of anxiety), or a practice (see, for example, [7] for evaluating physical activity).

Numerous studies have focused on the collection of medical data by sensors. There is a wide variety of systems and sensors [8][9]. One of the challenges is to capture in a reliable and precise way physical activities [10][11] because they are an accompaniment to many therapies. Our goal was to develop a self-gathering system that combines the two modes of collection (questionnaires and sensors), and to store data collected in a uniform manner in a warehouse so that they can be manipulated jointly. It is thus possible to carry out analysis combining the two types of data in order to search for correlations or to compute indicators which, will serve to improve the recommendations for the patients. For example, for osteoarthritis patients, we can search correlations between the number of steps per day (data coming from sensors) and pain level or difficulty to make some movements (data coming from questionnaires).

Systems combining questionnaires and sensors have already been proposed, but for specific purposes [12][13][14]. The work of [12] uses questionnaires to collect the values of situational variables and wireless sensors to collect cardiac activity and physical activity. But, the two types of data are stored and processed separately. In [13], monitoring physical activity using wireless sensors is experienced and discussed. Drawbacks are highlighted and it is suggested to combine sensors and questionnaires. The work of [14] studies the effects of different treatments on the quality of life for adults with diabetes. This study is based on data coming from sensors and others coming from questionnaires. But, the two types of data are not integrated in a same system. To our knowledge, there is no proposition for an integrated system able to deal with the two types of data. The main advantages of our system are the following: uniform and integrated treatment of data coming from questionnaires and that coming from sensors, full mobility of the patient, management of the system by the medical staff itself without the intervention of a specialist, direct interoperability with analysis tools. Moreover, the system is able to operate in different contexts, including medical and non medical domains.

We explain the functioning of our system through Sections II-VI, and then we present, in Section VII, the extensions that we are currently carrying out to obtain a general and flexible system.

II. OVERVIEW ON THE SYSTEM

1) General specifications. The proposed system promotes the collection of health data by the patient himself. The location of the patient at home or on the move is irrelevant. The only constraint is that the patient has to establish an Internet connection at regular intervals (for example, every evening). Two modes of collection are possible: on one hand, a collection by digital questionnaires via smart phones or tablets, on the other hand, automatic collection by using sensors embedded on smart phones or connected watches. All data is transmitted to a central server in a tabular format (compatible with Excel) for storage and subsequent analysis by a software tool, such as SAS (Statistical Analysis Software) [18] or R [19]. It must be possible to carry out analysis relating to a patient or a group of patients.

The creation of the survey questionnaires is carried out by members of the medical staff. A user-friendly interface is therefore available to perform this task. It is important that this interface can offer a good variety of question types.

Each patient responds to one or more survey questionnaires via the tablet or smart phone according to a pre-established timeline. Notifications are generated by the system as soon as a questionnaire is open. Physical activity (number of steps per hour or per day) is captured via an accelerometer installed on the watch. The advantage of the automatic capture of the activity results from the observation that the survey questionnaires do not allow a reliable collection. Experiments have shown that patients systematically overestimate their physical activity. Tracked data is transmitted from the tablet or smart phone to the central server via a certificate-based protocol using a patient specific identifier. The exchanges always take place on the initiative of the mobile devices.

It is the members of the medical staff who manage the patients (and in particular the assignment of an identifier to each patient) via a specific module.

The system must be simple to use so that its acceptability by the users (members of the medical staff on the one hand, patients on the other hand) does not pose any problem.

Mobile devices may be provided by the patient or by the medical service. The patient provides the Internet access device.

2) Architecture. The chosen architecture is simple (Figure 1). It is based on a central server that hosts the data and the questionnaire and the main computer application (called the server application thereafter). This server is installed in a protected intranet. Each patient has a tablet (possibly associated with a watch, or a bracelet, or other sensors) that can exchange data with the server via a secure Internet protocol. It is the mobile application installed on the tablet that initiates all exchanges with the server.

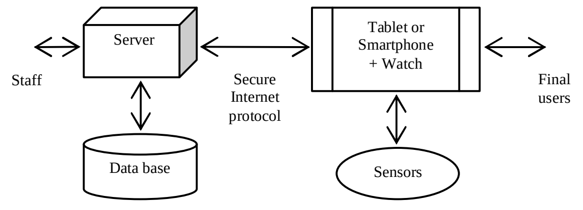


Figure 1. Architecture of the system

3) Main operations. The main operations permitted by the system take place chronologically as indicated below.

