TÉCNICO LISBOA

RFIDToys
A Flexible Testbed Framework for RFID Systems

Nuno Miguel Mendes Correia

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Doutor José Manuel da Costa Alves Marques
Eng. Mestre Miguel Filipe Leitão Pardal

Examination Committee

Chairperson: Prof. Doutor Pedro Manuel Moreira Vaz Antunes de Sousa
Supervisor: Prof. Doutor José Manuel da Costa Alves Marques
Member of the Committee: Prof. Doutor Alberto Manuel Ramos da Cunha

May 2014

Abstract

Radio Frequency IDentification (RFID) technology allows the remote identification of tagged objects without line of sight. Unfortunately, there are severe reliability problems due to the air protocol susceptibility to interferences from surrounding objects, causing *false positive* and *false negative* readings. To mitigate this problem, sliding windows, a middleware solution proposed by EPCglobal Inc, is commonly used.

The present work studies and compares, in the Fosstrak platform, the sliding window data cleaning strategy with the innovative SMURF algorithm that considers the stream data from the RFID antennas as a statistical sample of the world. To correctly compare both data cleaning strategies with statistical support, an easy-to-use RFID Testbed Framework is developed and two new metrics are proposed: $PresenteErrors$ and $PresenceRate$. To run an experiment, the testbed involves recording a set of physical repetitions to then, randomly replay the recorded sessions to avoid biased results. All the results are summarized following sampling statistical theory to, within a certain level of confidence, correctly compare the alternative strategies.

The research showed that the SMURF algorithm is capable to self-adapt, without any reconfiguration, to environmental conditions with a reduction of errors. The sliding window algorithm required reconfiguration for each scenario, potentially raising operational costs.

It is proved that the proposed testbed helps to obtain statistical correct answers when evaluating different EPCglobal platforms, paving the way to improve the state of art with statistical confidence on the results. Furthermore, it becomes an excellent choice to demonstrate such technology in both industrial and academic contexts.

Keywords: RFID , EPCglobal , Testbed , Data Cleaning , SMURF , Statistical Theory

Resumo

A tecnologia *Identificação por Radiofrequência* (RFID) permite a identificação remota de objectos sem linha de vista. Infelizmente, existem graves problemas de fiabilidade devido ao protocolo usado que é susceptível às interferências dos objectos envolventes causando leituras falsas positivas e falsas negativas. Uma das formas de resolver este problema é recorrendo a plataformas de middleware como as janelas deslizantes propostas pela EPCglobal Inc.

O presente trabalho estuda e compara, na plataforma Fosstrak, a estratégia de janelas deslizantes com o inovador algoritmo SMURF que considera o fluxo de eventos produzidos pelas antenas RFID como uma amostra estatística do mundo. Para uma comparação correcta, com suporte estatístico, um RFID Testbed Framework foi desenvolvido juntamente com duas novas métricas: $PresenteErrors$ e $PresenceRate$. Para executar uma experiência primeiro são gravadas diversas sessões físicas para depois as reproduzir de forma aleatória e evitar enviesamento dos resultados. Estes são depois sumarizados para que as alternativas sejam, dentro de um certo nível de confiança, correctamente comparadas.

O trabalho mostrou que o algoritmo SMURF é capaz de se auto-adaptar, com redução dos erros, às condições do cenário sem necessidade de reconfiguração. Por outro lado, as janelas deslizantes requerem reconfiguração para cada cenário podendo aumentar os custos operacionais.

É provado que o testbed proposto ajuda a obtenção de respostas estatisticamente correctas quando se evaliam as diferentes plataformas EPCglobal, possibilitanto o avanço do estado de arte. De acrescentar que se mostra uma excelente plataforma para demonstrações da tecnologia RFID tanto em ambiente empresarial como académico.

Keywords: RFID , EPCglobal , Testbed , Limpeza de dados , SMURF , Estatística

Contents

Abstract **i**

1 Introduction **1**

1.1 Objectives . . . 7

1.1.1 Methodology . . . 7

1.1.2 Testbed Framework . . . 7

1.2 Contributions . . . 8

1.3 Document Structure . . . 8

2 Background **9**

2.1 Concepts . . . 9

2.1.1 RFID Technology . . . 9

2.1.2 GS1 EPCglobal Architecture . . . 11

2.2 RFID Platforms . . . 13

2.2.1 Fosstrak . . . 13

2.2.2 Rifidi Edge Server . . . 15

2.2.3 BizTalk RFID . . . 15

2.3 Data Cleaning Strategies . . . 16

2.3.1 Physical Solutions . . . 17

2.3.2 Middleware Solutions . . . 17

2.3.3 Deferred Solutions . . . 21

2.4 Summary and Discussion . . . 21

3 RFIDToys Testbed **23**

3.1 Architecture 24

3.1.1 Middleware Platform: Fosstrak 24

3.1.2 The Robot 28

3.1.3 Dashboard 30

3.2 Summary and Discussion 32

4 Methodology **33**

4.1 Experiment Definition 34

4.1.1 Data Cleaning Evaluation Metrics 34

4.2 Record 36

4.3 Replay 38

4.4 Results Analysis 39

4.5 Summary and Discussion 40

5 Experiments Evaluation **41**

5.1 Experiments Definition 41

5.1.1 Experiments Setup 41

5.1.2 Record Phase Configurations 42

5.1.3 Replay Phase Configurations 43

5.1.4 Research Questions 45

5.2 Experiments Performed 46

5.2.1 Baseline 46

5.2.2 Track 1 Lap 47

5.2.3 Track 3 Laps 52

5.2.4 Track 3 Laps with Interferences 56

5.3 Results Interpretation and Discussion 58

6 Conclusion **61**

6.1 Data Cleaning Module: Sliding Window vs SMURF 63

6.2 Contributions Summary 64
6.3 Future Work 64

Bibliography **70**

A Statistically Valid Experiments **71**

A.1 Population Mean Estimation 71
A.2 Number of Repetitions Estimation 74
A.3 Randomization of Sessions 75

B System Metrics **77**

B.1 Process CPU Load 77
B.2 System Load Average 78
B.3 Memory Usage 78

List of Tables

5.1 Best data cleaning configurations results. 58

A.2 Maximum margin of error for different sample sizes. 75

List of Figures

1.1 Completeness and tag dynamics influencing the sliding window size. 4

1.2 Simulation of a conveyor belt. 5

1.3 The robot developed and used in the testbed. 7

2.4 RFID physical components. 10

2.5 RFID middleware between physical components and client's application. 11

2.6 GS1 EPCglobal Architecture Framework. 12

2.7 Fosstrak Architecture. 14

2.8 Rifidi Architecture. 16

2.9 BizTalk RFID Architecture. 17

3.10 Testbed high level architecture. 23

3.11 Testbed architecture. 25

3.12 Rec&Play library class diagram. 26

3.13 Refactoring of Fosstrak FCServer data cleaning module. 27

3.14 Robot components. 29

3.15 Track with segments and read areas. 30

3.16 Dashboard interface: business activity and robot controller. 31

3.17 Robot programming example. 32

4.18 Experiment methodology steps. 33

4.19 Testbed record phase with deactivated components. 36

4.20 Testbed replay phase with deactivated components. 39

5.21 Experiment setup. 43

5.22 Business actions flow implemented into Capture Application rules. 44
5.23 Baseline - System metrics results for the considered stages. 46
5.24 An *EventCycle* trigger that fails to start. 48
5.25 Experiment Track 1 Lap data cleaning results before the fine-tuning. 49
5.26 Experiment Track 1 Lap system metrics results in stage 2. 51
5.27 Experiment Track 1 Lap data cleaning metrics after the fine-tuning. 52
5.28 Experiment Track 3 Lap system metrics results. 53
5.29 Experiment Track 3 Lap data cleaning metrics for the three laps. 55
5.30 Disturbances in the reader covered area by introducing a metallic object. 56
5.31 Experiment Track 3 Lap with Interferences data cleaning results. 57

Chapter 1

Introduction

Radio Frequency IDentification (RFID) technology enables various industries to have more efficient and accurate business processes, in particular supply chains to track and trace products. For instance, in a warehouse, products are received, stored and dispatched. If these transitions were logged, they can be used to deliver useful and rich reports to the stock management department. A feasible solution would be to deploy RFID readers in every point of products passage to know exactly where they are and when they transit between areas. It is also possible to know the time that each product stays in each area, giving a clear vision about the real state of the warehouse and product flows.

Want (2006) and Nikitin *et al.* (2007) talk about RFID technology, particularly in its two flavors: short range also known as *Near Field Communication* (NFC) and long range. Both flavors have different applications with their own challenges. NFC is being used essentially with authentication and authorization features where a proximity between the antenna and the object is needed. The long range is being used in a variety of use cases where identification is the main requirement. In the *Internet of Things* topic, the long range and the no need of line of sight are the most interesting flavors since there is the need to easily identify every physical object. Though, RFID uses an air protocol through radio frequency waves, which are much weaker than commercial radio broadcast signals, consequently it can be blocked or disturbed by common objects, including the human body.

RFID technology can improve and speed up processes as well as the awareness about the products being transacted. Despite the advantages, there are challenges that still needed to be addressed, the major issue is the price of the tags that it is still too high to be used in every package. The same regarding the standardization of the protocols and flows that RFID systems must have. Without this, features like trace and track-ability are not possible in a global scale.

EPCglobal Framework. In the context of RFID being used in the supply chain, EPCglobal Inc was created as a subsidiary of the global standards organization GS1, that promoted the use of barcodes. Now it supports the adoption of the Electronic Product Code (EPC), a universal identifier that provides a unique identity for every physical object. An architecture framework proposed by Traub *et al.* (2010) called *EPCglobal Architecture Framework* was created with the goal of standardizing all the components and interfaces required for the expansion of the technology. The EPCglobal Framework was designed to solve a set of constraints and application requirements identified by Floerkemeier *et al.* (2007).

The most important RFID constraints identified are:

- *Reliability Issues* - RFID technology could face interferences produced by surrounding objects. As a consequence, *false positive* and *false negative* readings may happen. Hence, it is necessary to have a module where the interferences are mitigated.
- *Heterogeneous Reader Landscape* - the number of different type of RFID readers available is large. Moreover, each one has its own capabilities in terms of number of antennas, connections and configurations. So, the middleware must have an abstraction layer where the inner details of a specific reader are abstracted from the rest of the application.

Upon these constraints, a set of application requirements was also identified by Floerkemeier *et al.* (2007). They are the EPCglobal vision's pillars for a standardized RFID middleware platform:

- *RFID Data Dissemination*, the information collected by the readers is useful not only to a single application but also to a broader set;
- *RFID Data Aggregation*, the readers can generate a lot of information that can be grouped into different domains such as time or space;
- *RFID Data Filtering*, there are some applications only interested in a subset of tag information and therefore the bulk information coming from the readers must be filtered; and
- *Sharing of RFID Triggered Business Events*, the platform must be able to trigger business events as they happen.

The present research focuses in the *Filtering & Collection Server* (FCServer) module where the main RFID constraints are addressed and where the *Data Dissemination*, *Data Aggregation* and *Data Filtering* are implemented. As the entrance point for the antenna's events, this module is the responsible to clean the noisy stream that came from the antennas to eliminate of dissipate

as much erroneous reads as possible. Tu & Piramuthu (2011), Fritz *et al.* (2010), Darcy *et al.* (2009) and Bendavid *et al.* (2008) pointed the huge number of false reads as the major issue that hinders the widespread of RFID technology.

The platform *Free and Open-Source Software for Track and Trace* (Fosstrak) is an open source implementation certified by EPGglobal. Christian Floerkemeier, one of the creators of Fosstrak, mentioned during a workshop in the *IEEE RFID 2009* conference[1] that the platform was downloaded more than $3\ 000$ times and is used in a wide variety of companies around the world (33% of the downloads in Europe and 24% in United States). The project continues to be successful and used in commercial projects, nowadays.

Reliability Issues. Erroneous reads is a problem identified by Floerkemeier *et al.* (2007). It directly affects the performance of the system and its reliability if some events are missed or overseen. These erroneous reads can be categorized into two types: *false positive* readings where the antenna reports the presence of the tag but in reality the tag is not there; and *false negative* readings where the RFID antenna does not report the presence of the tag despite being there. Foster & Burberry (1999), Clarke *et al.* (2006), Nikitin & Rao (2008) and Wu *et al.* (2006) present some aspects that could affect the antenna-tag-antenna read efficiency like the presence of human bodies, water recipients, metallic objects as well as other radio frequency emitters. Plus, the package or the substrate where the tag is can also affect its read efficiency. Generally, water bodies absorb the radio signals and metallic objects reflects them.

Ahmed & Ramachandran (2008) and Bolic *et al.* (2010) state that in a noisy environment the percentage of erroneous reads can be up to 50%, that is, half of the antenna reads could be wrong. To minimize the problem there are three possible options: *a*) hardware solutions placing more readers to cover the same area to avoid miss readings, it is reported that with up to 10 readers installed the false readings can be as low as 0.1%; *b*) middleware solutions where a software layer is introduced between readers and client applications; or *c*) deferred solutions where the data is analyzed later from where the correct events are inferred.

Hardware solutions can be costly since RFID antennas are expensive devices. Consequently, RFID implementation costs raises if multiple readers in each point of control are needed. Jakkhupan *et al.* (2011) pointed out this issue and also software configuration costs as preventing a wider RFID adoption.

Alternatively, opting for a middleware solution, the EPCglobal framework specifies a basic strategy called *Event Cycles* where a period of time is used to register antenna's events and, at

[1] More information: `http://2009.ieee-rfid.org/files/2011/12/3C_-_3_-_FosstrakIEEERFID.pdf`

the end of that period, all tags are reported to the client application. During this period of time, the middleware is accumulating antenna's events and, within some conditions, *false negative* readings can be avoided because the middleware system ignores some of the non reported tag reads. However, this strategy has some problems: *i*) its configuration is static; *ii*) the window of time must be chosen for each scenario and it is hard to find its value; and *iii*) it does not react to interferences. Further, Jeffery *et al.* (2006) state that the task to find the correct period of time, or sliding-window, should take into account two opposite needs: *a*) *completeness* to ensure that all tags within reader vicinity are correctly read and reported; and *b*) *tag dynamics* to ensure that every tag transition is detected and correctly reported.

Completeness and Tag Dynamics. The purpose of the data cleaning module specified by the EPCglobal framework is to eliminate erroneous reads to improve the match between the reality and its image sent to client applications. Naturally, a larger sliding window must be chosen to ensure *completeness* whereas a shorter sliding window must be chosen to ensure *tag dynamics*. The former will depend on the reader's reliability and the latter will depend on the *physical* tag movement. Both are different needs with opposite consequences, thus, for every deployment, the configuration of the sliding window must be carefully considered.

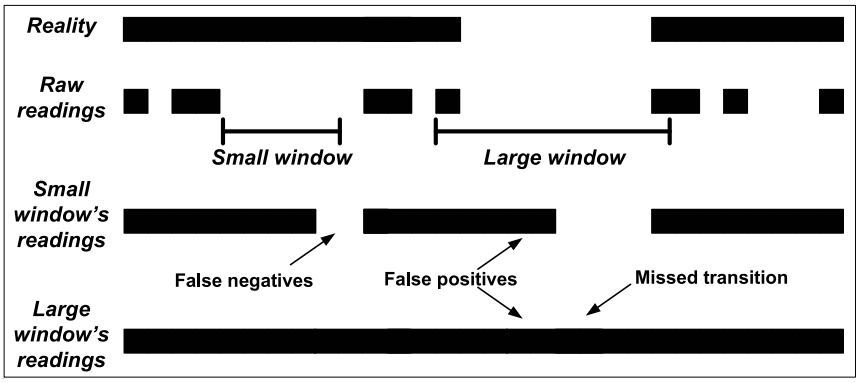


Figure 1.1: Completeness and tag dynamics needs influencing the sliding window size (Jeffery *et al.* (2006)).

The task of choosing the right configuration for the sliding window can be observed in Figure 1.1 where the consequences of both small and large windows are depicted. As shown in the figure, when a large sliding-window is chosen, it can guarantee *completeness* but it also can introduce *false positive* readings and also in some cases misses to report a tag transition. On the other hand, a small sliding window, tries to guarantee *tag dynamics* but it can introduce *false negatives* readings.

SMURF Data Cleaning Algorithm. Jeffery *et al.* (2006) proposed the SMURF algorithm as a way to balance both *completeness* and *tag dynamics* requirements by dynamically adjusting the sliding window size based on tag's parameters. It relies on the premise that the stream of events coming from the RFID antennas are a statistical sample of tags in the physical world rather than assuming that what is reported is the "only" truth.

SMURF has had a significant impact due to its innovative approach on how to deal with both: data noisy stream and tag presence estimation. Works from Zhang *et al.* (2011), Massawe *et al.* (2012a,b), Fan *et al.* (2012) and Xing & Wen-xiu (2013) proposed enhancements to the algorithm where other parameters like speed and direction of the tag were introduced.

It is clear that every platform needs to deal, sooner or later, with the erroneous reads before delivering the set of tag events. SMURF algorithm and its derivations, like any other data cleaning strategy, needs to be tested.

RFID Testbeds. Testbed setups are essential when a researcher wants to test and experiment a new approach, especially when experiments have real-world implications like RFID. This technology suffers from reliability issues due to all the interferences that real scenarios can introduce and so, scenarios must be carefully tested and evaluated. In a testbed scenario two types of RFID data can exist: *a*) *synthetically generated data* where generators can be configured with some inputs from where the expected values can be extracted; or *b*) *real-world data* coming from a real scenario with readers, antennas, tags and interferences that a particular environment could have. The most interesting type of data is the last one, where all the real world constraints and difficulties are not "cleaned" from the evaluation. On the other hand, it is harder to extract such expected values. So, it is necessary to have a testbed where real world data can be collected, analyzed and reproduced to stress and correctly evaluate an RFID system. Also, it should guarantee that it gives the necessary conditions to repeat an experiment in a consistent way.

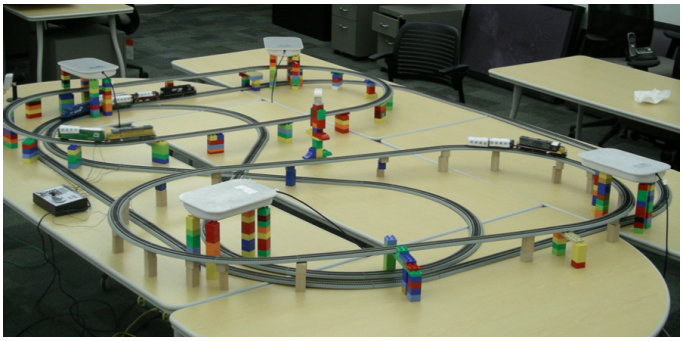


Figure 1.2: Simulation of a conveyor belt (Ahmed & Ramachandran (2008)).

Ahmed & Ramachandran (2008) defined a testbed prototype to test its RFID middleware platform. It was done using a toy train going over a predefined track passing through RFID readers in different locations (Figure 1.2). The purpose was to study the behavior of a group of identifiable items in a conveyor belt at different speeds and destinations. The testbed was used to study the efficiency of the proposed load shedding strategy where *Virtual Readers* and *Virtual Paths* were introduced in order to face two challenges: the growing volume of data in large deployments; and the need to have timely available information.

Despite being a robust testbed, Ahmed & Ramachandran (2008) work lacks of flexibility to run and test multiple scenarios. Additionally, it focuses on evaluating a particular module, it does not take into account the whole system's performance and it does not detail a complete and flexible testbed methodology. Also, there is no evidence of statistic approaches to find the correct value based on a set of experiments. This omission was also observed in other author's work.

In such unpredictable systems, like RFID is, testing all cases with all the possible interferences that can happen in an experiment it is not practical. There is the need to take a sample of possible scenarios and use statistical theory to estimate the correct value. So, a concise and precise methodology is needed to correctly test and evaluate the results.

Present Work. This work studies the precision and accuracy of the data cleaning module in an RFID platform. First, an evaluation assessed the performance of the Fosstrak's implemented strategy. Second, the SMURF algorithm was compared with the static sliding window configurations and both were evaluated by introducing an object in the scenario that caused interferences.

Specific metrics were designed in order to evaluate the whole system performance in terms of *presence errors* and *presence time*. In detail, the metric $PresenceErrors$ depicts the number of transition errors, and the metric $PresenceRate$ corresponds to the ratio between the time the object spent in the reader's covered area, and the system's reported time.

To guarantee that all the results can be correctly analyzed, the scenarios were designed and documented. A robot was used to make sure that all the experiments were repeated under the same conditions without major deviations (Figure 1.3) such as the robot's program, its battery level as well as the experiments conditions (metallic objects, ambient temperature, etc).

With the testbed developed, technical RFID technology demonstrations in an industry context or for pedagogic purposes could be possible, enabling practitioners to learn and test their own scenarios and suggest new improvements.

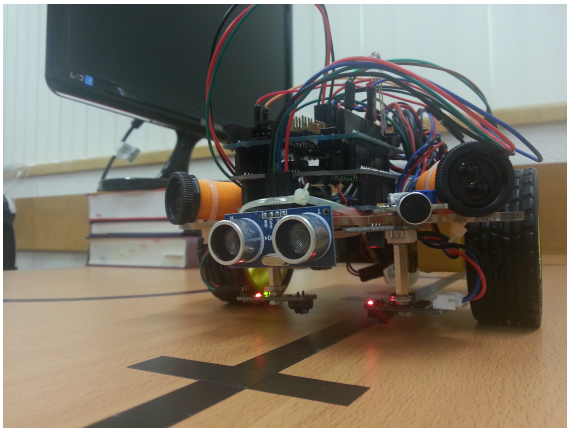


Figure 1.3: The robot developed and used in the testbed.

1.1 Objectives

The aim of this work is to choose the best data cleaning strategy for a set of scenarios. In order to reach meaningful conclusions, a testbed and a methodology are proposed and described, guaranteeing a correct study of the performance of the system regarding the data cleaning module. Moreover, the introduction of disturbances in the antenna's read area can be studied and conclusions can be taken on which data cleaning configuration performs better.

Finally and based on the present work's conclusions, insights are provided on how to improve not only the SMURF algorithm but also data cleaning strategies in general.

1.1.1 Methodology

The methodology used in this work is a vital piece since it will guarantee a correct execution of the experiments to allow a correct statistical analysis of the results.

In detail, the proposed methodology has the following steps: *i*) experiment definition; *ii*) recording of physical experiment repetitions using the same conditions; *iii*) "software" repetitions of the physical recorded samples; and *iv*) results evaluation and conclusions.