① Creation of questionnaires (actor: medical staff in intranet).

② Creation of patients (actor: medical staff in intranet).

③ Assignment of questionnaires to patients (actor: medical staff in intranet).

④ Initialization of the mobile devices (actor: medical staff in intranet): The mobile application is installed on the patient's tablet and watch. The server connection information is initialized on the tablet. The questionnaires are transferred to the tablet.

⑤ Initialization of the tablet connection to the Internet network (actor: patient).

⑥ Response to questionnaires and possible activation of the watch (actor: patient): The data is stored temporarily on the watch and the tablet.

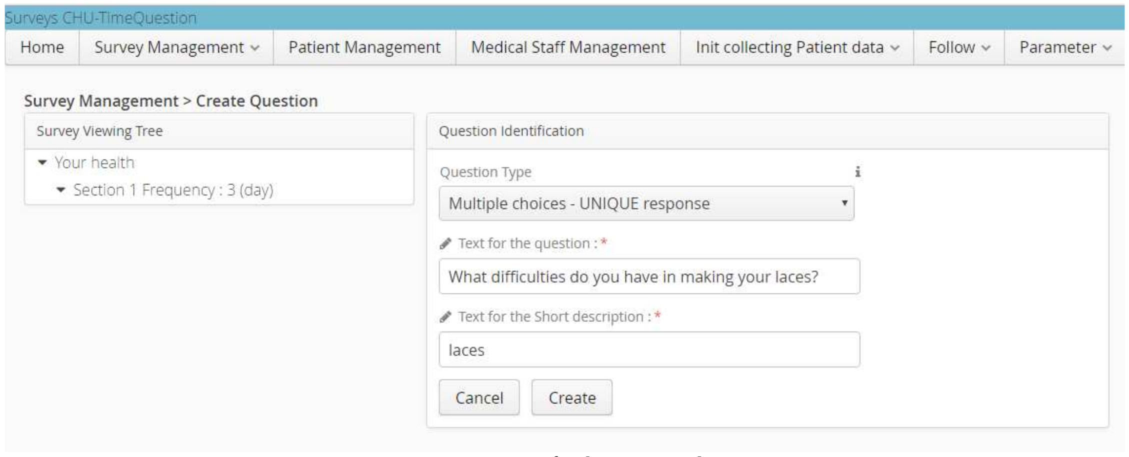
⑦Transmission of data to the server (actor: mobile application in the Internet): The data is transmitted to the central server as soon as the Internet connection is established. Connection and transmission are fully automatic. It is the mobile application that drives the exchanges.

4) Technologies. For the server, we chose a WINDOWS technology associated with MYSQL [15] to manage data storage. The server application is encoded in Java. For mobile devices (tablet or smart phone, watch), we chose an Android technology [16] associated with SQLite [17]. The advantages of Android are two-fold: great variety of mobile devices supported by this system, affordable prices. The mobile application is also encoded in Java. The tablet and watch are interconnected in Bluetooth mode. Data exchanges are carried out by Web services using REST (Representational State Transfer) technology [20].

III. COMPUTER APPLICATION FOR THE SERVER

1) General interface. The general menu (Figure 2) contains tabs for managing medical staff, managing patients, managing survey questionnaires, assigning questionnaires to patients, initializing collection, reporting about a patient or a survey.

2) Model for questionnaires. As soon as a questionnaire is created, a name is assigned to it (this name is used to locate it on the tablet's home page). A questionnaire can be divided into sections. A section may be submitted to the user several times in the form of a series of predetermined deadlines or in the form of a regular repetition over time. A section can include different types of questions: multiple choice questions, cursor questions, grid questions, open-ended questions. These types correspond to those which are most frequently encountered in the medical field. Other types can be added with the same specification approach. The wording of a question is handled in two formats: a long format that is the full text of the question as it is displayed on the tablet screen, and a short format that is used to locate the question in the result table.



Survey Management > Create Question

Survey Viewing Tree
- Your health
  - Section 1 Frequency : 3 (day)

Question Identification

Question Type: Multiple choices - UNIQUE response

Text for the question :* What difficulties do you have in making your laces?

Text for the Short description :* laces

Figure 2. Main menu for the server application

Once the creation of the questionnaire is validated, it switches to the "finalized" state. This status means that the survey is ready to be published to patients.

3) Assignment of questionnaires to patients. Any finalized questionnaire may be associated with patients. The "Assignment Survey" tab allows to associate surveys with patients. It is possible to specify several associations simultaneously.

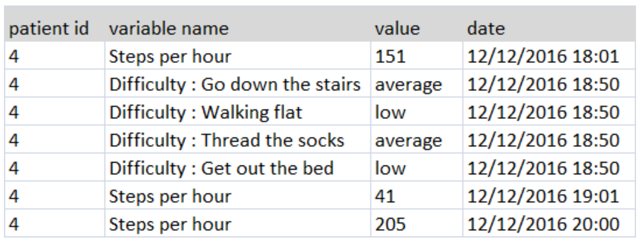
4) Initialization of the mobiles devices of a patient. The initialization of the mobile devices is obtained by scanning a QR (Quick Response) code generated by the server application. This QR code consists of the patient id, the initialization date, the web service address on the server that must be used for the data exchanges.

5) Results management. We separated the results of the surveys and the results of the sensors in two different classes: *result* and *sensorResul*t. The typing of the collected data is the same for the two classes and respects the following format:

(patient id, variable name, collected value, date)

The variable name corresponds to the short label of a question or to the label of the parameter collected by a sensor.

6) Reporting about a survey or a patient. The results for a survey (all patients combined) or the results for a patient (answers to questions and parameter values coming from sensors) can be downloaded in a same Excel file. An example is given in Figure 3. From this file, we can then make ad-hoc reporting or in-depth analysis by using a tool such as SAS or R. It is interesting also to note that such a file can be seen as the fact table of a warehouse with three main dimensions : patients, variable, time. Approaches proposed for calculating indicators in data warehouses can thus be usefully exploited.



patient id	variable name	value	date
4	Steps per hour	151	12/12/2016 18:01
4	Difficulty : Go down the stairs	average	12/12/2016 18:50
4	Difficulty : Walking flat	low	12/12/2016 18:50
4	Difficulty : Thread the socks	average	12/12/2016 18:50
4	Difficulty : Get out the bed	low	12/12/2016 18:50
4	Steps per hour	41	12/12/2016 19:01
4	Steps per hour	205	12/12/2016 20:00

Figure 3. Reporting about a patient (excerpt)

IV. COMPUTER APPLICATION FOR THE MOBILE DEVICES

1) Answering a questionnaire. The names of the different surveys associated with the patient are displayed in different bannners on the tablet home page (Figure 4). These names are those that were specified when creating questionnaires. Banners with a gray background correspond to surveys that are not due at the time of the consultation and are therefore inaccessible. The patient can answer any survey accessible by clicking on the corresponding banner. The accessibility of a survey is determined in accordance with the frequency or timelines specified at the time of its creation.

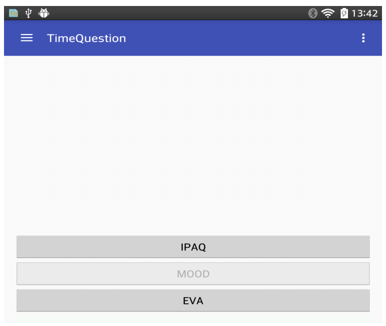
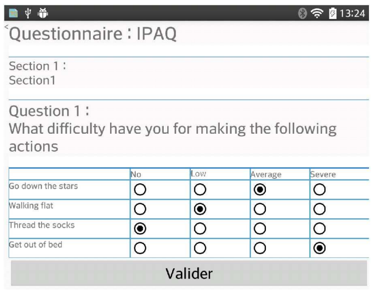


Figure 4. Accessing surveys from the tablet home page



Questionnaire : IPAQ

Section 1 :
Section1

Question 1 :
What difficulty have you for making the following actions

	No	Low	Average	Severe
Go down the stars	○	○	●	○
Walking flat	○	●	○	○
Thread the socks	●	○	○	○
Get out of bed	○	○	○	●

Valider

Figure 5. Answering a question on the tablet (grid question)

Figure 5 shows the presentation of a grid question on the tablet screen.

2) Step counting. The accelerometer of the watch and the Google Step component are used to count the steps. A higher layer was developed to aggregate the count over each 60 minutes interval. It is the value of this aggregate that is transmitted to the server via the tablet.