1.1.2 Testbed Framework

Along with the methodology, a testbed framework is provided where everyone can extend and improve the existing modules to test other scenarios or evaluating different strategies. Additionally, the framework through the Rec&Play module, produce session files which can be distributed among the community to enable correct comparisons between test runs.

As part of the testbed is the dashboard which controls the robot and receive business events from the system. This way it is possible to see in a near-real time basis what are the impacts that slightly changes in the scenario's configurations can make in terms of business events.

1.2 Contributions

The main contributions of this work are: *i*) a testbed framework to demonstrate and test RFID platforms; *ii*) a methodology to obtain statistical significant results from the experiments; *iii*) metrics on how to evaluate the data cleaning module performance; *iv*) procedure on how to find the best data cleaning configuration for a specific scenario; and *v*) comparison of the static sliding window strategy and the SMURF algorithm.

1.3 Document Structure

The present document is structured as follows:

- **Chapter 2. Background** introduces some key concepts that support the present work such as the EPCglobal framework architecture, analysis of RFID platforms, and analysis of alternative data cleaning modules with an emphasis on the SMURF algorithm.
- **Chapter 3. RFIDToys - Testbed** shows the extensions made to the Fosstrak's architecture in order to deliver a testbed platform, namely: the Rec&Play module, possibility to choose at configuration time the data cleaning algorithm to use, the Robot and the Dashboard.
- **Chapter 4. Experiments Methodology** details all the steps and concerns that should be taken into account to ensure statistical meaningful results.
- **Chapter 5. Experiments Evaluation** describes all the experiments made to meet the defined objectives with an analysis of its results.
- **Chapter 6. Conclusion** where the main conclusions are presented as well as some key research points for future work.
- **Appendix A. Statistically Valid Experiments** where is explained how metrics must be collected in order to apply statistics theory to obtain statistically valid results. Also, is explained how random sessions must be selected to avoid any biased choice.
- **Appendix B. System Metrics** where the system metrics used along the work are detailed.

Chapter 2

Background

This chapter presents the related work relevant to understand the work developed. The first section briefly describes the main RFID technology concepts. It is followed by a description of the most used middleware platforms and by a section where the most relevant Data Cleaning strategies are presented. To end with a set of paragraphs with the summary and conclusion.

2.1 Concepts

2.1.1 RFID Technology

Radio Frequency IDentification (RFID) allows an object to be identified without line of sight between the object and the reader. As stated by Uckelmann *et al.* (2011), it may play an important role in the *Internet of Things* where each object could have interchangeable information.

The next paragraphs describes the essential components of RFID. Detailed information about it can be found in Finkenzeller & Muller (2010), Bolic *et al.* (2010) and Roussos (2008).

RFID Tags An RFID tag is the device used to identify an object, and it has two main parts: *a*) an antenna; and *b*) an *Integrated Circuit* (IC). Traub *et al.* (2010) details four types of tags:

- *Class-1 Passive tags*: the most basic tag available where all the energy needed to activate the IC and answer back is captured by the antenna.
- *Class-2 Higher-function tags*: identical to Class-1 tag but with the possibility to have an extended user memory and authenticated access control.

- *Class-3 Semi-Passive tags*: it has an auxiliary battery to power the IC and other sensors. Despite this, it requires a reader to initiate the communication and it also uses the backscatter technique to send information back to the reader.
- *Class-4 Active tags*: this type of tag have a power source, typically a battery. It has the capability to initiate a communication channel directly with the reader or with another tag.

Figure 2.4(a) shows two RFID tags, each one with different shapes and capabilities. In the tag on the right (*Class-1* or *Class-2* tag), it is possible to see the IC at the center and then around it the antenna. The antenna is used to capture energy from radio frequency waves enabling the IC to be activated. Then the IC, through the same antenna, answers back to the RFID readers through a mechanism called *backscatter*.

(a) RFID tags. (b) RFID reader and antennas.

Figure 2.4: RFID physical components.

Costs reasons explain why passive tags are the most used, however it is not low enough to enable its wider adoption. Additionally and in terms of range, *Class-1* and *Class-2* tags have the lowest range while the *Class-4* (tag on the left in Figure 2.4(a)) tags have an higher range due to its battery support.

Reader + Antennas An RFID reader is a transceiver device capable to send and receive radio frequency waves. The reader through the antenna sends a radio frequency wave with enough energy to activate the tag. Then, the tag uses backscatter to send, in the most cases only, the identifier back to the antenna. Figure 2.4(b) shows two antennas on the left and a reader on the right of the image.

The communication antenna-tag-antenna is done through the “air”, meaning that every thing in the middle can interfere with this “link”, such as the presence of metal or human bodies. Water is known to absorb radio waves while metallic objects reflect them. Further, when in the same space there are several readers installed, collisions between them may occur and the signal from the tag may be get lost. Yet, there are some strategies to reduce these constraints at hardware

and middleware level. More information can be consulted in Nikitin & Rao (2008) and Bolic *et al.* (2010) about how RFID readers and antennas work, as well as the main constraints faced by these components.

RFID Middleware As shown in Figure 2.5, the RFID middleware sits between the low level components and the business client application. Standardized interfaces are available to isolate hardware vendors from business applications.

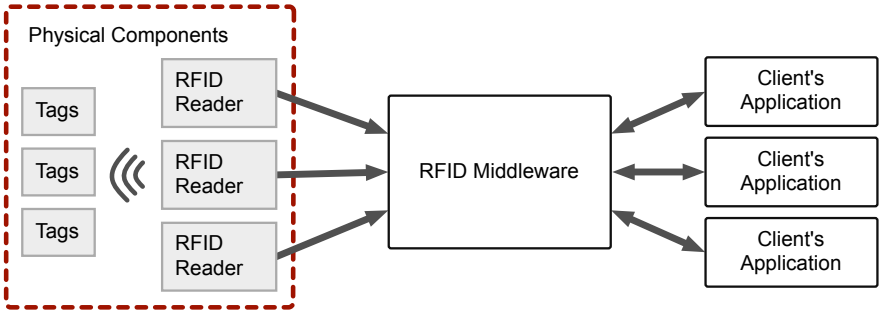


Figure 2.5: RFID middleware between physical components and client's application.

As described in the next section, efforts from major organizations are being made to enforce the standardization of both modules and protocols.

2.1.2 GS1 EPCglobal Architecture

GS1[1] is the organization that is promoting the use of the *Electronic Product Code* (EPC) standard (EPCglobal (2013)) which is a unique serial identifier for RFID tags. The GS1's subsidiary EPCglobal Inc[2] defined a specification based on the constraints and application requirements called *EPCglobal Architecture Framework* (Traub *et al.* (2010)). It represents the standard in terms of RFID platforms with several layers and components (Figure 2.6).

[1]More information about GS1 can be found at: `http://www.gs1.org`
[2]More information about EPCglobal Inc can be found at: `http://www.epcglobalinc.org`

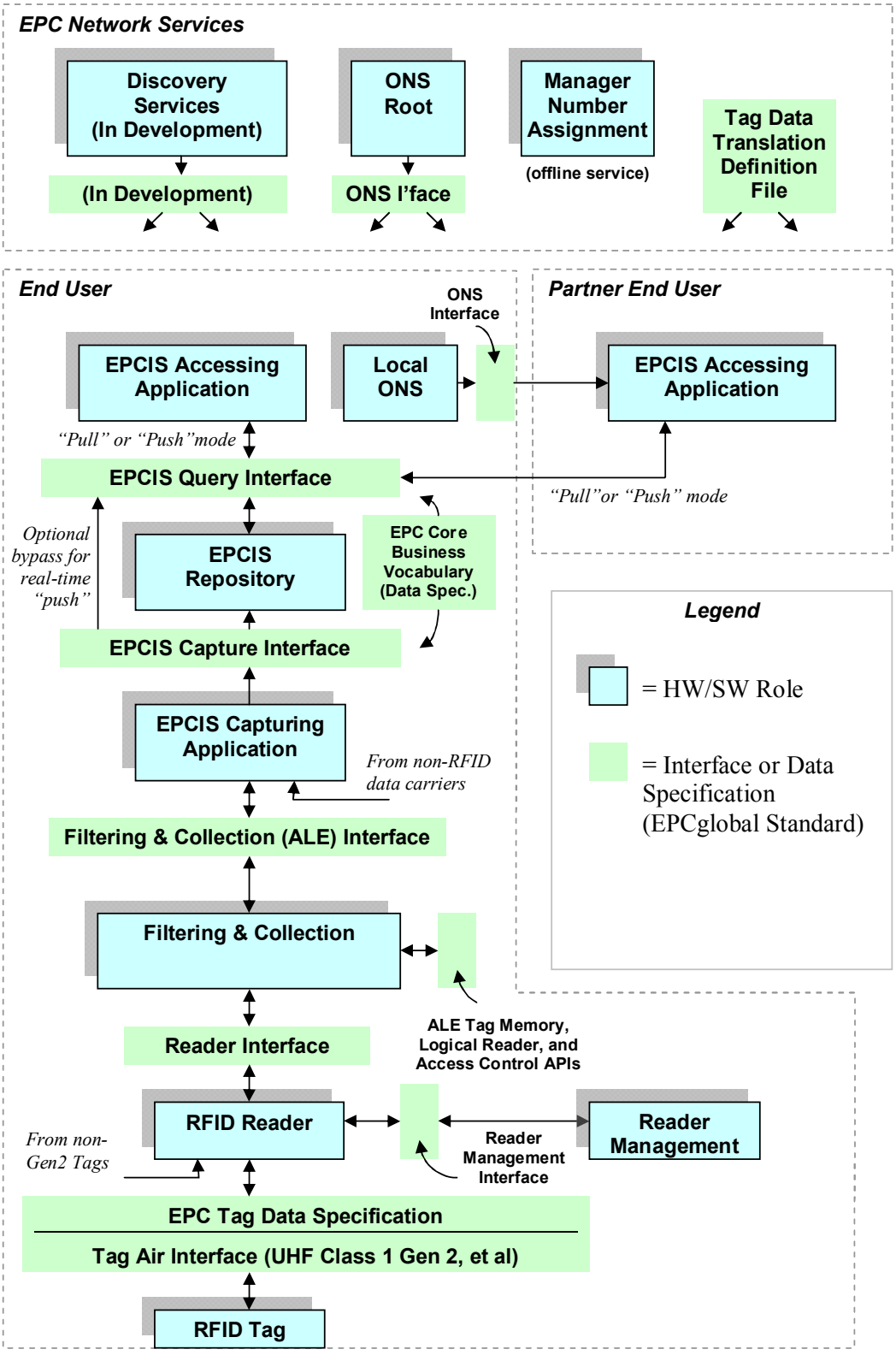


Figure 2.6: GS1 EPCglobal Architecture Framework (Traub *et al.* (2010)).

The standardized interface of the framework enables the interchange of information between entities. For the purpose of this work, the most important modules in the architecture are:

- *Reader Interface* specifies the *Low Level Reader Protocol* (LLRP) that readers should implement as specified in EPCglobal (2010). This represents the lowest middleware layer.
- *Filtering & Collection* is a module where RFID readers are abstracted from the upper layers. It allow low level operations on tags like read and write. Besides, it should filter tag reads and aggregate them if requested.
- *Filtering & Collection (ALE) Interface* specifies how client applications can interact with filtered and aggregated data. The *Application Level Events* (ALE) are a selection of events that client applications are interested in (EPCglobal (2009)).
- *EPCIS Capture Application* is the *EPC Information System* (EPCIS) module responsible for translating ALE events into meaningful business events. It could make a direct mapping between both layers or it could consult external sources (eg. an ERP system) to produce an event. The resulting events are sent to the *ECPIS Repository* through the *EPCIS Capture Interface*.
- *EPCIS Repository* is where all the meaningful business events are stored to be requested by an *EPCIS Accessing Application*. The *EPCIS Query Interface* defines how external client applications can retrieve information (EPCglobal (2007)).

2.2 RFID Platforms

The EPCglobal Architecture Framework is widely used on current RFID platforms. The next subsections describe the most relevant platforms based in the EPCglobal Architecture giving particular detail to the one used in the present work.

2.2.1 Fosstrak

The *Free and Open-Source for TRAcK and trace* (Fosstrak), designed and implemented by Floerkemeier *et al.* (2007), is an open source platform compliant with EPCglobal Network specifications, freely available to any company and / or researcher that wants to use or develop on top of it. Figure 2.7 shows the architecture with all the available modules. Comparing with EPCglobal architecture presented in Figure 2.6 the Fosstrak platform covers all the “End User” components except the “Local ONS” module.

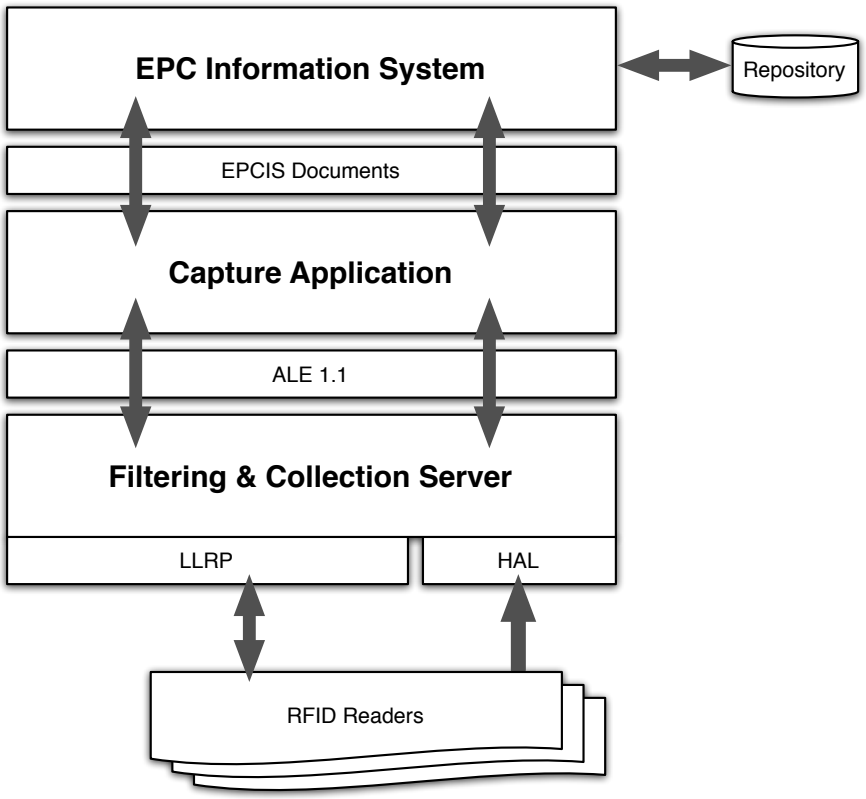


Figure 2.7: Fosstrak Architecture.

Filtering & Collection Server FCServer is the module responsible to abstract the readers from the rest of the Fosstrak's architecture. It has two types of connections, one named *Hardware Abstraction Layer* (HAL) where it is possible to develop a custom connection to each reader. And the LLRP connection to enable all LLRP complaint readers to be connected to the middleware just by configuration. Internally, these readers are *LogicalReader* instances and configured through a *LRSpec* document as defined by EPCglobal (2009).

Fosstrak implements the mechanism specified in EPCglobal (2009) to reduce the number of false readings called *Event Cycle*. This mechanism is implemented in the *Data Cleaning* module where it is possible to programmatically change the data cleaning strategy. The output of the *Data Cleaning* module is an *ECReport* document that is sent to the *Capture Application*.

Capture Application Before a business event is sent to the *EPCIS Repository*, the capture application may request more information about the identified object to external services. For example, a specific product is identified in the reader deployed in the warehouse's entrance, the capture application can now request information about this product from the ERP system to know if it is an expected product from a supplier or a returned product from a costumer.

Regarding its implementation, the Capture Application is built on top of a Drools[1] engine where rules can be specified in the form of: "when" a tag is received from the warehouse entrance, "then" do "this". Unfortunately, all rules must be predefined and can not be changed during runtime if needed.

EPCIS This module is EPCglobal-certified meaning that all Fosstrak EPCIS modules and interfaces are verified by the EPCglobal organization. Additionally, it implements a *Query Callback Interface* on which those events matching a specific criteria are sent. Criterias are defined through subscriptions that must be previously configured by an *EPCIS Accessing Application*.

The *EPCIS Repository* stores the following information for each business event: *a*) *Read Point* where the read was taken, for instance in the warehouse's dock door; *b*) *Business Location* where the tag was identified, for instance the warehouse as a whole or other specific area; *c*) *Disposition* specifies the state of the tagged object, for example the tagged product is in the packaging area; and *d*) *Business Step* is the business action associated with the event, normally it is used a verb to represent the action being taken, for instance entering the packaging area.

2.2.2 Rifidi Edge Server

The *Rifidi Edge Server*[2] is an open source middleware platform. Its main components are shown in Figure 2.8. Comparing it with the EPCglobal Architecture Framework, Rifidi has a *Sensor Abstraction Layer* where RFID readers are connected. Besides, the platform has a filtering and a collection layer called "Application Engine Layer" where an event processing bus service is used to aggregate and filter raw data. Finally, the *Communication Layer* exposes the events through several outputs.

Despite having a limited compliance with the EPCglobal framework, it has a strong simulation component of RFID environments, enabling designing scenarios to test and demonstrate the technology.

2.2.3 BizTalk RFID

The *BizTalk RFID* is a middleware from Microsoft Corporation for RFID solutions. Its main advantage is the capability to interoperate with other layers and modules from Microsoft technologies (Simms *et al.* (2009)). Figure 2.9 shows the architecture of *BizTalk RFID* server.

[1] More information about Drools can be found at: `http://www.jboss.org/drools/`
[2] More information about *Rifidi* solutions can be found at: `http://www.rifidi.org`

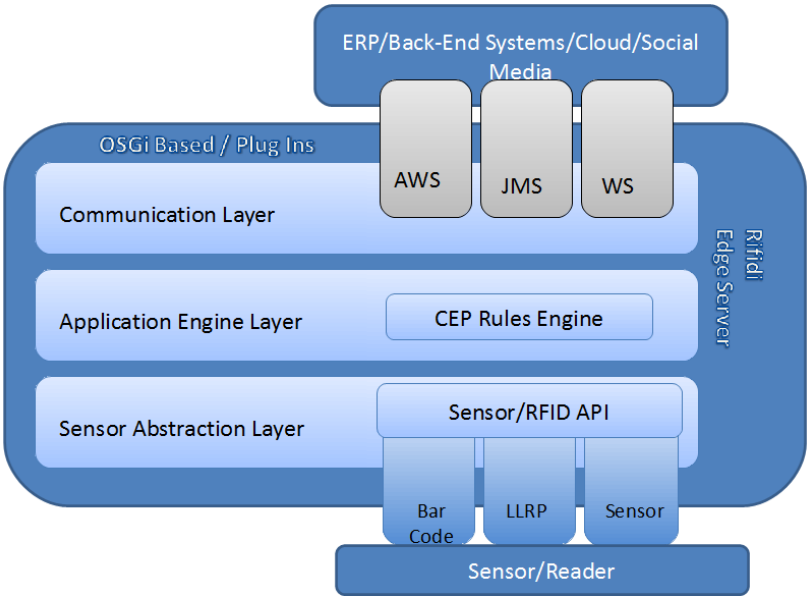


Figure 2.8: Rifidi Architecture.

Regarding the EPCglobal compliance, the *BizTalk RFID* server has the "Reader Interface" to communicate with the standardized readers. Its version of the *Filtering & Collection* module is the *Process Engine & Runtime* module where rules are implemented to process all readers events. Also, it has a simplified *Capture Application* to generate EPCIS documents and store them in a local database. The *BizTalk RFID* server does not provide a full EPCIS implementation but there are online documentation on how to implement the capture and the query interfaces using the *BizTalk Server* component.

2.3 Data Cleaning Strategies

Reliability problems in the communication between antennas and tags can be mitigated through several ways. The *false positive* and *false negative* readings are the product of the lack of reliability in the link antenna-tag-antenna. The data cleaning module is an important piece to minimize these erroneous reads.

As stated by Darcy *et al.* (2009), the following strategies try to minimize and dissipate the occurrence of false reads, and they could be applied at several levels: *a*) physical solutions; *b*) middleware solutions; and *c*) deferred solutions.

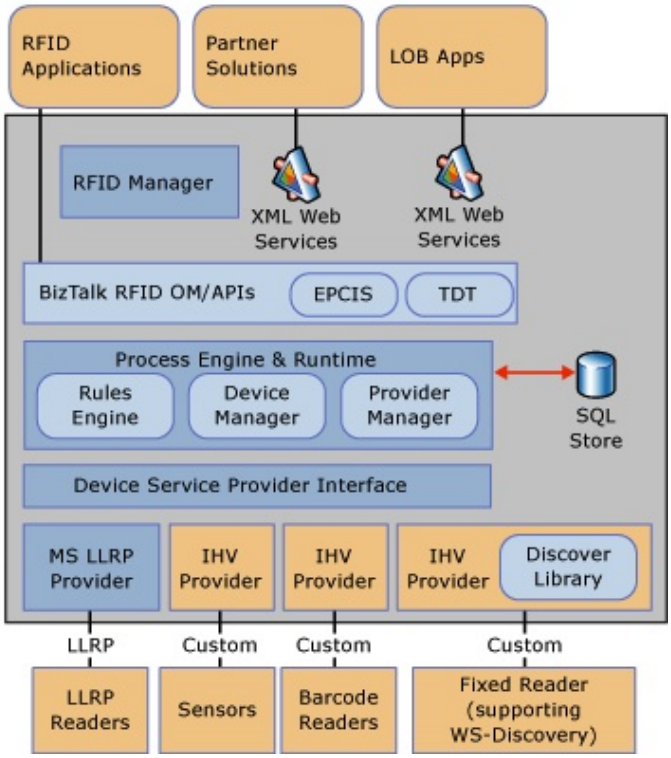


Figure 2.9: BizTalk RFID Architecture.

2.3.1 Physical Solutions

Hardware can be improved in order to increase its performance as shown by Trotter & Durgin (2010). Another approach is to have redundant hardware such as more tags to identify the same object or more readers to cover the same area. Rahmati *et al.* (2007) explored in detail how the orientation of the tag regarding the RFID antenna affects the read efficiency. Also, Chen *et al.* (2010) studied how duplicate readings from an array of antennas can reduce the false readings.

Empirically, the easiest way to improve the overall reliability is deploying more readers to cover the same area. But despite being a costly solution, readers collisions can degrade the overall efficiency of the system as pointed out by Jakkhupan *et al.* (2011).

2.3.2 Middleware Solutions

Middleware platforms can receive events from the antennas and apply algorithms and strategies to clean most of the erroneous readings. The EPCglobal Network specifies a window of time in which the platform receives the reader events to later report them to clients applications. This and other approaches are detailed below.

2.3.2.1 Sliding Window

A sliding window is a period of time in which the platform waits for reader events. The events are then aggregated and reported to the client's application. In the Fosstrak implementation, sliding windows are configured through the *EventCycle Specification* document where the size and duration of the sliding window is defined. These configurations, however, are static so they can not be changed in the runtime.

As it was experienced, and Jeffery *et al.* (2006) work corroborates it, finding the appropriate value for the sliding window is not a practical task since it varies according to the business needs and scenario conditions, forcing a tailored sliding window value. Additionally, in certain situations unexpected objects can appear within the reading area causing interferences that may affect the performance of the system in terms of false events. Ideally, the sliding window size should be adapted to the conditions to compensate the disturbances that may appear during operation.

2.3.2.2 SMURF

Jeffery *et al.* (2006) introduced the *Statistical sMoothing for Unreliable RFid data* (SMURF), a declarative and adaptive sliding window technique to filter RFID reader's data. It views the RFID data as a statistical sample of tags in the physical world. This assumption allows sampling theory techniques to be explored to drive the cleaning process. The key idea behind SMURF is that RFID readers' data streams can be modeled as a random sample of the tags in the readers' detection range.

SMURF algorithm uses its sampling operations in a per-tag basis rather than being per-reader basis. It tries to continuously adapt the sliding window size for each tag based on the observed readings. Yet, it must balance and distinguish between periods of tag dropped readings and periods when a tag has left the reader area.

For SMURF, a tag event represents a sample of the state of the world, it means that it does not assume that an event corresponds to the presence of the tag, rather it assumes the presence with a certain value of probability. This probability value will directly affect how the algorithm judges the presence of tags, dynamically changing the window size for each tag. Initially the window is 1 and it grows additively and decreases multiplicatively depending on the current window size and tag's probability value. The probability function can be any that performs best in a specific scenario and with the available platform. So, this value will be as accurate as the amount of information available for any given tag.

Massawe *et al.* (2012a,b) extends SMURF by proposing a comparison between two consecutive sliding windows to detect transitions more efficiently. Fan *et al.* (2012) propose an approach where the probability function for each tag considers its kinetic properties.

Despite the original algorithm was proposed in 2006, there are still research work being made on top of SMURF. It is a disruptive and solid approach that may lead to better algorithms.

2.3.2.3 Evaluation Metrics

The work of Jeffery *et al.* (2006) (SMURF algorithm) use as a evaluation metric the *number of errors per epoch*. An epoch is a fixed period of time, typically around 0.2-0.25 seconds. The number of errors per epoch can be calculated with the following equation:

$$ErrorsPerEpoch = \frac{FalsePositives + FalseNegatives}{NumEpochs}$$

Although Jeffery *et al.* (2006) do not provide any explanation on how to obtain the *false positives* and *false negatives* readings, it is quite probable that are extracted from the synthetic generated data used at their studies.

Other works that use or extend the SMURF algorithm, Zhang *et al.* (2011), Fan *et al.* (2012), Massawe *et al.* (2012a,b) and Xing & Wen-xiu (2013), used the reading rate to evaluate their approaches. More specifically, Zhang *et al.* (2011) compare error against noise percentage to later calculate an accuracy rate obtained from the reading rate. Fan *et al.* (2012), besides the reading rate, use the error rate against tag speed to evaluate the impact of a tag moving in the data cleaning efficiency. Massawe *et al.* (2012a,b) analyze the number of errors per epoch against several parameters, eg. tag velocity.

Cao *et al.* (2011) present a design of a scalable and distributed RFID platform, and evaluates its data cleaning algorithm with the following metrics: *i*) *Error Rate* (%) to compare the inference module results with real world; *ii*) *F-measure* to evaluate the accuracy of the reported changes; and *iii*) *Running Cost* to measure the time that the system takes to process a data trace. The metric *Running Cost* only captures the total time needed to evaluate a specific data set, ignoring how system resources are being used.

Ahmed & Ramachandran (2009) work focuses on increasing the reliability of the entire system by reducing the rate of the *false negative* readings. They assume that the amount of data will increase and systems should be prepared to correctly process all the information. They introduce the notion of *virtual paths* and *virtual readers* to allow the distribution of the readings among several software nodes. A *virtual path* represents a logical channel along a set of predefined *virtual*

readers that represent physical readers. This approach is only suitable in scenarios where the physical conditions does not change very often. Regarding metrics, they use accuracy as being the ratio between the number of correctly identified items and the total number of items along a specific virtual path. Although they use generated and real-world data from a testbed, it is not mentioned how erroneous reads are obtained.

Tu & Piramuthu (2011) propose a few algorithms to reduce the false readings to enhance the system reading rate. Both false readings types are taken into account in the algorithms. Furthermore, the count of *false positive* and *false negative* readings are used as the evaluation metric. Liu & Zheng (2012) also propose several models to reduce the number of false readings, but they use the *false negative* and *false positive* reading rate and not the global value.

The contribution from Müller *et al.* (2010) establishes how to evaluate RFID software components. The article presents: *a*) several ways to generate test data; *b*) metrics to use when a RFID system is evaluated; and *c*) how to setup a test environment. Regarding metrics, they consider: *i*) *Apdex*, a metric to evaluate the system's response time; *ii*) *tasks throughput* over a time period; *iii*) *memory and database memory usage*; and *iv*) *availability* of the system. Their main focus was to correctly evaluate the EPCIS Repository component performance. *False positive* and *false negative* readings were not considered.

Finally, Schmidt *et al.* (2011) use a different approach regarding the data cleaning module. They believe that an ALE instance can be connected with several hundred readers and so they propose a distributed ALE engine based on a Distributed Hash Table (DHT). Since some information about the identified objects could be lost along the distributed system, the work uses the percentage of report completeness. That is, at the end of an experiment, all the reports of tags entering and exiting are collected and verified if the system reports was as expected. Instead of an absolute number they use a percentage value of false reads.

2.3.2.4 Other Strategies

Cao *et al.* (2011) proposes a distributed inference model to mitigate the false readings from the RFID readers. Liu & Zheng (2012) presents a comparison between seven algorithms to filter the readers supply chain raw data. To finalize, Xing & Wen-xiu (2013) propose the *Sliding Window based on Kalman Filter pre-processing* (SWFK) where its data cleaning module uses a Kalman filter implementation in conjunction with the sliding window to analyze all reader events.

2.3.3 Deferred Solutions

Deferred solutions means that data is analyzed in later stages in order to find errors and correct them, typically using data intelligence techniques on top of the data. Rao *et al.* (2006) work relies on the cleansing of data at query time. That is, when a client application uses the *EPCIS Query Interface* to request data, different policies are applied to the stored data in order to give a more accurate vision of the world.

Darcy *et al.* (2009) present an approach to recover some of the missing data caused by *false negative* readings. Their proposal is to have an intelligent data analysis module and a reasoning engine designed to recover the data not reported by the RFID readers. This data analysis is done before business data is stored and ready to be delivered.

These approaches generally suffer from the delay introduced by the reasoning modules where historical data must be analyzed. Ideally the middleware receive all the possible events to analyze them with as much information as possible. This approach is not useful in contexts where real-time answers are needed.

2.4 Summary and Discussion

The RFID reliability issue is a serious problem when clients applications requires precise and accurate data. In detail, the *false positive* and *false negative* readings can affect the conclusions made by client systems. Therefore, the data cleaning module must be improved to be more efficient giving the realistic vision of the world.

The middleware platform Fosstrak is a good choice for everyone that wants to deploy an RFID system that is EPCglobal compliant. It is open-source so it is possible to extend with new features both in academical and commercial spaces. One of its modules is the Data Cleaning where the noisy stream of events coming from the antennas are filtered. SMURF shows to be a reliable and solid algorithm given its disruptive approach in contrast with the static-sliding window configuration. Further, it continues to be developed and improved in recent works with new tag parameters to reduce even more the number of false reads enabling more reliable systems avoiding costly hardware solutions.

Despite there are other approaches, all of the data cleaning strategies are sliding windows extensions. Even SMURF, that has a statistical vision about the events, also uses the sliding window technique.

Chapter 3

RFIDToys Testbed

This chapter proposes a platform to easily test an RFID system. Together with Chapter 4, where a comprehensive methodology is described using this testbed, both enable practitioners and researchers not only to quickly demonstrate, but also to access a platform where new configurations can be tested, current modules enhanced and new functionalities implemented.

The main requirements that shaped the testbed architecture were the following: *a*) offer a testable platform for RFID technology; *b*) a modular platform where some components can evolve or be switched by others without affecting the others; *c*) be able to change some configurations to easily adjust the scenario if needed; and *d*) hide some of the RFID complexity to be more accessible for a wider range of users. Figure 3.10 shows the testbed high level architecture.

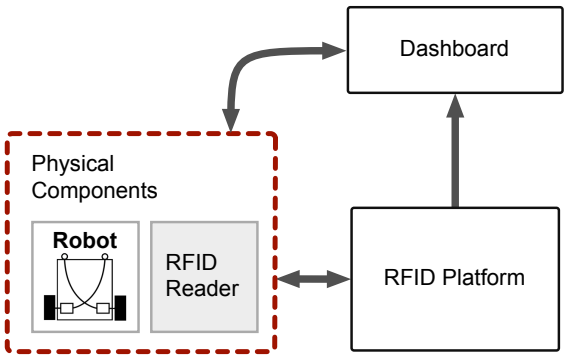


Figure 3.10: Testbed high level architecture.

The RFID middleware platform could be any of the EPCglobal compliant platforms currently available. The Fosstrak platform was chosen because it is open source and it is available to use, extend and modify.

The dashboard is where the operator can easily configure some aspects of the experiment

and also get visual feedback from it. For example, it is possible to program the robot to move and stop at a specific point and receive an event from the RFID platform informing the robot has arrived.

All the test configurations can be changed to represent other test cases. Additionally, within the same test case the environment conditions can change intentionally. For instance, a metallic object can be introduced in the experiment environment to check its influence / impact both at the raw readings and business level events.

The reason why having a physical testbed instead of a synthetically generated data set is to experiment with scenarios that best replicate the deployment interferences that usually real world has. Generated data sets are always smoothed by a particular model and therefore are not rich enough to test a system which is supposed to be used in real environments. The module Rec&Play can answer this requirement allowing the recording of a physical session to later be replayed exactly in the way it happened. Plus, the differences of the recorded sessions can be studied and analyzed to understand with detail what happened. This way, the same physical session can be used to test as many software configurations as desired.

3.1 Architecture

Figure 3.11 presents the detailed architecture diagram and components that are described in the next subsections.

3.1.1 Middleware Platform: Fosstrak

For the present work, Fosstrak did not meet all the testbed requirements mentioned earlier in this chapter, therefore in the architecture (Figure 3.11), the FCServer component was extended with the following components: *i*) the Rec&Play library to record and replay RFID sessions; and *ii*) refactoring of the data cleaning module to be able to choose between data cleaning algorithms.

3.1.1.1 Rec&Play

The Rec&Play was developed as a self-contained and generic library, allowing to be used in a completely different domain. The Rec&Play module has the following features: *i*) manage different sessions; *ii*) store objects keeping the order and time from the beginning of the session; and *iii*) ability to replay a session where each object is delivered in the correct order and delay.

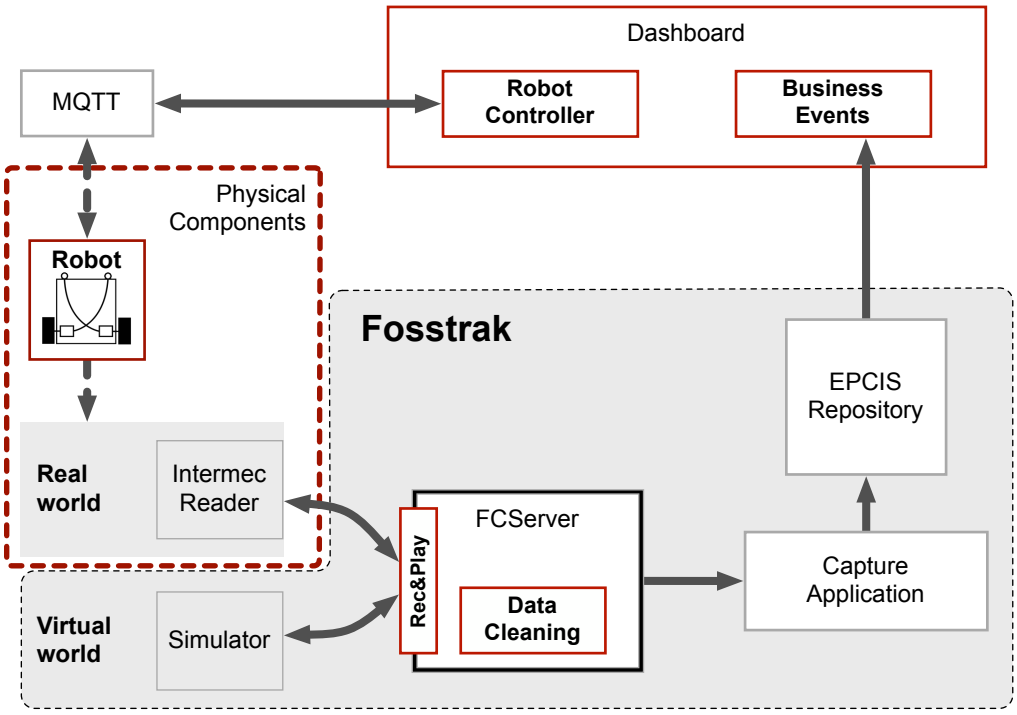


Figure 3.11: Testbed architecture.

With the developed library, it is possible to have pre-recorded sessions to test a specific component of the RFID middleware with several configurations or even different implementations to later compare the results. Indeed, it is possible to maintain all the "real-world" events such as interferences and duplicates to stress the data cleaning module. This way, it is easier to discover which configuration or algorithm provides better results.

The Rec&Play library was developed in Java to allow an easy integration with the Fosstrak platform. However, as it is stated by Tanenbaum (2008), the *Java Virtual Machine* (JVM) is not a real-time system, and so the library could introduce delays that should be taken into account. Thus, profile tests were done to study its precision and the library showed a mean deviation of 0.6 milliseconds for a set of 100 objects. For $1\ 000$ objects the value raised up to 2.4 milliseconds.

The most important aspect to understand is how the library average delay is compared with the average cadence of the expected objects. For instance, if the delay between each object is less than 2 milliseconds then, the library should be refactored or not being used. Looking for the RFID environment and the tests made, the cadence of received objects from the RFID antennas were in average 57 milliseconds. So, in this case the maximum error presented by the library will not significantly affect the results.

The Rec&Play integration into Fosstrak's platform was done at the logical readers level where LLRP messages can be read and therefore saved by the library (diagram 3.12). Similarly, the replay functionality was integrated at the logical readers level. As a result, it is guaranteed that both recording and replaying of LLRP messages happen at the lowest level in the middleware architecture.

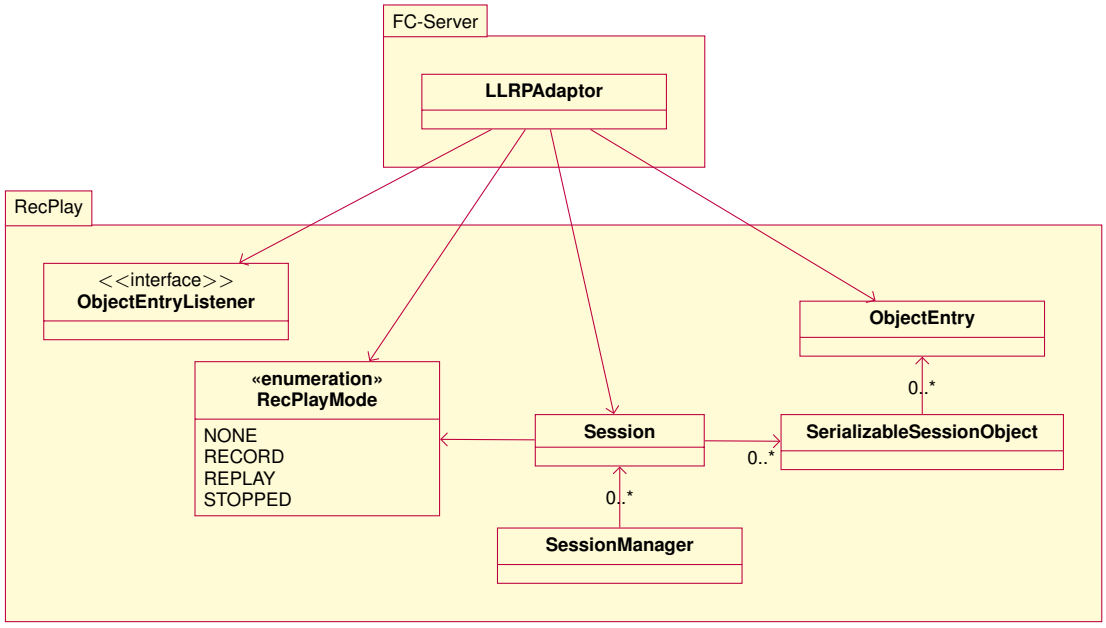


Figure 3.12: Rec&Play library class diagram.

The Rec&Play configuration is done by adding a new property "RecPlayMode" to the *LogicalReader* specification file, as can be seen in Listing 3.1.

Listing 3.1: Rec&Play module integration in *LogicalReader* specification file.

```
<LRSpec>
  <isComposite>false</isComposite>
  <properties>
    ...
    <property>
      <name>RecPlayMode</name>
      <value>replay</value> <!-- possible values: record or replay -->
    </property>
  </properties>
</LRSpec>
```

3.1.1.2 Data Cleaning Module

The data cleaning module is part of the Fosstrak's FCServer module and is responsible for filtering and aggregating the readings from the physical layer. Regarding its implementation, the *LogicalReader* connects to the reader to receive its events. Then, they are sent to the *EventCycle* entity that filters and aggregates all the events to produce reports for the clients applications.

One side the EventCycle registers itself to receive notifications from the *LogicalReader*, and on the other, sends an "ECReport" (Event Cycle Report) to the registered clients. Further, the *EventCycle* is instantiated as defined in the "ECSpec" (Event Cycle Specification) document where is defined how to aggregate events, from whose to read and to whom report.

Figure 3.13 shows the refactoring made in the Fosstrak's data cleaning module. Instead of the *EventCycleImpl* being directly instantiated by the *ReportsGeneratorImpl*, it uses the configuration specifications file to know which data cleaning implementation it will load. In the present work, the *EventCycleSMURF* was created with the SMURF algorithm implementation (described in section 2.3.2.2) and, as it is defined by the interface, receive the antennas events from the *LogicalReader* objects. This way, it is possible to compare SMURF performance with the static sliding windows which are the default Fosstrak's data cleaning algorithm.

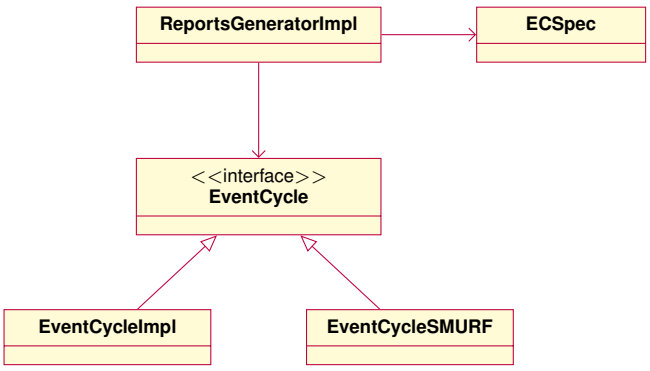


Figure 3.13: Refactoring of Fosstrak FCServer data cleaning module.

On the configuration side, the changes were reflected in the "ECSpec" document where it is possible to add extensions. Listing 3.2 depicts the new XML node "datacleaning" added to the ECSpec file which specifies the algorithm to use. Since a *ECSpec* document only configures a single *EventCycle* instance, it is possible to define two *EventCycle* using two different algorithms both working at the same time even if connected to the same readers.

Listing 3.2: Data cleaning module configuration.

```
<ECSpec>
  <logicalReaders>
    <logicalReader>SomeReader</logicalReader>
  </logicalReaders>
  <reportSpecs>
    ...
  </reportSpecs>
  <datacleaning
     implementation="org.fosstrak.ale.server.impl.EventCycleSMURF"/>
</ECSpec>
```

3.1.2 The Robot

The robot was used to guarantee a flexible and "portable" testbed. The Arduino platform[1] was chosen for its modularity, ease to use and its expandability to add more features and functionalities. For instance, it is possible to add a pair of arms to the robot to hold and transport objects, or a multi-color sensor to allow the differentiation of multi-color paths or objects. In terms of software, the Arduino platform offers the essential tools to build and expand its source code.

Evans *et al.* (2012) and Monk (2010) work provides extensive information on the Arduino platform and, specifically, on how to assemble and load the software.

Minimal parts that guarantees the following requirements were chosen to build a robot: *i*) ability to receive external messages with the sequence of actions to execute; *ii*) ability to follow a line, that is, a black line on a white surface; *iii*) ability to go through an intersection, that is, going through a black line; *iv*) ability to stop for a number of seconds; and *v*) ability to repeat a set of actions a number of times. Altogether, these make the minimal subset of actions needed to have a simple scenario using the robot.

Figure 3.14 shows the components used in the robot. The bluetooth is the bridge to send and receive messages from the *MQ Telemetry Transport* (MQTT)[2] server. Two line trackers are used to "see" the floor to enable to robot to follow a black line in a white space. Finally, the Arduino board controls all the actions to be executed.

Internally, the robot has three states. When a message is received the robot tries to execute it. If the action says to the robot to move it goes to the state "moving", when the special action "hold" is received, the robot shifts its state to "hold commands". Then, the forthcoming messages are all stored in an internal queue until an action "release" is received. When it happens, the robot goes to the previous state and starts executing the actions from the internal queue.