Other medical sensors are embedded on Android supports and can be installed on our system. They permit to capture parameters such as Temperature, Blood Pressure, Pulse, and Heart Rate.

3) *Data exchanges*. The sending and receiving of data between the watch and the tablet, on one hand, between the tablet and the server, on the other hand, is done automatically without user action. A short message is displayed at the bottom of the page for a few seconds to signal the shipment. The sequence diagram below (Figure 6) describes the principle of exchanges between the mobile devices and the server. The data of the watch is first stored in its internal memory and then transmitted on its initiative to the tablet. The data generated at the tablet level (ie the answers to the questions) and those recovered from the watch are stored in its internal memory and then transmitted to the server at its own initiative. Initialization of the exchange by the transmitter (watch or tablet) occurs every 30 minutes if the network is available (Bluetooth for communication to tablet, Internet for communication to server). The transmitter keeps the data until the receiver has returned an acknowledgment of receipt. If this acknowledgment fails within 30 minutes, the sender attempts a new sending. When the acknowledgment is received, the sender removes the data from its internal base.

4) *Deployment on the playstore*. Our mobile application, called TimeQuestion, was deployed on the playstore to simplify the propagation of updates and initialization. TimeQuestion includes the codes to be installed on the tablet and the watch. The installation of TimeQuestion is done automatically on the tablet and also on the watch, if a watch is connected. The application is optimized for a 7-inch tablet, but it can also be installed on a smart phone.

V. EXPERIMENTS

The system is now fully operational and the server is permanently active. Various experiments have been carried out with members of the Physical Medicine and Rehabilitation Department of the Clermont-Ferrand Hospital (France) and patients of this department in order to evaluate the acceptability of the system.

First, multiple demonstrations were carried out by the authors. Once the mobile devices have been initialized, it is no longer necessary to worry about them. If they become inactive following a discharge of their batteries, simply recharge the batteries and restart them.

Several patients were asked to test the system when they were in the PMR (Physical Medicine and Rehabilitation) Department.

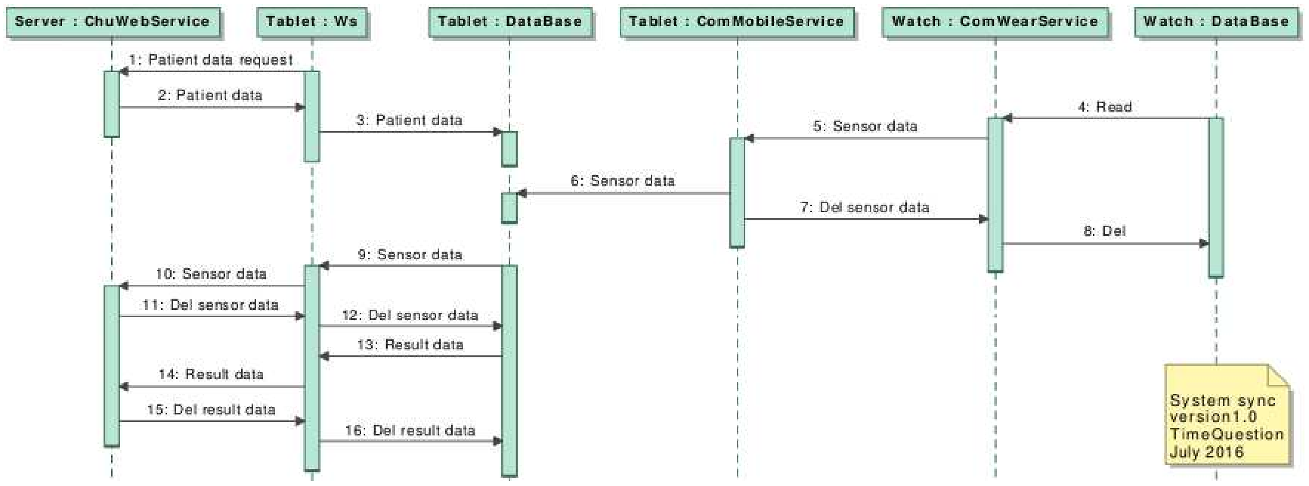


Figure 6. Data exchanges between the mobile devices and the server

One patient was asked to use the system for two weeks. The PMR (Physical Medicine and Rehabilitation) Department has put at her disposal a tablet of 7 inches as well as an android watch. A survey with 16 questions relating to movement difficulties was assigned to her. A response was requested every two days. This patient was asked to walk 30 to 60 minutes a day. We were able so to verify that the data went back to the server on a regular basis. Figure 3 illustrates an excerpt of the collected data during this experiment.