[1] Arduino web site: `http://arduino.cc`
[2] More information about MQTT can be found at: `http://mqtt.org`

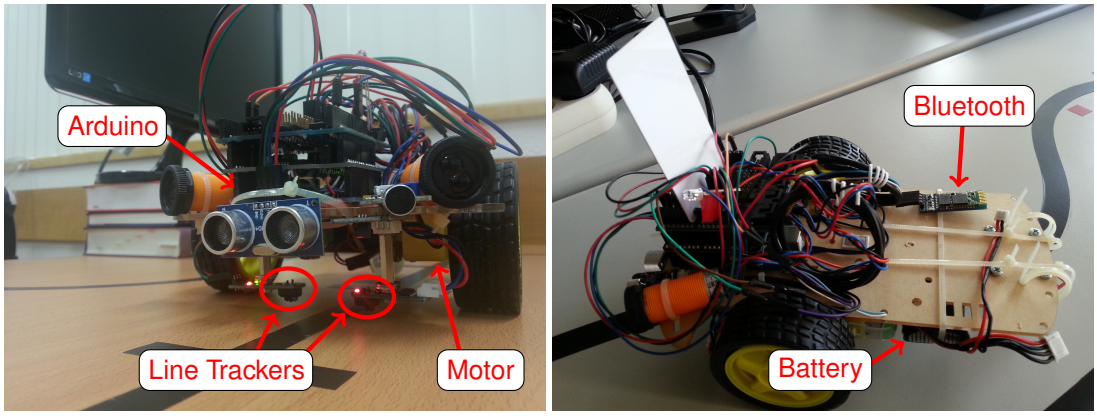


Figure 3.14: Robot components.

3.1.2.1 Track

The feature to have the robot completely decoupled from any external guidance was considered, but it was left as future work because of the effort needed. Instead, two line trackers were used in the robot to enable it to follow a black line.

On the setup, two RFID antennas were used facing each other in both sides of an office desk. Figure 3.15 shows a photo of the track used in the experiments with both antennas. In the experiments only the closed track was used to form a closed circuit connecting both antennas. A straight track was initially considered but it was not used in this work because the results achieved were not enough to take meaningful conclusions.

The track can explore different properties like the way the tag is oriented or the way the tag moves in front of the antenna. In addition, the speed of the robot can also be changed to measure the impacts in the results. With this track the robot can be more autonomous, making any number of laps.

The track is divided in segments and with a set of segments read areas can be defined. In Figure 3.15, the track is divided in 8 segments numerated counter-clockwise. The points of interest, let us call them *flag points*, are when the robot passes from:

- segment I to II flags the entrance in the read area, let say, *A*.
- segment III to IV flags the exit from the read area *A*.
- segment V to VI flags the entrance in the read area *B*.
- segment VII to VIII flags the exit from the read area *B*.

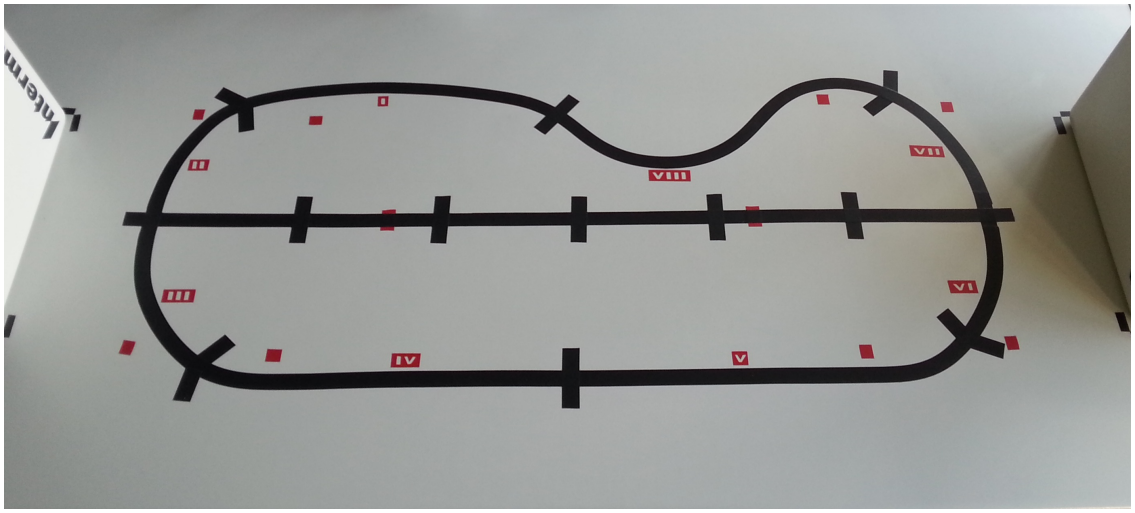


Figure 3.15: Track with segments and read areas.

The other marks in the track are called *pause points*, where the robot should not flag its passage. They are useful to add sleeping points where the robot can stop for a number of seconds. Since they are not used for any metric, the *pause points* can be marked on the track without any major consideration. However, the *flag points* should be marked carefully because they will dictate the entrance and the exit of the reading areas and therefore the time that the tagged object spends inside a reading area.

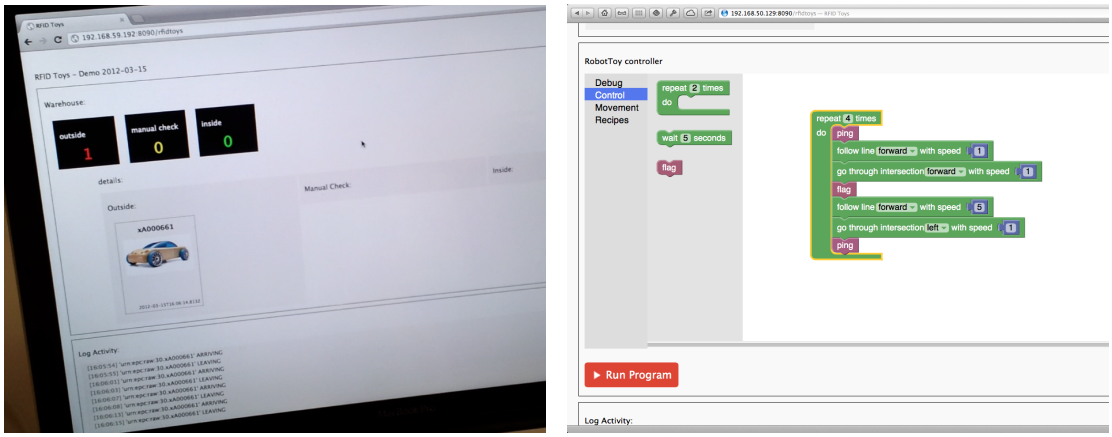
To carefully mark the limits of the reading areas, the following procedure was used: *i*) first the RFID antennas location was set; *ii*) the antennas were turned on and connected to a testing application that comes with the antenna[1]; *iii*) a sound is emitted when one of the antennas detect a tag; *iv*) pass the tagged object in front of the antennas through the track until the sound stops; and *v*) repeat the previous step until a point is identified as the boundary of the antenna field. A video to demonstrate the procedure can be found online[2].

3.1.3 Dashboard

The dashboard is the user interface for operating the platform. It has two main components, both in Figure 3.16: *a*) *business activity monitor*; and *b*) a *robot controller*. Business events generated by the RFID middleware can be visualized in the business activity monitor. It receives *EPCIS Reports* from the *EPCIS Repository* where a query was previously subscribed. On the Robot Controller (Figure 3.16(b)) is possible to visually program what the robot will receive and execute.

[1] It was used in this work the Intermec reader IF61 which has a testing application "JRFID APPLICATION" available at: `http://www.intermec.com/products/rfidif61a/`

[2] How to find the *flag points* video: `http://web.tecnico.ulisboa.pt/~nuno.correia/msc-rfidtoys/demonstration-flag_points/`



(a) Business Activity (b) Robot Controller

Figure 3.16: Dashboard interface: business activity and robot controller.

The dashboard is built in Javascript using the NodeJS[1] runtime with several libraries that helped to speed up the development. The communication between the server and the client browser is done using WebSockets[2] that enables a bi-directional communication channel between both parties. The communication between the server and the robot is done through the MQTT server that allows an easy and lightweight message exchange with the robot. The MQTT protocol is specially designed for devices with sparse resources like the robot that was built. The MQTT adoption is being pushed by the *Internet of Things* community for its lightweight properties.

3.1.3.1 Business Activity

The business activity component shows what is happening from the business perspective. Figure 3.16(a) shows an example where the total number of objects in each area are displayed and also the detailed view of the objects inside each business area.

When introduced objects cause interferences, the *Business Activity* can immediately show the impacts. Despite being simple, the tool can be extended to other scenarios.

3.1.3.2 Robot Controller

This module is used to program a robot giving a sequence of actions. The robot can also send back messages with its status or debug information.

[1] More information about NodeJS can be found at: `http://nodejs.org` and Teixeira (2012).
[2] More information about WebSockets can be found at: `http://websocket.org`

Figure 3.17 shows a set of instructions that will program the robot to[1]: *i*) repeat the following actions 4 times; *ii*) follow a line; *iii*) pass over an intersection; and *iv*) stop for 5 seconds. Alternatively, other combinations can be done and even new commands can be developed to allow a wider range of control possibilities, e.g. “go to point A” and “then turn left”.

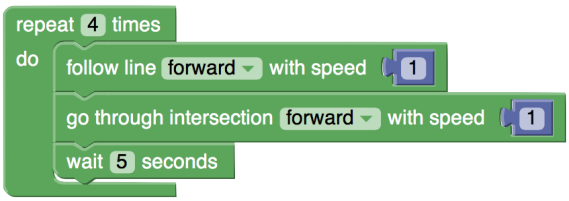


Figure 3.17: Robot programming example.

The visual interface is based on the open source project Blockly[2]. It is an extensible framework that allows a full customization of its “blocks” not only in terms of the visual aspect but also the output that every block generates. Moreover, it has a built-in code generator that translates every block to a piece of code which, at the end, is the sequence of instructions that make the final “program”. Particularly, in the robot controller, this final “program” is sent through WebSockets to the dashboard server and then forwarded to the robot through the MQTT server.

3.2 Summary and Discussion

This chapter introduced the testbed and the main modules that constitute it. The whole setup allows flexibility and testability. This way, it is possible to easily demonstrate the RFID technology.

The chosen middleware platform was improved with the Rec&Play library to save and repeat physical world sessions. The data cleaning module was refactored to allow the choice of a different data cleaning algorithms. The Rec&Play module is an important piece in the testbed because it reproduces exactly what happened in the real world, allowing a truly statistical approach.

The robot offers flexibility on what can be done in an RFID experiment. The track plays an important role since it enables the robot to flag the read areas automatically. This is convenient feature given that human bodies can interfere with the RFID air protocol.

Finally, the dashboard gives an overview on what is happening in the system. The technologies used offer the option to easily extend to other dashboards or scenarios.

[1] Robot Controller demonstration video is available at: `http://web.tecnico.ulisboa.pt/~nuno.correia/msc-rfidtoys/demonstration-robot_controller/`

[2] More information about Blockly can be found at: `http://code.google.com/p/blockly/`

Chapter 4

Methodology

The present chapter describes a methodology to correctly use the testbed platform, introduced in Chapter 3, providing representative samples of the experiments, to obtain meaningful results.

The proposed methodology is presented in Figure 4.18. The ultimate goal is to obtain statistical valid results while coping with the natural unpredictability of an RFID system. Briefly, the steps are: *a*) definition of the experiment; *b*) record phase where the experiment will be repeated a number of times while the RFID antenna readings are recorded in sessions; *c*) replay phase where the previously recorded sessions will be used as inputs to test all the configurations needed; and *d*) results processing.

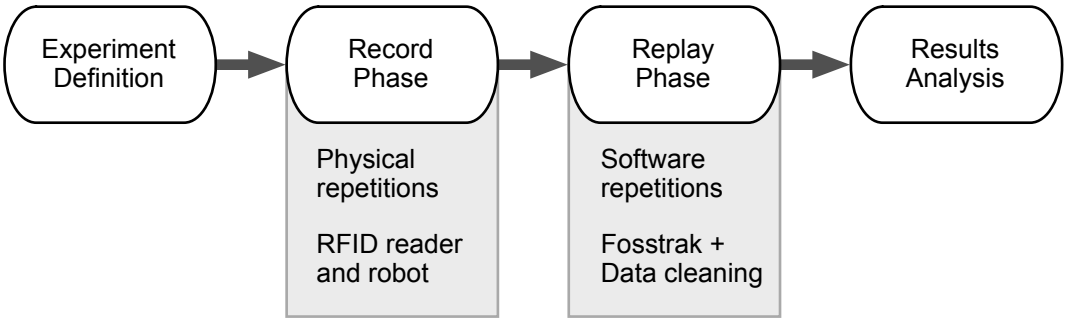


Figure 4.18: Experiment methodology steps.

With the proposed methodology, only a few number of physical experiments must be repeated so that a large number of software configurations could be tested repeating the previously recorded sessions. Doing so, the variability in test results will be lower and, more importantly, it will be possible to compare among test results.

The following sections describes in detail the phases of the procedure.

4.1 Experiment Definition

The first step is clearly define the experiment goals, setup and conditions under it will be run, as well as, to select the metrics that will be used to check if the expected results were achieved. For the present work the its ultimate goal is to find the best data cleaning configuration in terms of: *i*) least presence of errors; and *ii*) better value of presence rate. The metrics created were $PresenceErrors$ and $PresenceRate$. Besides specific data cleaning metrics, system metrics are also considered to evaluate the overall performance of the system.

The system metrics used in the testbed are the $ProcessCPULoad$, $SystemLoadAverage$ and $MemoryUsage$. The $ProcessCPULoad$ indicates the percentage of time the process if active in the available CPUs. $SystemLoadAverage$ is the average number of processes that are in the runnable state and currently running in one of the available CPUs. Finally, the metric $MemoryUsage$ indicates the memory current usage, in the case of a JVM process is the heap memory in bytes. Detailed information about these metrics can be found in Appendix B.

4.1.1 Data Cleaning Evaluation Metrics

Besides the system metrics, this work presents specific metrics to evaluate the module under discussion. The performance of a data cleaning module is measured by the quality of its output. As stated in Floerkemeier *et al.* (2007) that characterized the RFID middleware constraints, the purpose of the data cleaning module is to filter and remove any noise resulting from both *false negative* and *false positive* readings. With this in mind, the number of erroneous tag transitions and the ratio between the actual tag presence time within the reading area, and the time reported by the system, are the quantities that this work is interested in. All things considered, these metrics will dictate the contribution of the module being tested for the global quality of the system.

4.1.1.1 PresenceErrors

The errors induced by the data cleaning module could have an impact in the business events. As an example, consider two different data cleaning configurations where one answers with the correct data and the other gives an over-counting report. In a warehouse where RFID technology is used to report how many times an object entered the facility, a reporting problem may exist and the inventory levels.

An RFID system is expected to be accurate and reliable. However, problems could appear like the one described above. The metric $PrsenceErrors$ can detect this kind of problems.

The metric measures the difference between the number of times the object entered in the reading area and its expected number of times, as defined in the following equation:

$$PresenceErrors = \mid NumberTimesReported - NumberTimesExpected \mid \tag{4.1}$$

A number of 0 errors should be the ultimate metric goal, whereas a different value necessarily means that some error did occurred.

For this work, it is important to highlight here that although this metric is similar to the one used in the Jeffery *et al.* (2006) work, it differs on how it is extracted and what it is evaluating. In Jeffery's work solely the algorithm is being tested, while here the $PresenceErrors$ evaluates the whole system.

4.1.1.2 PresenceRate

A more precise metric is the $PresenceRate$ where the number of errors are not taken into account, but instead is considered the reported time. For some businesses, the time that the object spends inside some facility is important to be accurate. For instance, the RFID technology is deployed to accurately report the time an object is exposed to some harmful environment. Hence, it is highly important to not under-estimate or over-estimate the reported time. Despite being a particular scenario, if every platform could be as precise as possible without affecting other parties, it will be highly valuable for RFID reputation to be considered a reliable and accurate technology.

The $PresenceRate$ is the ratio between the reported time spent by the object in the read area and the expected spent time, as defined in equation 4.2. Let us consider an experiment where the reported time of an object is 12 seconds and the expected time is 10 seconds. The platform is reporting that the object spent more time in the reading area than it should be. With these values, the $PresenceRate$ value will be 1.2, meaning that the reported time was 20% over the expected. On the other hand, if the reported time is 8 seconds the presence rate drops to 0.8, meaning that the time reported was 20% under the expected. The ideally value for the $PresenceRate$ is 1, meaning that the reported time is exactly the expected.

$$PresenceRate = \frac{ReportedTime}{ExpectedTime} \tag{4.2}$$

A deeper analysis on the metric behavior can show the following:

- $PresenceRate = 0$: the object was not detected by the system.
- $0 < PresenceRate < 1$: there are *false negative* readings and they were reported.
- $PresenceRate = 1$: ideal scenario where the reported time is exactly the same as expected.
- $PresenceRate > 1$: there are *false positive* readings over-estimating the time spent by the object in the read area.

4.2 Record

The second step in the proposed methodology is to record a set of physical repetitions from an experiment. To record the most precise sequence of events as possible, only part of the testbed components will be used. The grayed out parts in the Figure 4.19 are installed with the minimal configurations. The active components are: *a*) the robot controller in the Dashboard to program the robot; *b*) MQTT server to send and receive messages; *c*) the robot to execute the experiment; *d*) RFID reader and its antennas to detect the tagged object; and *e*) FCServer with the Rec&Play library active to receive and record all the *LLRP* messages sent by the reader.

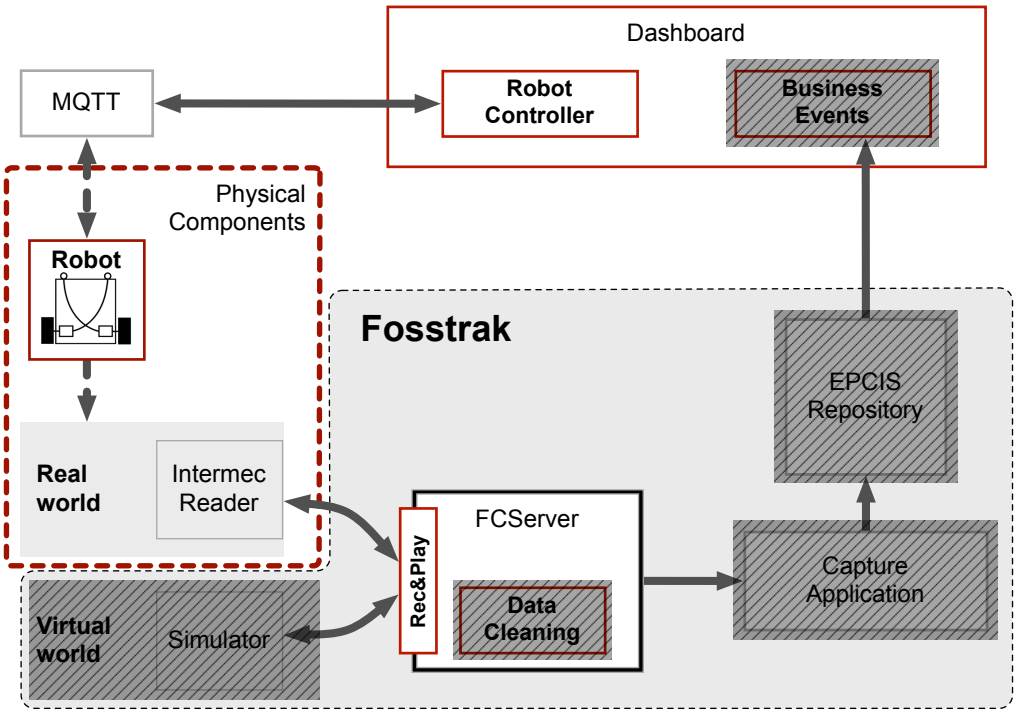


Figure 4.19: Testbed record phase with deactivated components (grayed out).

In this phase the FCServer component must be configured with the *LogicalReaders* so that both antennas could be active and connected. The Rec&Play mode in the *LogicalReader* must be set to “record” (section 3.1.1.1). Additionally, an empty *EventCycle* must be configured to activate the *LogicalReader*, otherwise the *LLRP* messages will not be received by the FCServer and consequently by the Rec&Play library.

The sequence of actions in Procedure 4.1 are the steps in the record phase that need to be repeated for every experiment. The number of times the procedure’s repeat block should be run is defined by the metric value and the acceptable margin of error. Appendix A explains in detail how to calculate the margin of error, as well as the number of repetitions needed.

The metric that must be collected in this phase is the denominator $ExpectedTime$ of the metric $PresenceRate$ (Equation 4.2). So, for each repetition the value could be tested to see if the margin of error obtained for the samples collected is below an acceptable value. Otherwise, more repetitions will be needed to reduce the margin of error.

Procedure 4.1: Testbed record procedures.

Input: γ := desired margin of error;
Output: $sessions$:= recorded sessions;
$\varepsilon = \infty$; $sessions$ = [];
repeat
  clean all logs;
  power on RFID reader;
  start application container;
  configure FCServer;
  program Robot;
  $sessions[]$ = wait and collect session;
  collect all logs;
  ε = calculate margin of error ($sessions$);
until $\varepsilon \geq \gamma$;

Record Deliverables. Each record session will produce a set of deliverables that are archived for later use. Namely:

- Log files from each testbed component to allow a detailed debug if needed.
- One recorded session (output of the Rec&Play library) per each *LogicalReader* configured.
- Metric $ExpectedTime$ for each reading area, as being the difference between the entrance and exit *flag point* obtained from the robot logs.

For each recording session a value for the metric $ExpectedTime$ is found. After a set of sessions, this value can be summarized following the Appendix A. What is obtained will be a sample

mean and then a population mean estimation with a certain confidence level. For example, a metric value of $1.2 \pm 0.3(25\%)$ seconds should be read in the following way: "the value of the metric $ExpectedTime$ is 1.2 seconds with a maximum deviation of 0.3 seconds which corresponds to an error of a 25%, with a confidence level of a 95%". Following the statistical theorems, confidence level means that for a 95% there is still a 5% of probability where the time spent in the reading area could be outside of the margin obtained of 1.2 ± 0.3 seconds.

At this point, there are a set of recorded sessions that represent a sample of the infinite possible sessions that could exist. Each session is representative of the population with a specific margin of error, meaning that any of the recorded sessions can represent the experiment with the defined confidence level.

4.3 Replay

In the third step, previous recorded sessions will be replayed to stress various software configurations. Figure 4.20 shows the active components, which namely are: *a*) FCServer with the Rec&Play library active to replay a session where *LLRP* messages are delivered to the configured *LogicalReader*; *b*) the data cleaning module singularly configured; *c*) the *CaptureApplication* configured for receiving an *EventCycleReport* and forwarding it to the *EPCIS Repository*; and *d*) the *EPCIS Repository* to store business events.

For each data cleaning configuration, a number of repetitions should be done to achieve a statistically valid experiment. This way, each configuration should be tested several times where its input is a random recorded session to guarantee the results will not be biased by a particular session. Section A.3 in Appendix A details how a session can be randomly picked. Since the recorded sessions are statistically representative of the entire experiment, any of the sessions could be used to represent the experiment.

The sequence of actions in Procedure 4.2 are the steps in the replay phase that should be repeated for each configuration. Like in the record phase, the number of times the procedure's repeat block should be run is defined by the metric value and the acceptable margin of error. The value $ReportedTime$ must be obtained in this phase to calculate the metric $PresenceRate$ (Equation 4.2), and its value could be inspected in each repetition to check if the margin of error is below an desired value. Else, more repetitions will be needed to reduce the margin of error.

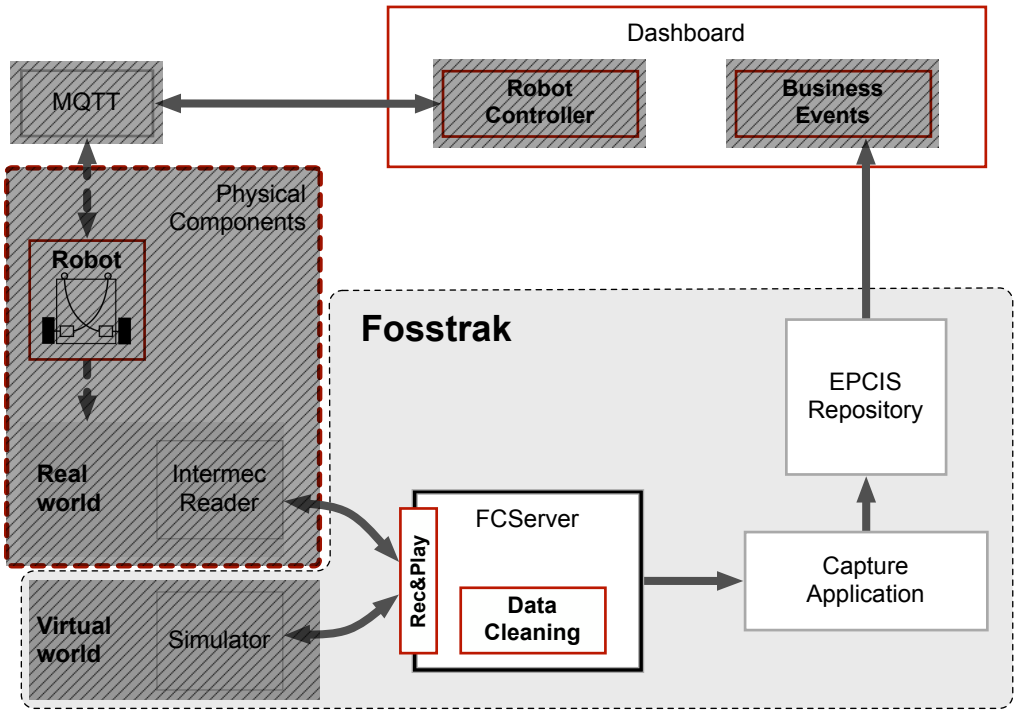


Figure 4.20: Testbed replay phase with deactivated components (grayed out).

Replay Deliverables. Each replayed session will produce a set of deliverables. Namely:

- Log files from each testbed component to carefully debug if needed.
- System metrics and data cleaning metrics ($NumberTimesReported$ and $ReportedTime$) obtained from components logs and from the *EPCIS Repository*.

After a set of repetitions taken, both metrics can be summarized as detailed in Appendix A. For a predefined confidence level, the population mean estimation for the $ReportedTime$ will be, for example, $1.2 \pm 0.3(25\%)$ seconds meaning that its estimation with a confidence level of 95% is to be 1.2 seconds with an maximum deviation of 0.3 seconds with 95% of confidence level.

With both record and replay phases, equations 4.1 and 4.2 can now be used to calculate the data cleaning metrics.

4.4 Results Analysis

The last step of an experiment is the results processing to draw some conclusions. It will be concluded if the initial goals were achieved. In particular, the metric $PresenceRate$ (Equation 4.2) can now be calculated to each reading area.

Procedure 4.2: Testbed replay procedures.

Input: γ := desired margin of error; $sessions$:= replay sessions;
Output:
$\varepsilon = \infty$;
repeat
  i = choose a session randomly;
  clean all logs;
  start application container;
  configure FCServer;
  wait for "stop session" signal;
  collect all logs;
  $metrics$ = obtain metrics' values from EPCIS database;
  ε = calculate margin of error ($metrics$);
until $\varepsilon \geq \gamma$;

After the record and replay phase, the final metrics could be calculated. All values binded with a confidence level must be take into account that its operations are not regular additive or multiplicative operations. For this math operations, Appendix A section A.1 explains how these values must be manipulated. In detail, the $PresenceRate$ calculation involves a fraction of two value estimations.

4.5 Summary and Discussion

In this chapter a methodology was proposed to correctly use the testbed introduced in Chapter 3 so that the experiments can be correctly repeated by other researchers.

A correct and precise execution of any experiment is essential to obtain meaningful results, especially in environments with such variability as RFID. These experiment steps enable an accurate comparison between results as well as an exchange of results and conclusions among the research community. The record phase is essential to correctly understand how an experiment performs. More precisely, the RFID natural unpredictability is transformed into a deterministic sequence of events where several configurations or alternatives can be tested in the replay phase. Hence, a concise methodology was introduced in the replay phase to guarantee the results are not biased by an outlier experiment sample.

Chapter 5

Experiments Evaluation

The existing Fosstrak data cleaning strategy (section 2.3.2.1) was evaluated and compared against the SMURF algorithm (section 2.3.2.2) in terms of precision and accuracy. The number of errors introduced by the system was analyzed and the system reported time was compared with the expected time. Further, these two evaluation components were discussed using the metrics $PresenceErrors$ and $PresenceRate$ defined in Chapter 4.

The first section of this chapter details the setup in which the experiments were ran as well as the parameters used. Section 5.2 details the experiment done with the individual results and conclusions. Finnaly, section 5.3 covers the overall conclusions.

5.1 Experiments Definition

5.1.1 Experiments Setup

All the experiments were conducted in a virtual machine with a $2.4GHz$ dual-core processor with $2Gb$ of RAM and running the *Linux Ubuntu 12.04.2 LTS* operating system. Regarding the RFID hardware, an *Intermec IF61 RFID* reader[1] was used connected to two *Intermec IA33F* antennas[2]. The output power of both antennas was set to $5dBm$ to restrict its operation area, and the robot (section 3.1.2), was used in all the repetitions needed to get statistically valid results. Finally, all the experiments were made on top of a wooden office desk with dimensions $160cm$ x $70cm$.

[1] Intermec IF61 reader: `http://www.intermec.com/products/rfidif61a/`
[2] Intermec IA33F antennas: `http://www.intermec.com/products/rfid_ant_ia33f/`

Regarding software components, the application container used was the *Apache Tomcat 7.0.26*[1] with *Java version* $1.7.0$ *update* 2. The RFID platform used was the Fosstrak described in section 2.2.1. The versions that were deployed, all available from Fosstrak's[2] source control system, were: *a*) *FCServer* version *1.2.1-SNAPSHOT*; *b*) *Capture Application* version *0.1.1*; and *c*) *EPCIS Repository* version *0.5.0*. Furthermore, the *EPCIS Repository* was connected to a MySQL *5.5* server to store all the EPCIS events.

Figure 5.21 shows the disposition of the physical components. Figure 5.21(a) shows the office desk that was used with the two RFID antennas facing each other. Figure 5.21(b) shows the delimiter marks used to establish the covered reading area for each antenna. Between both antennas there was a track where the robot transported a tagged object as shown in Figure 5.21(c). To facilitate the interpretation of the experiments and consequently the results, each covered area has the following designation: *i*) the area covered by the antenna on the left represents the warehouse where products can be picked and delivered; and *ii*) the area covered on the right represents the client site to receive and return products.

In all the experiments made, the robot always started on the segment I and proceeded counter-clockwise. The term *lap* refers to the movement made by the robot along all segments from segment I to segment VIII.

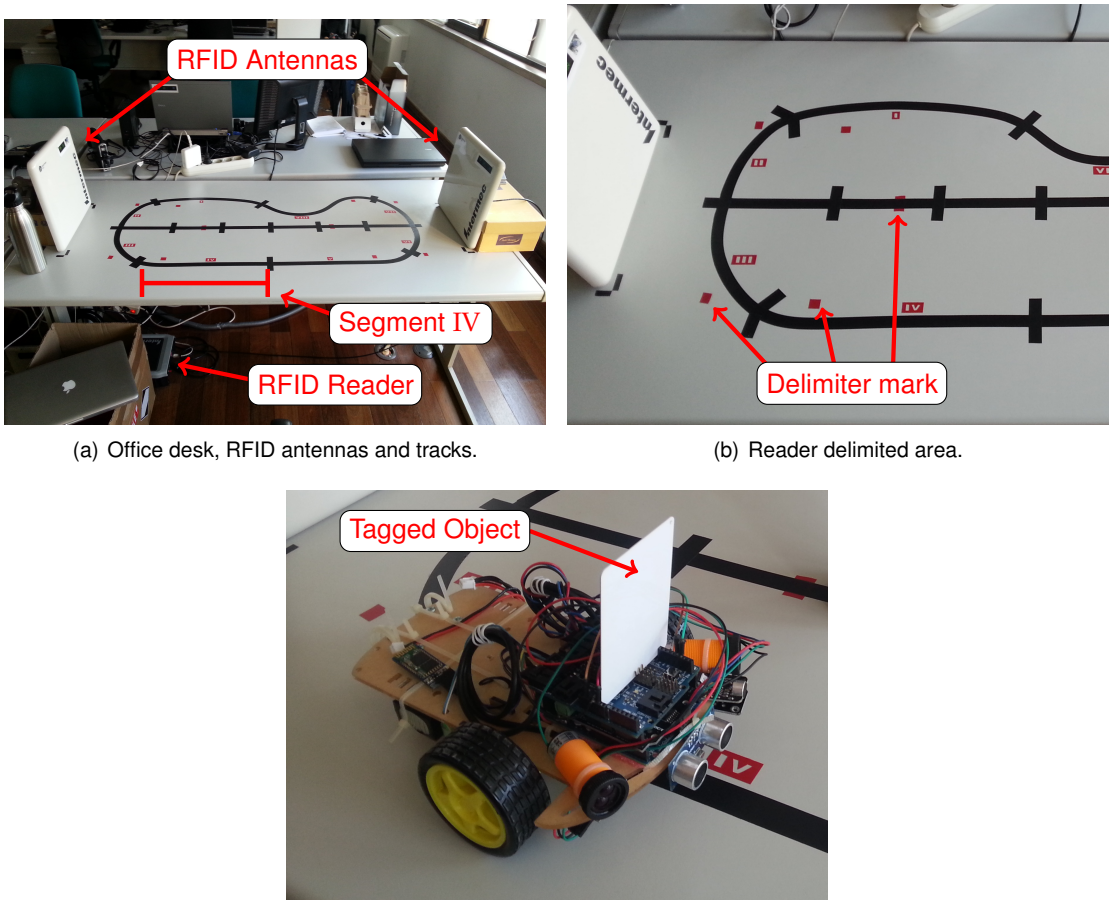
5.1.2 Record Phase Configurations

As described in section 4.2, two *LogicalReaders* were configured in the FCServer to represent both RFID antennas, the reader in the warehouse and the reader in the client area. Both *LogicalReaders* were configured with the Rec&Play mode set to "record". Additionally, one *EventCycle* was defined to activate both *LogicalReaders* to receive and record the RFID antennas events.

Only data cleaning metrics were collected in this phase, namely: *a*) $NumberTimesExpected$ used to obtain the $PresenceErrors$ metric; and *b*) $ExpectedTime$ used to calculate the metric $PresenceRate$. The maximum desirable margin of error was set to 10% to obtain a population value estimation with a confidence level of 95%. Each metric was calculated for each reading area, as well as for each lap.

As explained in the methodology section, the number of repetitions needed to have statistical significance is defined by the margin of error obtained for the metric estimation. As explained in Appendix A section A.2, it would be needed to take 96 repetitions if the desired margin of error is 10% with a confidence level of 95%. However, the number of repetitions do not need to be exactly

[1] Apache Tomcat: `http://tomcat.apache.org`
[2] Fosstrak's source control system: `https://code.google.com/p/fosstrak/source/checkout`



(a) Office desk, RFID antennas and tracks.

(b) Reader delimited area.

(c) Robot and tagged object.

Figure 5.21: Experiment setup: (a) office desk, RFID antennas and robot tracks; (b) detail of the reader covered area; (c) robot and tagged object.

96 if the margin of error obtained in the first set of repetitions is already below the desired level. This was the case for all the experiments in this work.

5.1.3 Replay Phase Configurations

Section 4.3 shows how to configure the system for the replay phase. Briefly, the first step was to switch the Rec&Play mode in the *LogicalReaders* (FCServer component) to “replay”. The *EventCycles* were both configured to report tag “ADDITIONS” and “DELETIONS” in the warehouse and client areas. Hence, when a tag is detected for the first time or when it disappears from the antennas covered area, an *ECReport* document is sent to the Capture Application.

Figure 5.22 presents the state diagram of the chosen business domain that was modeled into Capture Application rules. Succinctly, the rules implemented for all the experiments were:

- When a warehouse *ECReport* is received with an "ADDITIONS" report: an *EPCIS Document* is sent with disposition "insidewarehouse" and business step "enteringwarehouse".
- When a warehouse "DELETIONS" report is received: an *EPCIS Document* with disposition "intransit" and business step "leavingwarehouse" is sent to the *EPCIS Repository*.
- When a client *ECReport* is received with an "ADDITIONS" report: an *EPCIS Document* is sent with disposition "inclient" and business step "arrivingclient".
- When a client "DELETIONS" report is received: an *EPCIS Document* is sent with disposition "intransit" and business step "leavingclient".

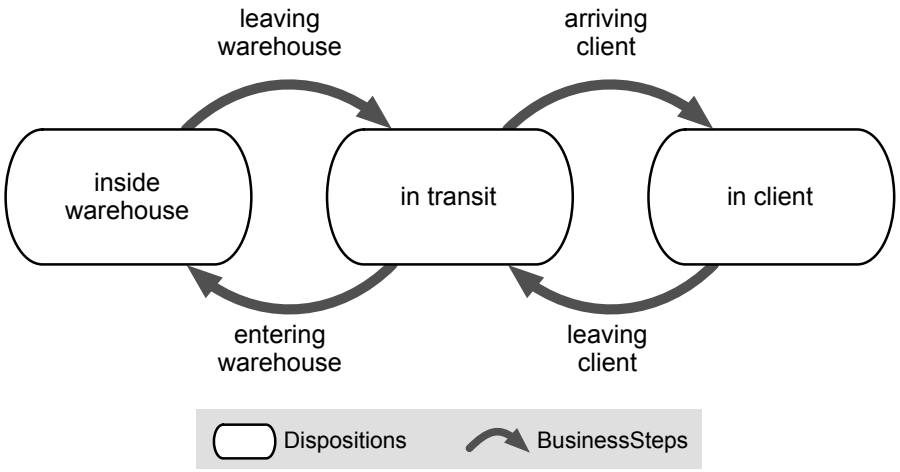


Figure 5.22: Business actions flow implemented into Capture Application rules.

In the replay phase, all the system and data cleaning metrics needed to calculate the final performance metrics were collected and summarized. Namely they were: *a*) system metrics $ProcessCPULoad$, $SystemLoadAverage$ and $MemoryUsage$; *b*) data cleaning metric $NumberTimesReported$ used to obtain the $PresenceErrors$; and *c*) $ReportedTime$ used to calculate the $PresenceRate$.

Regarding the data cleaning module configurations, a set of sliding window sizes were tested and the results compared with the SMURF implementation. The sliding window sizes chosen were $\{0.5; 1; 2; 3; 5; 8; 13; 21\}$ in seconds to cover a wider range of configurations and to avoid multiples values.

5.1.4 Research Questions

The following questions were the root of the main goals:

1. For a specific scenario, which data cleaning configuration offers the best trade-off between $PresenceErrors$ and $PresenceRate$?
2. What is the impact, in terms of system resources, of each data cleaning configuration in the system?
3. Does the alternative data cleaning SMURF algorithm perform better than the sliding window in an EPCglobal environment?
4. Given the unpredictable nature and multiple and complex configurations that characterizes an RFID system, can its performance be affected by a non-stable state?
5. What are the impacts in the data cleaning performance when interferences are introduced?

With the above questions in mind, the following experiments were chosen:

Baseline, the system without any tag was studied to obtain a baseline to make comparisons and study the impacts of the following configurations.

Track with 1 Lap, the robot does a lap in the track to study how the system reacts, starting from an empty database until it receives the firsts reads. This simple experience will allow an easy observation and a performance analysis of all the middleware components.

Track with 3 Laps, the robot does 3 laps to study if a steady state of the system affects the results. Configurations are compared only looking the the results obtained in the last lap. This way, if the non-steady state exists it can be avoided.

Track with 3 Laps with Interferences, study the impacts on the data cleaning performance by introducing a metallic object in the experiment to cause interferences. With the introduction of disturbances, this experience will depict how the different algorithms behave and to what extent the system resources are affected.

An experiment to test the type of movement towards the antennas was considered. Although it was performed, the experiment did not provide meaningful conclusions. Therefore, it was omitted and not considered in this analysis.

In the next section, the mentioned experiments are detailed in terms of what was done, what were de results as well as the main observations taken from each experiment. In section 5.3 the results and conclusions are interpreted.

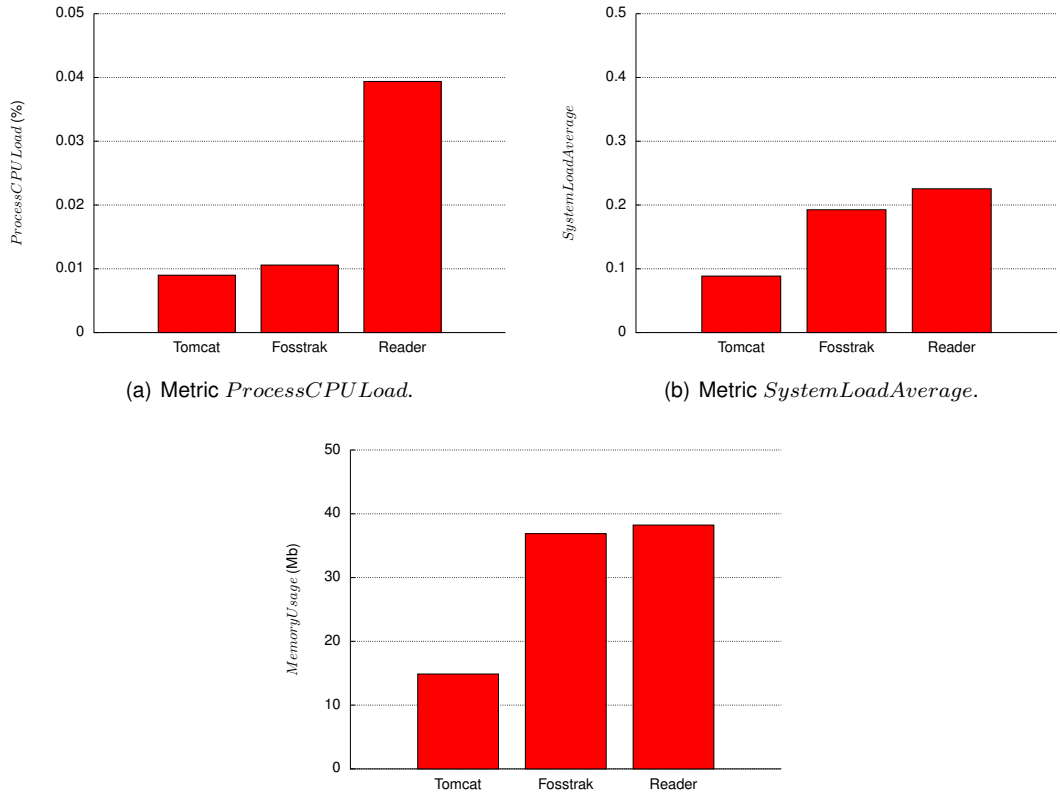
5.2 Experiments Performed

5.2.1 Baseline

A baseline allows a correct comparison in terms of usage resources. The system was studied in 3 different ways: *i*) application container without any application deployed; *ii*) Fosstrak platform deployed as it is downloaded without any RFID reader connected; and *iii*) platform connected with the two RFID antennas without any tagged object inside the read range.

The experiment was run for 5 times because the margin of error obtained was enough low to not demand further repetitions.

Figure 5.23 shows the summarized results for the system metrics. Each bar on each graph corresponds to one of the three system stages analyzed: *a*) *"Tomcat"*, Apache Tomcat running without any application; *b*) *"Fosstrak"*, application container with all the Fosstrak components deployed; and *c*) *"Reader"*, Fosstrak configured and connected to the RFID reader. Only in the last stage the system is receiving antenna events, in the other stages the platform does nothing.



(a) Metric $ProcessCPULoad$.

(b) Metric $SystemLoadAverage$.

(c) Metric $MemoryUsage$.

Figure 5.23: Baseline - System metrics results for the considered stages.

Graph 5.23(a) ($ProcessCPULoad$) shows CPU use increases when the RFID reader was connected. Precisely raised from 0.011% to 0.039%. This result is explained by the number of threads that must be active when a RFID reader is connected, that is: *i*) the *LLRP* library used to encode and decode the reader messages; *ii*) the *LogicalReaders* used to group physical readers; and *iii*) the *EventCycles* where the data cleaning module tries to eliminate any false readings even if there are no reported tags. Despite the increment of 354% on the CPU usage when the reader is connected, the value itself is low (0.039%) to consider this an application CPU intensive. Yet, this value should not be ignored when tags are on range or when multiple readers are connected because more events more demand from the CPU is expected.

Considering the metric $SystemLoadAverage$ in graph 5.23(b), the behavior of the system when the Fosstrak platform and when the RFID reader were introduced was the expected. That is, when the Fosstrak platform was deployed, the average number of processes running in the CPU or blocked waiting for an IO event (as explained in Appendix B) was 0.193, more than twice the value (0.089) when there was only Tomcat running. Also, contrasting the $SystemLoadAverage$ and $ProcessCPULoad$ it is visible that Fosstrak platform contributed with more processes running in the system but most of the time they were blocked. In contrast, the introduction of the RFID reader did not significantly affect the $SystemLoadAverage$, yet it increased the CPU utilization.