All these actors found that the system was very convenient to use and very useful.

We did not have the opportunity to address the acceptability of the system by patients with disabilities, such as those with hand tremors or those with visual deficiencies. We are confident about the efficiency of our system for those with hand tremors because tablets with large screen sizes can be handled by the system. We have not explored solutions for patients with visual deficiencies. For this type of handicap, specific solutions need to be studied.

The MPR department studies the effects of hydrotherapy for its osteoarthritis patients. It planned to use the system with about ten patients for the next treatment periods.

VI. Towards a more general and flexible system

We explain in this section the extensions we have undertaken to make the system more general and more flexible.

1) *Separate management of questions*. In the current version, each question is linked to a questionnaire. Experiments have shown that the same question may appear in different questionnaires. It is therefore a question of being able to specify the questions separately, then to assemble them to form the questionnaires.

2) *Semantic standardization of labels*. In the current version, the wording of the questions and parameters captured is left to the free choice of the medical staff. It is then difficult to integrate the data coming from different services (remember that these labels serve as semantic reference for the collected data). Yet, this integration would be interesting to deal with more massive data coming from different horizons. To facilitate this integration, we propose to constrain the choice of labels through a shared ontology of the domain. We are adapting the specification of long and short labels in order to impose the choice of a term in an ontology.

3) *Sensor assisted installation*. In the current version, each sensor is associated with a specific software component which, collects the raw data transmitted, ensures their filtering and aggregates this data over the relevant period before transmitting them to the tablet. In addition, the label of the sensed parameter is hard-coded in the component. The extension consists of decomposing this component into two parts: a part that remains specific to each sensor (this part collects the raw data and performs the filtering), a part that ensures the naming of the associated parameter and the calculation of the aggregate. The main menu of the server application is redesigned to allow the choice of the sensor, the choice of the associated label, the choice of the type of aggregation to be performed (sum, average, etc.), the aggregation time interval. The second part of the component can be then automatically generated. The installation of a sensor can be so specified directly by a user manager without requiring the intervention of a developer.

VII. Conclusion

The initial objective of this work was to design and develop a system for the self-collection of medical data. The data come from the responses to questionnaires transmitted via the tablet, on the one hand, and the parameters collected by the watch via sensors, on the other hand. The questionnaires are defined by the members of the medical staff through a convivial interface and their structures are stored on the server in a relational database. They are then loaded onto the tablet during an initialization procedure. The application allowing the reading of the values of the sensors is also automatically installed on the watch during this initialization. All data collected from questionnaires and sensors are stored in a unified tabular format to facilitate their recovery by a spreadsheet in order to activate various statistical analyzes or data mining treatments.

Our system has also other main advantages. It allows full mobility of the patient. Its management can be handled by a member of the medical staff without the intervention of a specialist. Its operation is automatic as soon as the initialization of the mobile devices has been carried out.

We have conducted experiments which have shown that the system is well accepted by the patients.

It is interesting to note that it is possible to incorporate other types of data into our warehouse. For example, in the medical domain, the fact table could be used to store medical analysis results or imaging reports. Technically, we need to study the interconnection of our system with the other systems used by physicians.

The data collected, in particular via the sensors, can quickly become bulky, and one can wonder about the suitability of such a system for handling big data. The main problem is the server's storage capacity. A relational table under Windows NTFS (New Technology File System) has a maximum capacity of 2GB. We can evaluate the length of a line in the *result* table or the *sensorResult* table to 200 Bytes. Suppose the server is used by a medical department to track 100 patients. It is thus possible to store 200,000 answers to questions for a single patient and to store the number of steps per 60 minutes over 25 years for a patient. It is very comfortable. But if we want to integrate data coming from several departments into hospitals across the country, this capacity may become insufficient and other storage technologies should be considered. Today, there are technologies for big data that remain compatible with our architectural choices and that do not put into question our software.