Regarding memory usage patterns, graph 5.23(c) shows what was expected when as more objects were allocated when the Fosstrak platform was deployed. In average, the introduction of the RFID reader did not significantly increased the memory usage. However, more variability is expected because there are more objects being allocated and deallocated.

Different system stages were profiled to allow an analysis in terms of usage resources enabling a comparison between different configurations and data cleaning strategies in terms of system resources usage.

5.2.2 Track 1 Lap

In this experiment the robot did a complete lap going from the warehouse area to the client area both specified in section 3.1.2.1. The questions behind this experiment are: which is the best configuration and what are the impacts in terms of resource allocation?

The robot was programmed to complete a lap with the following actions: *i*) *follow the line* until the *flagged point* to indicate the entrance of an reading area; *ii*) *go over the segment delimitation*; *iii*) *flag* action to report to the Dashboard module the passage over an *flagged point*; *iv*) *follow the line* until the next stop; *v*) *go over the segment delimitation*; *vi*) *pause for 5 seconds*; and finally *vii*) *repeat the previous actions 4 times* to go over all segments in order to complete a lap.

This experiment had two stages. In the first one the results were analyzed from the recorded sessions and components logs. In the second one, some middleware configurations were changed to solve issues identified in the first stage.

5.2.2.1 Stage 1

After the first set of repetitions, two possible improvements were detected related to the RFID platform modules, and the data cleaning configurations. In detail, the issues found in the Fosstrak platform were related with the startup of some modules and libraries. When the *Capture Application* receives an *ECPReport* for the first time it takes around 5 seconds to load the business rules. After the first request the delay was no longer observed. So, the testbed was changed to send an empty report at the beginning of the experiment to force the loading of the rules. Further, there was another problem related with the loading of the *Tag Data Translation* (TDT) library used in the *FCServer* component that takes around 200 milliseconds to load. Despite being an issue with an easy solution or workaround, the library was left intact since it is part of the module being evaluated (data cleaning) and not of an external component.

The other improvement detected was directly linked to the data cleaning configurations. In the *ECSpec* document, where the *EventCycles* are configured, its behavior is configured using two values: *repetition time* of the reading cycle, and the *duration time* in which the reading cycle remains active. At the end of a *duration time*, all the tags collected in the event cycle are reported to the *Capture Application*. When the interval between the stop trigger and the beginning of the next event cycle is short, the next event cycle fails to start and therefore the start trigger will only happen in the next period. Consequently, there is a period of time between the end of the event cycle and the next one proportional to the period interval. Figure 5.24 represents a failed start trigger and its consequences.

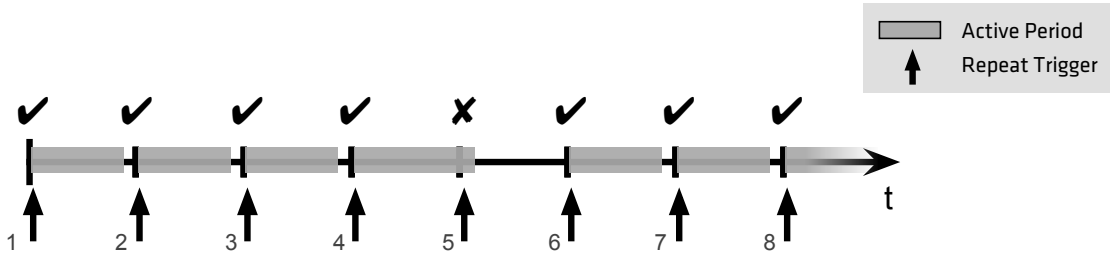


Figure 5.24: The 5th *EventCycle* trigger fails to start because the previous active period was still active.

These insights were reflected in the sliding window configuration in order to have a higher margin between the end of the active period and the start of the next one.

Stage 1 Results. The results obtained before the fine-tunning of the system are depicted in Figure 5.25. Firsts impressions when comparing it with Figure 5.27 is the positive correlation between $PresenceRate$ and the sliding window size. The higher the window size, the higher is the $PresenceRate$. This behavior can be explained by how sliding window works (in section 2.3.2.1) and by the insights gathered in this experiment. The ideal value for the $PresenceError$ is 0 and for $PresenceRate$ is 1. A value below 1 means there are *false negative* events, and above 1 there are *false positive* events. As far it gets away from 1 worse is the result.

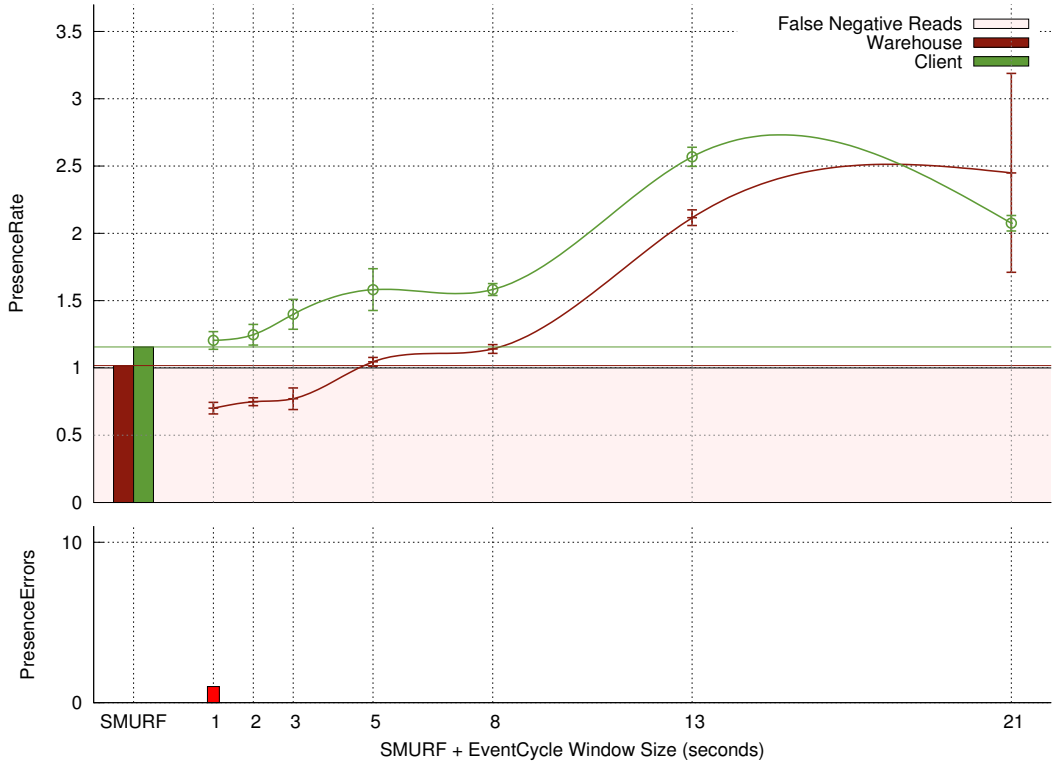


Figure 5.25: Experiment Track 1 Lap data cleaning results before the fine-tuning. The graph shows the $PresenceRate$ metric on top and the $PresenceErrors$ on the bottom where it shows the number of errors perceived by the *Capture Application*.

Analyzing only the first stage (Figure 5.25), it is possible to depict the constant difference between the $PresenceRate$ on both covered areas where the time reported by the system for the warehouse presence was always lower than the client area. This reflects the issue with the delay introduced when modules are starting up which directly affected the warehouse $PresenceRate$ because the robot always started from segment I towards the warehouse, the first reading area. It can be concluded that the system reported *false negative* readings for the warehouse was caused by the *Capture Application* and not by the data cleaning module.

From the log analysis was possible to depict the other delay introduced by the *TDT* module

when it is used for the first time. Again, it was when warehouse's *ECReports* were generated that this delay was induced and so the next *EventCycle* failed to start as explained in the experiment definition section.

The system performance was only studied in the second stage to avoid the identified issues.

5.2.2.2 Stage 2

After all configurations were fine-tuned, more sliding window sizes were added in this second stage in order to have a better understanding of the data cleaning performance behavior. The sliding window sizes added were $\{0.3; 4; 6; 7; 9; 10; 11; 12; 15; 30\}$ seconds.

Stage 2 Results. Figure 5.26 presents the system metrics values for the current experiment after the fine-tunning. The graphs shows the difference between all sliding window configurations and the SMURF algorithm represented with a bar on the left side of the graph. Generally, the values for all system metrics are stable independently from the size of the sliding window. Although, for extremely small windows, such as 0.3 seconds, all system values are higher due to the greater activity in terms of threads and objects being used. SMURF presented higher values when compared with the static windows being expected since it necessarily involves more computation to make all the decisions needed to estimate the tag presence.

Figure 5.27 shows the data cleaning metrics results for the second stage after the fine-tuning. *False negative* readings are no longer visible except for very small window sizes where the duration of the window to receive new events is so small that the system is always saying the tagged object entered and exit the covered area.

It is possible to depict a significant drop in the warehouse $PresenceRate$ for the sliding window of 13 seconds. This value is correct due to a coincidence of having all the warehouse antenna events in a single sliding window. In detail, the estimated $ExpectedTime$ for the tagged object inside the warehouse was $10.459 \pm 0.282(2.70\%)$ seconds and all the antenna events happened between the start and the end of an *EventCycle* of 13 seconds. As a result, the $PresenceRate$ was $1.20 \pm 0.03(2.82\%)$, meaning that the reported time was 20% more than the expected. On the other hand, the client $PresenteRate$ value for the same sliding window size kept the increasing behavior because the antenna events happened in two different *EventCycles*, therefore the $PresenceRate$ was higher. The same effect can be depicted with the sliding window of 21 seconds where both antennas' events happened inside a unique sliding window and so the values of $PresenceRate$ were $1.96 \pm 0.05(2.72\%)$ for the warehouse and $2.06 \pm 0.06(2.79\%)$ for the client, which are directly linked to the window size.

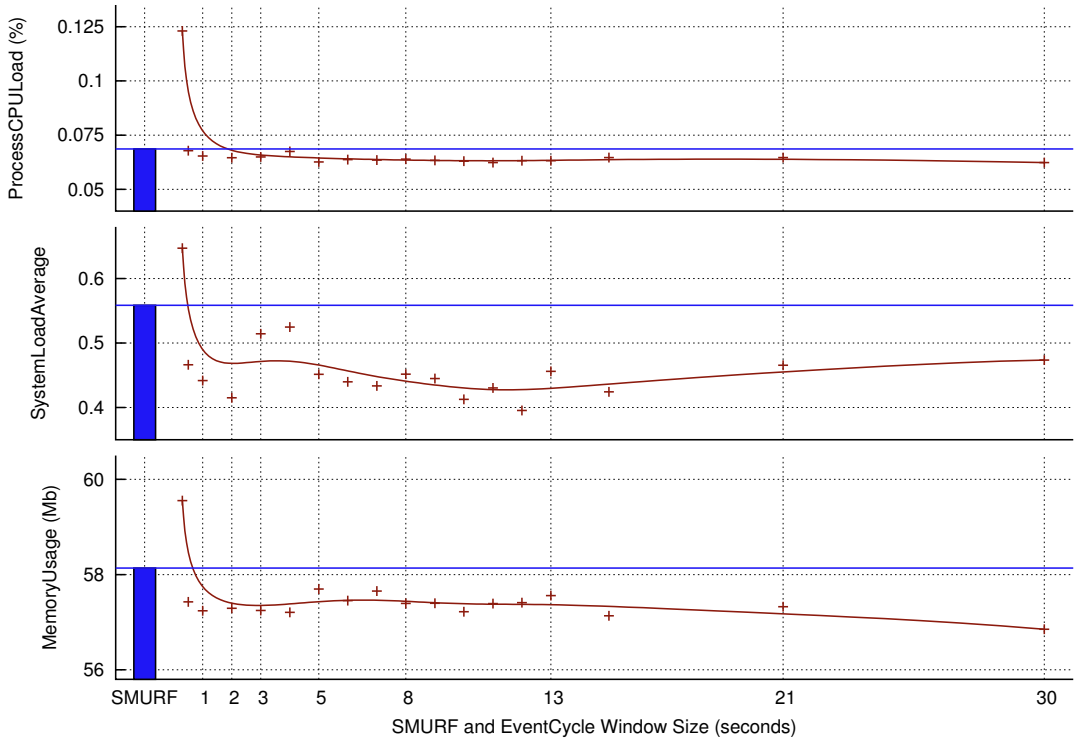


Figure 5.26: Experiment Track 1 Lap system metrics results in stage 2.

For the sliding window configuration of 30 seconds the reported client $PresenceRate$ is 0 meaning that the entrance of the tagged object in the client area was not reported by the system. This is half true since the testbed for this particular configurations did not give enough time for the system to report the exit of the tagged object from the client area. Besides, this explains the number of errors reported in $PresenceErrors$ for the sliding window of 30 seconds.

For this experiment, the SMURF data cleaning algorithm did not report any errors and its $PresenceRate$ value is close to the ideal value of 1, it is $1.02 \pm 0.06 (6.17\%)$ and $1.16 \pm 0.05 (4.41\%)$ for the warehouse and client, respectively. So, in terms of performance it sits between the static sliding-windows of 0.5 (with 1 error detected) and 1 seconds without any identified error.

To summarize, this experiment showed that an RFID system could have a poor performance not only because of the platform or software problems, but also its configuration. Platform and configurations problems were identified and solutions pointed out to improve the system.

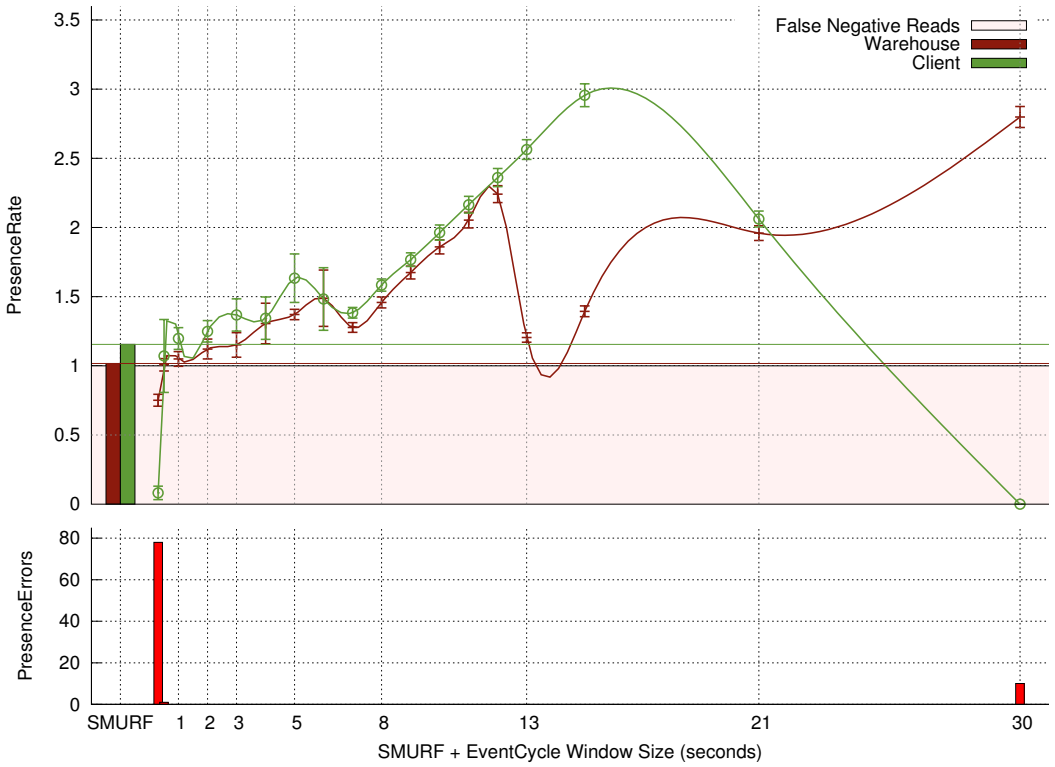


Figure 5.27: Experiment Track 1 Lap data cleaning metrics after the fine-tuning.

5.2.3 Track 3 Laps

This experiment is similar to the previous one but with a different number of laps: the robot did 3 complete laps rather than just 1. In this experiment the goal to achieve was overcoming the initialization problems found in the previous experiment. The questions this experiment tried to answer were which is the best data cleaning configuration and if the Fosstrak as an RFID system can be affected by a non-steady state phase. Also, the adjustments identified in the previous experiment were reflected into the current experiment in order to improve the final results.

Figure 5.28 shows the system metrics $ProcessCPULoad$, $SystemLoadAverage$ and the $MemoryUsage$ for the complete experiment. Comparing the values obtained in this experiment with the values in the previous experiment, the behavior is similar with all the values of sliding windows configurations stable. The SMURF data cleaning algorithm had a lower value in the current experiment but without a significant difference for the rest of data cleaning configurations.

The other system metrics analyzed showed the same behavior when comparing with the previous experiment but within a different range of values. The metric $MemoryUsage$ value for all sliding window configurations was around 64.17 Mb, while in the previous experiment with

just one lap the value was 54.45 Mb. It is evident that the usage of memory increased and, considering that the time analyzed is 3 times more than in the previous experiment, a memory leak in the whole system may exist. However, with the presented graph it is not possible to consolidate this observation.

Regarding the SMURF algorithm the same behavior is observed in terms of memory used with 15.20 Mb more than in the last experiment with one lap (raised from 47.16 Mb to 62.36 Mb).

Contrarily, the metric $SystemLoadAverage$ present lower values than in the previous experiment. This fact is associated with the duration of the experiment and the normal behavior of an exponential moving average where its decaying behavior tends to a certain value along the time.

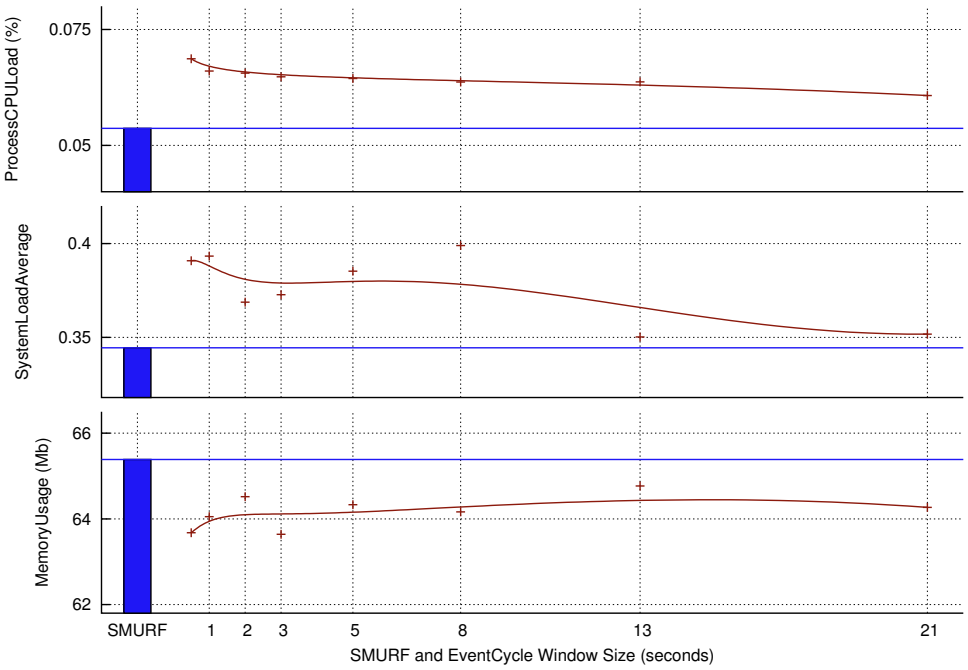


Figure 5.28: Experiment Track 3 Lap system metrics results.

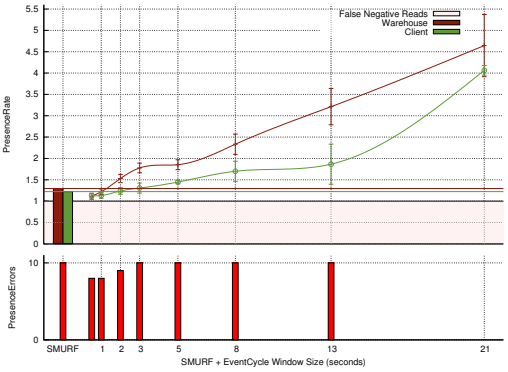
To analyze the data cleaning performance metrics $PresenceErrors$ and $PresenceRate$ the values for the 3 laps were separated as Figure 5.29 shows. In general terms, the same behavior is perceived: the $PresenceRate$ increases with the size of the sliding window. The major difference between both experiments is the margin of error that is bigger in the this experiment. This behavior in the results is because the variability induced by the antenna events, that did not fit in a unique sliding window. Besides, it took more time to execute the whole experiment, hence more variability was introduced to conditions of execution.

In Figure 5.29 it is visible that, regarding $PresenceErrors$, there are errors reported in the first lap for all configurations except for the sliding window of 21 seconds. This analysis will be separated in two parts. Firstly, why errors happened? Secondly, why do they appear in the results? On the beginning of almost all the experiments physically taken, there was a tag detection by one of the RFID antennas, even with the tagged object was outside the both covered areas. At that time it was not noticeable but when the results were extracted, the number of errors in the first lap appeared. Since the power of the antennas, to reduce the covered area, is controlled by the reader, for some unknown and undocumented reason when the antennas are turned on, a spike of energy appears to be emitted and energizes the tag, being detected by the antenna. The sliding window with 21 seconds in the first lap is the only data cleaning configuration that was able to eliminate this false event, even SMURF was not able to ignore it. As expected, this erroneous event was no longer present in the second and third laps.

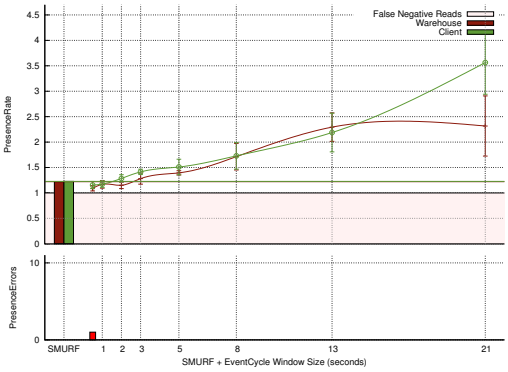
SMURF, besides it was not able to eliminate the first erroneous event, maintained a good performance in terms of $PresenceRate$. The worst value observed was in the warehouse area in first lap with $1.29 \pm 0.05(3.59\%)$, while in the third lap it showed a value of $1.23 \pm 0.06(4.91\%)$ for the warehouse and $1.24 \pm 0.08(6.41\%)$ for the client covered area. In the last lap, the SMURF algorithm is comparable with the sliding window of 1 and 2 seconds.