This system was initially defined on the basis of wishes expressed by a hospital department. But it has been designed and developed in a generic way and can be used to collect any kind of data from an individual or a natural or artificial entity. First, it can be used for monitoring individuals in various situations: athletes in training, workers in the performance of certain tasks, etc. But, it can also be used for monitoring any type of non human entity. The sensors are then installed on the entity and the questionnaires are activated by a human observer of this entity. For example, sensors can be used to monitor vegetal growth and collect immediate environmental conditions (e.g., moisture and temperature for air and soil). Questionnaires can collect more environmental information (e.g., nature and evolution of the surrounding plantations), useful for explaining the vegetal growth.

This system is currently being extended to make it more general and flexible. These extensions mainly concern three directions: the separate specification of the questions, the semantic standardization of the labels identifying the collected data, the assisted installation of a sensor.

REFERENCES

[1] C. Schmitz, “Lime Survey: the free and open source survey software tool”, Available on: http://www. limesurvey. org/, Accessed on June, 12, 2017.

[2] D. H. Granello and J. E. Wheaton, “Online data collection: Strategies for research,” Journal of Counseling & Development, no 82(4), pp. 387-393, 2004.

[3] K. B. Wright, “Researching Internet-based populations: Advantages and disadvantages of online survey research, online questionnaire authoring software packages, and web survey services,” Journal of Computer-Mediated Communication, no 10(3), 2005.

[4] A.Tellegen and N. G. Waller, “Exploring personality through test construction: Development of the Multidimensional Personality Questionnaire,” The SAGE handbook of personality theory and assessment, no 2, pp. 261-292, 2008.

[5] D. Y. Leung and D. Kember, “Comparability of data gathered from evaluation questionnaires on paper and through the Internet,” Research in Higher Education, no 46(5), pp. 571-591, 2005.

[6] M. N. van Poppel, M. J. Chinapaw, L. B Mokkink, W. Van Mechelen and C. B. Terwee, “Physical activity questionnaires for adults. Sports medicine,” no 40(7), pp. 565-600, 2010.

[7] K. Kroenke, R. L. Spitzer, J. B., Williams and B. Löwe, “The patient health questionnaire somatic, anxiety, and depressive symptom scales: a systematic review,” General hospital psychiatry, no 32(4), pp. 345-359, 2010.

[8] A. Pantelopoulos and N. G. Bourbakis, “A survey on wearable sensor-based systems for health monitoring and prognosis,” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), no 40(1), pp. 1-12, 2010.

[9] H. Alemdar and C. Ersoy, “Wireless sensor networks for healthcare: A survey,” Computer Networks, no 54(15), pp. 2688-2710, 2010.

[10] J. Parkka, M. Ermes, P. Korpipaa, J. Mantyjarvi, J. Peltola and I. Korhonen, “Activity classification using realistic data from wearable sensors,” IEEE Transactions on information technology in biomedicine, no 10(1), pp. 119-128, 2006.

[11] O. D. Lara and M. A. Labrador, “A survey on human activity recognition using wearable sensors,” IEEE Communications Surveys and Tutorials, no 15(3), pp. 1192-1209, 2013.

[12] A. Gaggioli, G. Pioggia, G. Tartarisco, G. Baldus, D. Corda, P. Cipresso and G. Riva, “A mobile data collection platform for mental health research,” Personal and Ubiquitous Computing, no 17(2), pp. 241-251, 2013.

[13] D.R. Bassett Jr, “Validity and reliability issues in objective monitoring of physical activity,” Research quarterly for exercise and sport, no 71, pp. 30-36, 2000.

[14] R. Rubin and M. Peyrot, “Treatment satisfaction and quality of life for an integrated continuous glucose monitoring/insulin pump system compared to self-monitoring plus an insulin pump,” Journal of diabetes science and technology, no 3(6), pp. 1402-1410, 2009.

[15] J. Greenspan and B. Bulger, “MySQL/PHP database applications,” John Wiley & Sons, Inc., 2001.

[16] R. R. Lombardo, J. Mednieks and B. Meike, “Android application development: Programming with the Google SDK,” O'Reilly Media, Inc., 2009.

[17] M. Owen and G. Allen, “SQLite,” Apress LP, 2010.

[18] R. C. Littell, ”Sas,” John Wiley & Sons, Ltd, 2006.

[19] R, “The R Project for Statistical Computing,” https://www.r-project.org/, Accessed on June, 12, 2017.

[20] C. Riva and M. Laitkorpi., "Designing web-based mobile services with REST," Service-Oriented Computing ICSOC 2007 Workshops, Springer Berlin/Heidelberg, 2009.