The best $PresenceRate$ value was observed in the last lap by the sliding window of 0.5 seconds. Two aspects should be considered: first, there were perceived 3 errors in 10 repetitions; and second, it presented a higher margin of error comparing to the others configurations (23.69% vs. a mean of 12.10%). To better understand this behavior and also to reduce the margin of error more repetitions should be done with this particular configuration.

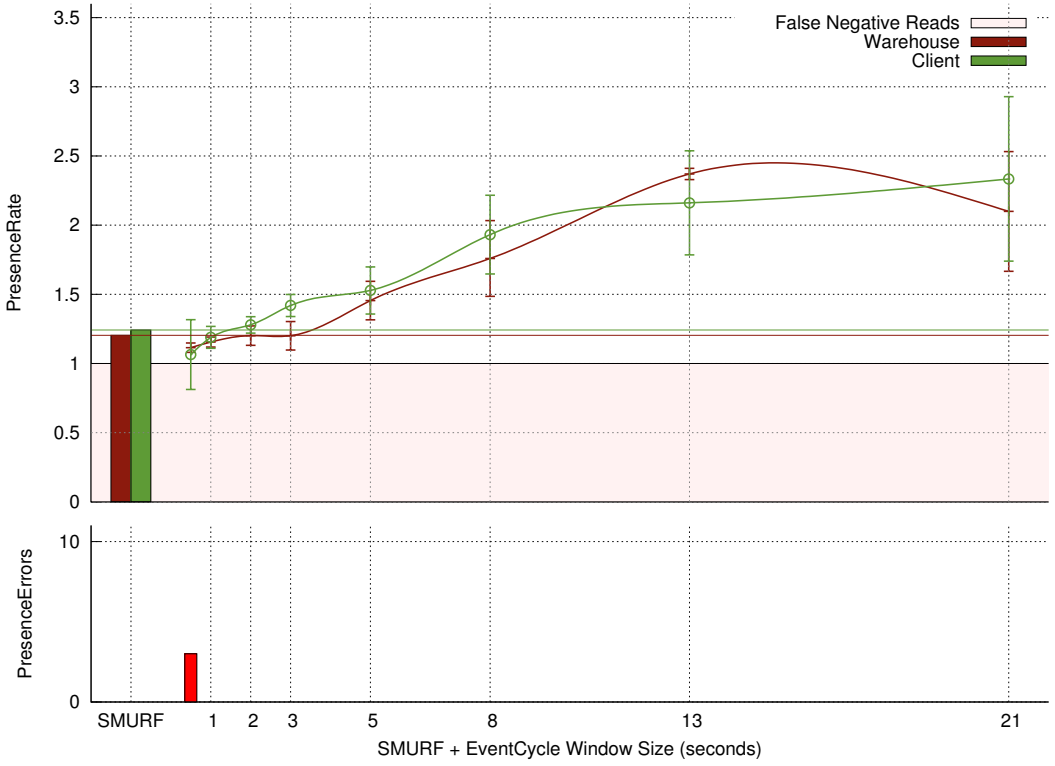
To summarize, in terms of erroneous readings there are data cleaning configurations that are capable of eliminating such events. The number of *false negative* readings however, is higher when trying to reduce errors being a penalty in terms of $PresenceRate$. For very small window size there is the risk of error occurrence even in a steady state.



(a) Experiment lap 1.



(b) Experiment lap 2.



(c) Experiment lap 3.

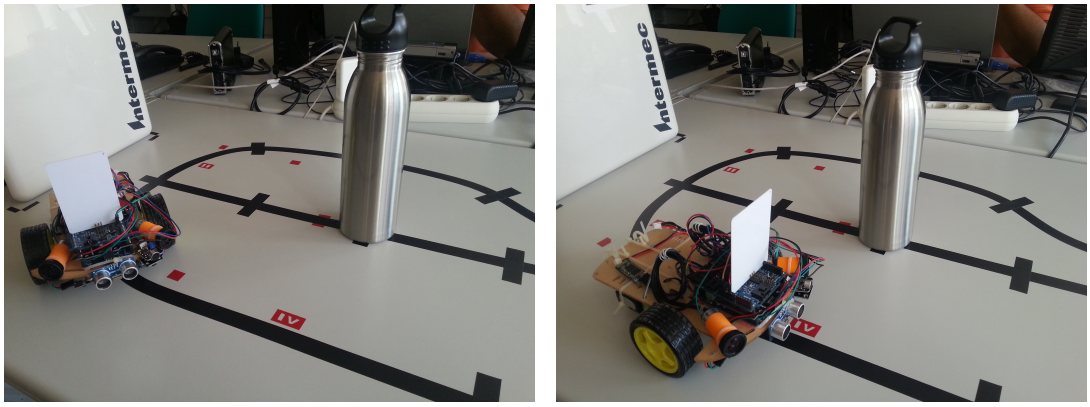
Figure 5.29: Experiment Track 3 Lap data cleaning metrics for the three laps.

5.2.4 Track 3 Laps with Interferences

In the this experiment an object was introduced in the warehouse to cause interferences in order to analyze what would be its impact. In other words, how efficient is the data cleaning module to eliminate such interferences.

A metallic object was placed in the scenario to introduce disturbances in the raw events of the RFID antenna. This was obtained by placing a metallic bottle in front of the warehouse's antenna as depicted in Figure 5.30. Consequently, the warehouse's covered area changed:

- Destructive interferences that partly occluded segment III reducing the warehouse's covered area (Figure 5.30(a)).
- Constructive interferences that partly included segment IV in the warehouse's covered area (Figure 5.30(b)).



(a) Segment III with destructive interferences. (b) Segment IV with constructive interferences.

Figure 5.30: Disturbances in the reader covered area by introducing a metallic object.

The warehouse area with the introduction of the metallic object suffered two types of false readings. On one hand, *false negative* readings by the occlusion of segment III. On the other, *false positive* readings by the inclusion of the segment IV in the covered area. To this, should be noted that the interferences were caused by an object placed outside the covered areas.

Regarding the system metrics, the observed values in this experiment followed the same behavior as in the previous experiment without interferences. The biggest, but still small to be considered significant, difference between the observed values was in the $SystemLoadAverage$ where in the sliding window configuration of 0.5 seconds was observed an increase of 0.06, going from 0.39 to 0.45. Thus, the introduction of interferences seems to not significantly affect the system in terms of resources usage.

Figure 5.31 presents the data cleaning metrics obtained for this experiment. It is visible that the introduction of disturbances had some effects. Only the third lap will be analyzed since it is the one with the most stable results.

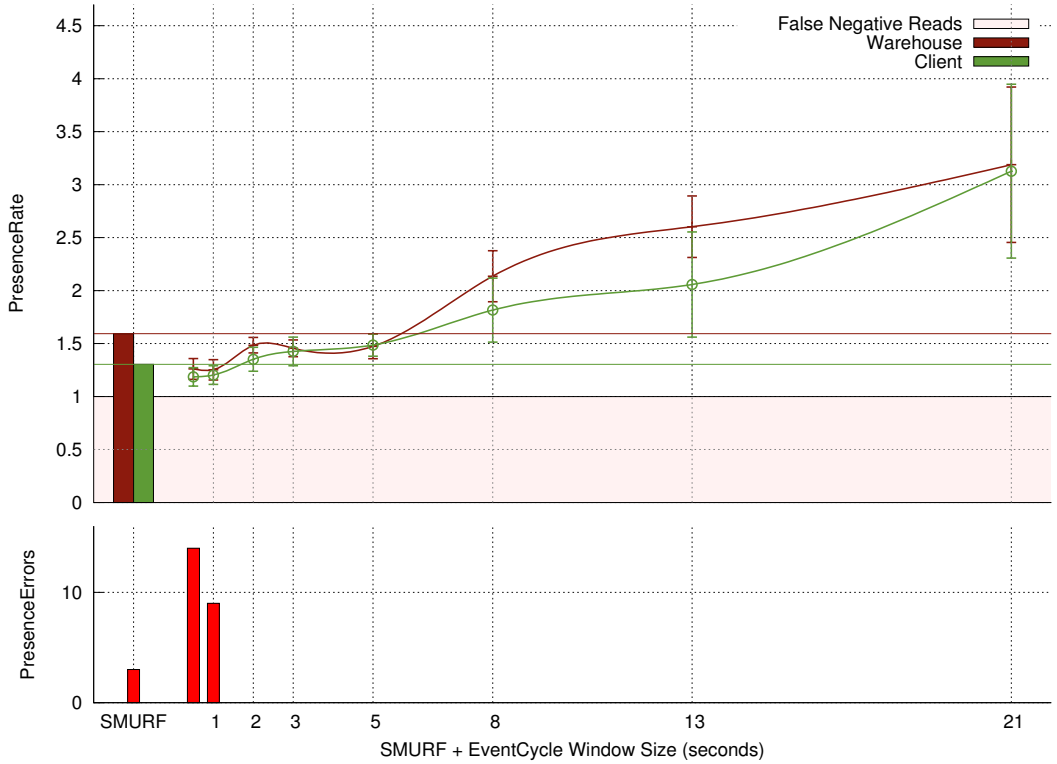


Figure 5.31: Experiment Track 3 Lap with Interferences data cleaning results.

It is visible from the graph 5.31 that erroneous reads exist for lower sliding windows such as 0.5 and 1 seconds. The other sliding window configurations were capable to eliminate all the erroneous reads resulting from the introduction of interferences. The reason why errors are observed only for lower sliding windows is because the time between the disappearance of the tagged object and its appearance is relatively short and so, bigger window sizes are capable to ignore such transitions. This follows the *completeness* and *tag dynamics* properties described in the introduction Chapter 1.

Analyzing the behavior of the SMURF algorithm proofs that it was unable to eliminate all the errors. In detail, in 10 repetitions of the experiment the algorithm erroneously reported 3 times the entrance and exit of the tagged object. However, the effort to reduce the number of errors had a significant impact in the warehouse $PresenceRate$, where its value was $1.59 \pm 0.23 (14.37\%)$ while in the experiment without interferences was $1.20 \pm 0.04 (3.01\%)$. First, it has a much higher percentage of error and second it reported more 39% of *false positive* readings.

The $PresenceRate$ for the client area remained within the same values as in the previous experiment. Summarizing, the SMURF algorithm was able to reduce the number of errors but with a trade-off for the $PresenceRate$ value.

5.3 Results Interpretation and Discussion

Based on the results obtained, it is possible to choose which is the best configuration for the data cleaning module for every each scenario. However, there is a clear trade-off between accuracy ($PresenceErrors$) and precision ($PresenceRate$) that must be take into account. For instance, the configurations that offered best values for the metric $PresenceRate$ in experiment *Track with 1 Lap* was the sliding window configuration of 0.5 seconds with $1.01 \pm 0.04(4.45\%)$ but in 10 repetitions 1 error was identified.

Table 5.1 resumes the best configuration for each experiment with no errors reported and the $PresenceRate$ closest to the ideal value of 1. Only steady state stages were considered, that is, the results obtained after the fine-tuning process and in the last laps.

Experiment	Warehouse		Client	
Track 1 Lap	[SMURF]	$1.02 \pm 0.06(6.17\%)$	[SMURF]	$1.16 \pm 0.05(4.41\%)$
Track 3 Laps	[EC 1s.]	$1.16 \pm 0.04(3.15\%)$	[EC 1s.]	$1.19 \pm 0.08(6.66\%)$
Track Interferences	[EC 3s.]	$1.46 \pm 0.08(5.41\%)$	[EC 2s.]	$1.35 \pm 0.11(8.40\%)$

Table 5.1: Best data cleaning configurations results (EC stands for *EventCycle*).

In the experiment *Track 3 Laps* erroneous reads were identified induced by an energy spike in the reader at the beginning of the experiment and only the sliding window of 21 seconds was able to eliminate these false readings. The key point is that, even for the same scenario, the best data cleaning configuration will be different because of the scenario conditions. Further, when interferences were introduced in the warehouse, the best data cleaning configuration was the sliding window of 3 seconds rather than 1 second, as it was in the experiment without interferences. Hence, the best data cleaning configuration should be considered only for these particular scenarios and should not be extrapolated to the other scenarios.

Regarding the implications of a steady state system, it is visible from all the experiments that the results are affected if the system is not stable. This condition is not only concerned to software components, since there were disturbances when the RFID antennas were powered on.

The analysis should also take into account the resources used by these configurations. It is visible from all the results that any of the event cycle configurations has similar values

($ProcessCPULoad$, $SystemLoadAverage$ and $MemoryUsage$), except for very small sliding windows like 0.3 seconds. When interferences were introduced, the resource usage increased as expected since there were more variability than in a normal situation.

SMURF Performance. The SMURF algorithm alternative showed good results in all the experiments, despite not being the best results if compared with sliding window configurations. When using static sliding windows the configuration was adjusted when the results were not satisfactory. This was not the case with the SMURF algorithm, that was able to self-adapt to the various scenarios to minimize the number of errors. Overall, the SMURF algorithm performs better because it balances the number of errors with the time reported trying to guarantee both *completeness* and *tag dynamics* introduced in the Chapter 1.

The SMURF algorithm showed a good trade-off between $PresenceErrors$ and $PresenceRate$. The most important aspect to refer is that it presents this trade-off without any manual configuration. There are some cases, however, that SMURF was not able to eliminate all the interferences, consequently, reporting errors to the *Capture Application*. This happened when the period of time in which there were no tag readings was higher. To this, the algorithm interpreted this lack of readings as a transition.

In terms of resources, SMURF showed to use more resources than the static sliding window configurations. This behavior was expected, mainly because it is a more sophisticated algorithm, hence it has more computational needs.

The SMURF algorithm demonstrates that has room for improvement, as it was observed when used in an environment with interferences, and the $PresenceRate$ value raised as a way to reduce the $PresenceErrors$. In fact, one key improvement would be the probability value associated with each tag. If this probability, used as an estimation of how present is the tag within the reading range, could be calculated with other parameters different than those used currently, the reading rate of the antenna, the precision and accuracy of the SMURF algorithm may improve.

Chapter 6

Conclusion

To demonstrate to practitioners and researchers the functioning and main benefits of RFID technology, a tool was developed to show its “toy component”, to both industrial and academic scenarios. This “toy” has serious applications and allows the development of a testbed framework with a consistent methodology for RFID system testing.

RFID systems tend to experience the natural unreliability associated with the air protocol used in the communication link antenna-tag-antenna, where the presence of interferences in this link provokes the appearance of erroneous readings at client application layers. In an industrial context this problem is typically solved deploying more tags and / or readers. However, this strategy is counter-productive for the vast majority of scenarios from an operational and cost perspective. Alternatively, it is possible to address this constraint via software, with middleware platforms that eliminate or dissipate these erroneous readings.

The present work explores RFID unreliability in detail, and provides a testbed framework to evaluate these platforms. Precisely speaking, the present work focuses in the data cleaning module to find its best configuration to improve the reliability of the whole system. Eliminating erroneous readings is a laborious task, fundamentally because there are two opposite forces to deal with when configuring the system: *completeness* and *tag dynamics*.

The RFID platform used as the testing system was the open-source EPCglobal compliant framework Fosstrak. Its data cleaning module uses static sliding window and it was compared with the SMURF algorithm that uses and interprets the RFID antenna events differently. The EPCglobal data cleaning module uses a sliding window of time to receive and report the perceived vision of the world to the client platform.

The testbed showed to be a rigorous and effective tool to evaluate an RFID system. Indeed, with the testbed and methodology proposed, it was possible to prepare and execute a set of experiments using a repeatable and consistent process. To make the testbed a reality, the Fosstrak FCServer component needed to be extended with the Rec&Play module and a redesigned data cleaning module. The Rec&Play module was of extreme importance to take physical repetitions with the same conditions. After the record phase, the module reproduces all the original sequence of events. This enables to test, with the same set of inputs, several system configurations and therefore correctly observe the results. This way, it is possible to enhance the data cleaning algorithms and compare them with alternative strategies in the exact same conditions.

Regarding the system evaluation, two data cleaning metrics were proposed in order to evaluate the accuracy and precision: $PresenceErrors$ and $PresenceRate$. The former, $PresenceErrors$, analyzes the number of false transitions of the tagged object. The latter, $PresenceRate$, analyzes the time reported by the system versus the real time the tag was in the antenna coverage area.

With the testbed and methodology proposed, it was possible to achieve meaningful results and conclusions supported with statistical confidence and validity for each physical repetition make, representing a set of possible values of the experiment where its metrics can be summarized with statistical significance.

To correctly answer the question of which is the best data cleaning configuration, a set of experiments were defined, tested and evaluated. The testbed not only tests the RFID system as a whole, but also allows to benchmark the performance of manual versus automatic configurations of sensitive system settings. It was demonstrated that depending on the scenario, the static configurations may become inadequate for the system to operate properly and start to behave erratically. The SMURF algorithm was tested as an alternative to the sliding window and was demonstrated that it is not always the best alternative when interferences are introduced in the scenario. At those situations the algorithm reduced its performance in terms of $PresenceRate$ to maintain a low number of $PresenceErrors$. However, SMURF was able to adapt its behavior accordingly to the scenario conditions trying to minimize both *false positive* and *false negative* readings without any manual reconfiguration.

Regarding the RFID scenario, three experiments were defined and a baseline to better evaluate the entire system. A track was defined to enable the robot follow a line and loop a certain number of times. Starting with one lap to understand the basic behavior of the system, followed by a three lap tests to see how the system would perform in a steady-state situation. Finally, a three lap test with the introduction of a metallic object in the scenario to induce interferences.

Based on the results, it was shown that the performance of the data cleaning module it is not just concerned to the module itself. In the experiment with only one lap there were identified issues affecting the $PresenceRate$ value, one of them was the *Capture Application* that introduced delays affecting the final value. In the experiment with 3 laps the $PresenceErrors$ was highly affected by an unnoticed behavior when the RFID antennas were powered on.

As a conclusion, the system itself can affect positively or negatively its performance. For this reason, these sources of interferences were carefully studied and controlled so the actual performance of the algorithms could be highlighted. In this work, several obstacles / interferences sources were removed, which provided clarity when analyzing the results.

6.1 Data Cleaning Module: Sliding Window vs SMURF

The best configuration found for the experiment with one lap was the SMURF algorithm that produced the lower $PresenceRate$ value without any error detected. In this experiment a fine-tuning process was done in order to improve the final results. In the first stage one of the covered areas were reporting $PresenceRate$ values below 1, meaning that the system was undercounting the time that the tagged object spent in the covered area. These anomalies were caused by middleware modules that introduced delays when used for the first time. Also, the way how the sliding windows were configured also affected the distance to the ideal $PresenceRate$ value.

In the experiments with 3 laps the configuration that showed the best performance was the sliding window of 1 second. The $PresenceRate$ using SMURF was higher but a better performance comparing with the higher sliding window sizes. When the metallic object was introduced in the experiment to provoke disturbances, the best configurations were the sliding windows with 2 and 3 seconds. The performance of the SMURF algorithm worsened because its inability to eliminate all the errors caused by the interferences.

Nevertheless, SMURF reveals to adapt to various scenarios maintaining the *completeness* and *tag dynamics* properties. This happened when the interferences were introduced in the experiment, SMURF tried to reduce the impact of the errors induced but, at the same time, slightly degraded its performance.

In the end, strategies such as SMURF tries to reduce the overhead in terms of configurations and maintenance. That is, the system can answer to disturbances in the environment without any human intervention. It is clear that static configurations do not work because the environments are not static. So, more robust and advanced techniques are needed to improve the data cleaning performance and also the perception about the RFID technology.

6.2 Contributions Summary

RFID Testbed. A flexible and configurable testbed framework where it is possible to test several scenarios in a limited physical space such as an office desk. Further, its modular nature allows testing other EPCglobal platforms using the same procedures.

Test Methodology. A methodology on how to use the testbed to correctly test several system configurations with the same set of inputs, guaranteeing the statistical significance and adequate confidence level, important to have supported and meaningful results.

Rec&Play Module. A library used in the testbed to record a session of events to later replay them maintaining the order and the time between each event as they happened in the recorded sessions. Moreover, it is not RFID specific and can be used in other domains.

Data Cleaning Metrics. Two data cleaning metrics were proposed to evaluate the system performance regarding its data cleaning module. The $PresenceErrors$ to measure the number of erroneous tag transitions, and the $PresenceRate$ where it is evaluated the efficiency of the data cleaning module to eliminate false readings.

Data Cleaning Performance Evaluation. Experiments were made in order to find the best data cleaning module configuration for a set of experiments. Besides, the SMURF algorithm was compared with the basic sliding window configuration and all the constraints found during the execution of the experiments were documented.

RFID Demonstrator - RFIDToys. The testbed can be used as a facilitator to make demonstrations of the RFID technology in an industry context as well as for teaching purposes. Moreover, the robot "toy" and the dashboard are essential parts to better convey on communicate how the technology works. To facilitate even more its use, a visual programming language was used to program the robot, making it possible for non-programmers to experiment and test new scenarios. The demonstrator has been already used in a classroom where RFID was one of the topics discussed[1].

6.3 Future Work

Experiments Performed. Regarding the experiments made, the results showed the need to test another cases to explore even more the behavior of the data cleaning algorithms and therefore the behavior of the entire system. For instance, the experiments only used one reader

[1] The class was done during a BEST summer course at IST, August 2013, with 22 students from different engineering backgrounds.

covering one area and with a limited power, the usage of two or more readers to cover the same area could be explored to better understand how the system and its performance are influenced by the reader collisions that exists by having multiple readers operating in the same area.

Data Cleaning Module. The SMURF algorithm sees the antenna events as a sample of readings of the physical world. One of the key parameters this algorithm has is a probability value associated to each tag reading to estimate if the tag is inside the reading area. More probability functions could be studied to take into account other parameters like environment conditions, tag movement information, or business related parameters. The proposed testbed and methodology can be used to test these, and others approaches, in the SMURF probability function. Further, other algorithms could be tested and evaluated following the proposed metrics and compared with other strategies.

Regarding the proposed metrics, the $PresenceRate$ can be replaced by a more intuitive and easy to read metric. For instance, using a percentage value ($< 100\%$ to indicate *false negative* readings, 100% for the ideal value and $> 100\%$ for *false positive* readings). Another option could be to center the ideal value in 0 rather than 1.

Despite not being used in the experiments, the *Intermec* RFID reader has an interface[1], available through the LLRP protocol, with enhanced tag signal information, movement information and also presence confidence. Other RFID reader vendors have other technologies that can be explored as well. This proprietary information could be used as inputs for the improvement of the SMURF algorithm.

Besides tag related information, the scenario itself could provide parameters to better estimate the tag presence. For instance, the speed parameter used to program the robot can be used to improve the algorithm results, or from the RFID system's viewpoint it could inform the robot to slow down its operation if a weak tag presence value is detected.

Alternatively, and still using the bare sliding windows, a system of parameters could be developed to find what would be the best sliding window size taken into account the conditions where the system is deployed, such as the area covered by the antennas, the number of antennas available, the speed of the tag and the need of timely answers, etc. To conclude, a function of meaningful parameters could be developed to easily find what would be the most appropriate value for a particular scenario.

[1] More information about Intermec's ARX technology can be found at `http://www.intermec.com/public-files/application-briefs/en/ab_ARX.pdf`

Rec&Play Library. The Rec&Play library could also be improved. The session files could be stored in a standardized format to allow an easy distribution of the sessions to other projects. Further, a public database of recorded sessions could be built to allow practitioners, researchers and technicians to share RFID recorded sessions globally. This way, the range of completely different scenarios will be bigger allowing more rapid advances in the technology. This public space can reduce the need of buying expensive hardware to test real world scenarios. Additionally, if these sessions were used with the presented methodology, then all the results from different research groups will obtain comparable and statistical valid results.

RFIDToys Testbed. The dashboard used in the testbed can be extended to offer more data about the system being tested. The dashboard can be more intuitive with additional features such as real-time graphs to show a metric values. Also, it could be transformed in an operational dashboard if the necessary changed were made.

From the robot's perspective, the Arduino platform offers great flexibility but with very limited resources. The running code that receives and executes the commands from the dashboard, could be improved to minimize the memory usage freeing more memory to be used in other parts such as its internal queue. This way, the robot could hold more commands and be able to hold more abstract operations such as "*go to point B and pick box with label X*". Finally, it is an interesting module to allow other academic projects, from completely different areas rather than Computer Science, to participate in this RFID Testbed module.

The present testbed could be used as a certification tool, not only for new data cleaning algorithms but also for other levels of the EPCglobal Architecture. For instance, the testbed can be extended to test if certain business logic / rules are verified when facing specific recorded sessions from the Rec&Play module.

Bibliography

AHMED, N. & RAMACHANDRAN, U. (2008). Reliable Framework for RFID Devices. In *the 5th Middleware doctoral symposium*, 1–6, ACM Press, New York, New York, USA.

AHMED, N. & RAMACHANDRAN, U. (2009). Load shedding based resource management techniques for RFID data. *RFID, 2009 IEEE International Conference on*, 306–313.

BENDAVID, Y., LEFEBVRE, É., LEFEBVRE, L.A. & FOSSO-WAMBA, S. (2008). Key performance indicators for the evaluation of RFID-enabled B-to-B e-commerce applications: the case of a five-layer supply chain. *Information Systems and e-Business Management*, **7**, 1–20.

BOLIC, M., SIMPLOT-RYL, D. & STOJMENOVIC, I. (2010). RFID systems: research trends and challenges.

CAO, Z., SUTTON, C., DIAO, Y. & SHENOY, P. (2011). Distributed inference and query processing for RFID tracking and monitoring. *Proceedings of the VLDB Endowment*, **4**, 326–337.

CHEN, H., KU, W.S., WANG, H. & SUN, M.T. (2010). Leveraging spatio-temporal redundancy for RFID data cleansing. 51–62.

CLARKE, R.H., TWEDE, D., TAZELAAR, J.R. & BOYER, K.K. (2006). Radio frequency identification (RFID) performance: the effect of tag orientation and package contents. *Packaging Technology and Science*, **19**, 45–54.

DARCY, P., STANTIC, B. & SATTAR, A. (2009). A fusion of data analysis and non-monotonic reasoning to restore missed RFID readings. *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2009 5th International Conference on*, 313–318.

DOUGLAS, C.M. & GEORGE, C.R. (2003). Applied statistics and probability for engineers. *John Wiley & Sons, Inc*, **605**, 506–564.

EPCGLOBAL (2007). EPC Information Services (EPCIS) Version 1.0.1.

EPCGLOBAL (2009). The Application Level Events (ALE) Specification.

EPCGLOBAL (2010). Low Level Reader Protocol (LLRP).

EPCGLOBAL (2013). GS1 EPC Tag Data Standard. 1–226.

EVANS, M., NOBLE, J. & HOCHENBAUM, J. (2012). *Arduino in Action*. Manning.

FAN, H., WU, Q. & LIN, Y. (2012). Behavior-Based Cleaning for Unreliable RFID Data Sets. *Sensors*, **12**, 10196–10207.

FINKENZELLER, K. & MULLER, D. (2010). *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication*. John Wiley & Sons.

FLOERKEMEIER, C., RODUNER, C. & LAMPE, M. (2007). RFID Application Development With the Accada Middleware Platform. *IEEE Systems Journal*, **1**, 82–94.

FOSTER, P.R. & BURBERRY, R.A. (1999). ANTENNA PROBLEMS IN RFJD SYSTEMS. In *IEE Colloquium. RFID Technology*, 3–3, IEE.

FRITZ, G., BEROULLE, V., AKTOUF, O.E.K., NGUYEN, M.D. & HÉLY, D. (2010). RFID System On-line Testing Based on the Evaluation of the Tags Read-Error-Rate. *Journal of Electronic Testing*, **27**, 267–276.

JAKKHUPAN, W., ARCH-INT, S. & LI, Y. (2011). Business process analysis and simulation for the RFID and EPCglobal Network enabled supply chain A proof-of-concept approach. *Journal of Network and Computer Applications*, **34**, 949–957.

JEFFERY, S.R., GAROFALAKIS, M. & FRANKLIN, M.J. (2006). Adaptive cleaning for RFID data streams. *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*.

JOHNSON, R.A. & WICHERN, D.W. (2007). *Applied Multivariate Statistical Analysis*. Pearson Education International, Pearson Prentice Hall.

LIU, F. & ZHENG, L. (2012). RFID data filtering algorithm in supply chain. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, 2237–2240.

LUBY, M.G. & LUBY, M. (1996). *Pseudorandomness and Cryptographic Applications*. Princeton computer science notes, Princeton University Press.

MASSAWE, L.V., KINYUA, J.D.M. & VERMAAK, H. (2012a). Reducing False Negative Reads in RFID Data Streams Using an Adaptive Sliding-Window Approach. *Sensors*, **12**, 4187–4212.

MASSAWE, L.V., VERMAAK, H. & KINYUA, J.D.M. (2012b). An adaptive data cleaning scheme for reducing false negative reads in RFID data streams. *RFID (RFID), 2012 IEEE International Conference on*, 157–164.

MONK, S. (2010). *30 Arduino Projects for the Evil Genius*. McGraw-Hill/TAB Electronics.

MÜLLER, J., SCHAPRANOW, M., PÖPKE, C., URBAT, M., ZEIER, A. & PLATTNER, H. (2010). Best Practices for Rigorous Evaluation of RFID Software Components. *ITG-Fachbericht-RFID Systech 2010*.

NIKITIN, P.V. & RAO, K.V.S. (2008). Antennas and Propagation in UHF RFID Systems. In *RFID, 2008 IEEE International Conference on*, 277–288.

NIKITIN, P.V., RAO, K.V.S. & LAZAR, S. (2007). An Overview of Near Field UHF RFID. In *RFID, 2007. IEEE International Conference on*, 167–174.

RAHMATI, A., ZHONG, L., HILTUNEN, M. & JANA, R. (2007). Reliability techniques for RFID-based object tracking applications. 113–118.

RAND CORPORATION. (2001). *A Million Random Digits with 100,000 Normal Deviates*. RAND Corporation.

RAO, J., DORAISWAMY, S., THAKKAR, H. & COLBY, L.S. (2006). A deferred cleansing method for RFID data analytics. 175–186.

ROUSSOS, G. (2008). What is RFID. *Networked RFID*.

SCHMIDT, L., MITTON, N., SIMPLOT-RYL, D., DAGHER, R. & QUILEZ, R. (2011). DHT-based distributed ALE engine in RFID middleware. *RFID-Technologies and Applications (RFID-TA), 2011 IEEE International Conference on*, 319–326.

SIMMS, M., BECKNER, M. & VENKATESH, R. (2009). Pro RFID in BizTalk Server 2009.

TANENBAUM, A.S. (2008). *Modern Operating Systems*. Prentice Hall PTR, 3rd edn.

TEIXEIRA, P. (2012). *Hands-on Node.js*. Lean Publishing.

TRAUB, K., ARMENIO, F., BARTHEL, H., DIETRICH, P., DUKER, J., FLOERKEMEIER, C., GARRETT, J., HARRISON, M., HOGAN, B., MITSUGI, J., PREISHUBER-PFLUEGL, J., RYABOY, O., SARMA, S., SUEN, K. & WILLIAMS, J. (2010). The EPCglobal Architecture Framework.

TROTTER, M.S. & DURGIN, G.D. (2010). Survey of range improvement of commercial RFID tags with power optimized waveforms. 195–202.

TU, Y.J. & PIRAMUTHU, S. (2011). A Decision-Support Model for Filtering RFID Read Data in Supply Chains. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, **41**, 268–273.

UCKELMANN, D., HARRISON, M. & MICHAHELLES, F. (2011). An Architectural Approach Towards the Future Internet of Things. 1–24, Springer Berlin Heidelberg, Berlin, Heidelberg.

WANT, R. (2006). An introduction to RFID technology. *IEEE Pervasive Computing*, **5**, 25–33.

WU, N.C., NYSTROM, M.A., LIN, T.R. & YU, H.C. (2006). Challenges to global RFID adoption. *Technovation*, **26**, 1317–1323.

XING, L. & WEN-XIU, F.U. (2013). Efficient RFID Data Cleaning Method. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, **11**, 1707–1713–7.

ZHANG, C., LI, Y. & CHEN, Y. (2011). Application oriented data cleaning For RFID middleware. 1544–1549.

ZWILLINGER, D. (2011). *CRC Standard Mathematical Tables and Formulae, 32nd Edition*. Discrete Mathematics and Its Applications, Taylor & Francis.

Appendix A

Statistically Valid Experiments

This chapter provides the necessary background to statistically validate all the experiments. In this chapter, it will be analyzed how a metric can be summarized as an estimate of its population and also an estimation of how many repetitions are needed to guarantee a maximum margin of error for a certain confidence level.

The following sub sections are only focused on how to obtain the desired metrics estimations. Douglas & George (2003) and Johnson & Wichern (2007) can give a much more complete reference about the theory behind this chapter.

A.1 Population Mean Estimation

Let us consider an experiment which is focused in a metric $MetricA$ in seconds. Since it is not possible to survey the entire population for the experiment, a set of samples should be used to estimate the population behavior. μ is the estimated mean of $MetricA$ with a certain confidence interval CI with a confidence level of α:

$$MetricA = \mu \pm CIseconds \text{ with } \alpha \text{ of confidence.}$$

The steps needed to obtain an estimation are the following: *i*) repeat the experiment n times; *ii*) from the set of samples obtain the sample mean $\overline{x}$ and its standard deviation s; *iii*) with the sample measures and the confidence level α, the population mean μ and its confidence interval CI can be estimated; and *iv*) a margin of error ME for the estimated population mean can be calculated.

The following equations can be used to calculate the sample mean and standard deviation:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \tag{A.3a}$$

$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}} \tag{A.3b}$$

The estimation of the population mean relies on using the sample mean ($\overline{x}$) as an estimation of the population mean (μ) with a confidence interval (CI) around μ with a given probability (α). However, such estimate is only valid, and so the *Central Limit Theorem* (CLT) can be applied, if *a*) all samples are independent from each other; and *b*) all samples come from the same population with mean μ and finite standard deviation σ. If both requirements are guaranteed, the CLT can be applied and a confidence interval CI around the population mean μ can be calculated.

The formula to calculate the confidence interval will depend on the number of samples collected. The CLT states that if the number of samples is large (generally assumed to be $n \geq 30$) the sample mean $\overline{x}$ is approximately Gaussian distributed with mean $\mu = 0$ and variance $\sigma^2 = 1$, defined through the random variable Z where $z = (\overline{x} - \mu) \,/\, (\sigma/\sqrt{n})$. For a small number of samples ($n < 30$) the sample mean $\overline{x}$ follows a *Student's t*-distribution with $n - 1$ degrees of freedom and where T is the random variable with $t = (\overline{x} - \mu) \,/\, (s/\sqrt{n})$.

One of each boundaries of the confidence interval can be calculated through the following:

$$n \geq 30 : CI = \pm z_{1-\alpha/2} \frac{s}{\sqrt{n}}, \tag{A.4a}$$

$$n < 30 : CI = \pm t_{1-\alpha/2;n-1} \frac{s}{\sqrt{n}} \tag{A.4b}$$

Both values $z_{1-\alpha/2}$ and $t_{1-\alpha/2;n-1}$ can be obtained from its respective precomputed table available in Zwillinger (2011) where α is the chosen confidence level, typically 95% or 99%.

With equations A.3 and A.4, and assuming that the number of recorded repetitions of an experiment is less than 30, the metric $MetricA$ can now be calculated with the confidence level α using the following equation:

$$MetricA = \mu \pm t_{1-\alpha/2;n-1} \frac{s}{\sqrt{n}} \tag{A.5}$$

Given that all the variables of the equation are already defined, it is possible to obtain the value for the metric $MetricA$:

$$\begin{aligned}
\alpha &= 95\% \text{ of confidence} \\
n &= 10 \text{ samples} \\
\overline{x}_{MetricA} &= 12 \text{ seconds} \\
s_{MetricA} &= 0.8 \text{ seconds}
\end{aligned}$$

$$\begin{aligned}
MetricA &= 12 \pm t_{1-0.95/2;10-1}\frac{0.8}{\sqrt{10}} \text{ seconds} \\
&= 12 \pm 2.2622\frac{0.8}{\sqrt{10}} \text{ seconds} \\
&= 12 \pm 0.57(4.75\%) \text{ seconds with 95\% of confidence.}
\end{aligned}$$

The metric value should be interpreted in the following way: the metric $MetricA$ mean value is 12 ± 0.57 seconds with 95% of confidence. In other words, there is a 95% of probability that the actual distribution mean $\mu_{MetricA}$ is within the confidence interval $[11.43 - 12.57]$ seconds. Moreover, the *Margin of Error* (ME) of the estimated population mean is 4.75% with 95% of confidence.

To conclude this section, operations with confidence intervals must be considered. The present work uses a metric that is a fraction of other two, both distribution mean estimations. As consequence, there are two confidence intervals that should be divided. For the sake of completeness, additive operations are also showed.

Let us assume Z as the result of the operation between two values (A and B) for the same metric:

$$\begin{aligned}
\mu_A &= \overline{x}_A \pm CI_A \\
\mu_B &= \overline{x}_B \pm CI_B
\end{aligned}$$

For additive operations:

$$Z = \mu_A + \mu_B \iff Z = \overline{x}_A + \overline{x}_B, \quad CI_Z = \sqrt{{CI_A}^2 + {CI_B}^2}$$

$$Z = \mu_A - \mu_B \iff Z = \overline{x}_A - \overline{x}_B, \quad CI_Z = \sqrt{{CI_A}^2 + {CI_B}^2}$$

For multiplicative operations:

$$Z = \mu_A * \mu_B \iff Z = \overline{x}_A * \overline{x}_B, \quad CI_Z = \sqrt{\left(\frac{CI_A}{\overline{x}_A}\right)^2 + \left(\frac{CI_B}{\overline{x}_B}\right)^2} * Z$$

$$Z = \frac{\mu_A}{\mu_B} \iff Z = \frac{\overline{x}_A}{\overline{x}_B}, \quad CI_Z = \sqrt{\left(\frac{CI_A}{\overline{x}_A}\right)^2 + \left(\frac{CI_B}{\overline{x}_B}\right)^2} * Z$$

A.2 Number of Repetitions Estimation

The inherent margin of error in the population mean estimation will depend on the chosen confidence level and the number of samples taken. Without taking more samples, the only way to lower the margin of error is to reduce the confidence level. That is, the margin of error could be lower but its probability to be correct is also lower. So, without affecting the confidence level the only way to reduce the margin of error is to take more samples.

For some experiments there is the need to know beforehand how many samples should be taken to guarantee a certain margin of error (ME) with a specific confidence level α:

$$n = \left(\frac{z_{1-\alpha/2}}{ME}\right)^2 0.25 \tag{A.6}$$

Where: n is the number of samples; z is the $z - score$ that could be obtained from a precomputed table; and ME is the margin of error desired. For example, for an acceptable margin of

error of 10% and with a confidence level of 95%, the number of samples that should be taken is 96. However, taking such samples in RFID experiments can be impracticable. Table A.2 shows the margins of error for each sample size and desired confidence level.

	Confidence Levels		
Samples	80%	95%	99%
5	28.67%	43.82%	57.60%
10	20.27%	30.99%	40.73%
15	16.55%	25.30%	33.26%
20	14.33%	21.91%	28.80%
25	12.82%	19.60%	25.76%
96	6.54%	10.00%	13.15%

Table A.2: Maximum margin of error for different sample sizes.

From Table A.2 it is clear that for a low number of samples the margin of error is much higher. Therefore, the difference between 5 and 10 samples is the highest, so whenever possible it could be worthy to take at least 10 samples instead of just 5. Even though, having 30% of error with 95% of confidence could be problematic for some cases.

Though, one important difference should be noted, table A.2 and equation A.6 gives the maximum value possible and not the actual margin of error. In other words, it is the highest error an estimated population mean could have.

A.3 Randomization of Sessions

In the proposed methodology, the experiments should first be recorded and then replayed. The replay process involves the repetition of a randomized sequence of the recorded sessions.

As Luby & Luby (1996) and Zwillinger (2011) said, to randomly select a session, the function *random*, available in all operating systems and languages, can be used to get the random sequence. All *random* functions could be parameterized with an initial input called *seed* that is the only deterministic value of a pseudo random generator. Further, it is guaranteed that for the same *seed* the random sequence obtained will always be the same. However, the algorithm that chooses a random sequence from a specific *seed* could be different depending on the platform, it depends in what is the generator algorithm used to get the random sequence. Unfortunately, there are no easy access to what algorithm a generator is using.

And so, the generator used was based on the basic procedures used to obtain random randoms before computers era. The random sequence of recorded sessions was obtained in the following way:

- The *seed* chosen is the date when the experiment was done and a parameter value. The parameter is the window size of each data cleaning configuration in the format *<date><windowsize>*. For example, for the *seed 2013090201*, *20130902* is the date and *01* is the window size of *1 second*;
- From the chosen *seed*, a row number and a column should be obtained;
- In the book from Rand Corporation. (2001), pick the number that intersects the row and the column from the previous step;
- In the intersection point, check if the number selected is equal or below 10 (the number of recorded sessions available), if yes then pick it and go to the next row and repeat the process until the sequence of repetitions is complete;
- If the selected number is not within the available recorded sessions, it should be skipped.

As an illustration of the above instructions, for the *seed 2013090201*, the first 10 random numbers obtained for 10 recorded samples are: *05 08 03 10 08 10 01 09 06 05*. Notably, using the documented procedure it is possible to obtain a randomly sequence in a repeatable way without depending on any random function implementation.

Appendix B

System Metrics

System metrics are considered in this work to correctly evaluate the behavior and performance of the system. First, to understand and study the impacts that changes introduced in this work. Second, to profile the system in terms of resources usage, in a generic way.

To evaluate the data cleaning module, three system metrics were considered as described in the following sections. These descriptions were made from the Apache Tomcat documentation, the application container used in this work. Since it runs in a Java environment, the metrics range values follows the online documentation[1].

Detailed information about any of the proposed metrics can be found in Tanenbaum (2008).

B.1 Process CPU Load

The $ProcessCPULoad$ indicates the percentage of time the Java Virtual Machine (JVM) process was active in the available CPUs during the observed period of time. Following the JVM documentation, the possible range of values for this metric is between 0 and 1. A value of 0.0 means the JVM process was not found running in any of the available CPUs during the inquiry period. A value of 1.0 means that all CPUs were constantly running JVM processes during the analyzed period of time. Note the careful wording used, all CPUs available means that this metric already takes into account the number of available CPUs in the system. Additionally, the metric value not only counts application, specific processes and threads, but also all JVM internal threads.

[1]The Java documentation can be accessed here: `http://docs.oracle.com/javase/7/docs/api`

The number of JVM processes occupying CPU time is relevant not only to tell how the CPU is being used, but also to study the impact of each configuration. To make unbiased comparisons, all experiments should be run in the same conditions and with the same input. Altogether, it is possible to study the impacts in the JVM process of using a specific configuration or by implementing a new data cleaning algorithm.

B.2 System Load Average

The $SystemLoadAverage$ was also selected to be used as metric because it provides a wider vision of the system. Besides, it is not process centric like the $ProcessCPULoad$ metric. The $SystemLoadAverage$ is the measure that averages, over a period of time, the number of processes in the runnable state (processes that are waiting to run) and the processes currently being run. In addition, it uses an exponential moving average in order to be easily calculated by the system, since it does not involve floating point computations. Usually, load average comes in a form of three floats as the average of three different periods: *i*) 1 minute; *ii*) 5 minutes; and *iii*) 15 minutes. To this work it was just considered the last minute because is the one that provides more detail.

The range of values for the system load average starts from 0 without a defined upper boundary. The lowest value 0 means that in average there was not any process being run or waiting for a CPU slot. For the case where there are only processes being run and there are no processes in the runnable queue, the value of the metric load average will be equal to the number of processors available in the system. For instance, for a single processor machine the value of the metric will be 1. Still, a value of 1 could also mean that there was an average of 1 process waiting in the runnable queue without any process being actually run. Consequently, only from the metric value, one can neither assure how loaded are the CPUs nor the number of processors.

B.3 Memory Usage

The last system metric chosen to profile a system is the $MemoryUsage$ of the application. In the JVM there are two different types of memory: heap and non-heap. This work is focused in the heap memory as it is used to allocate objects and the relations between them. Moreover, an RFID middleware platform is expected to stabilize in terms of the number of classes loaded to memory and have much more variability in the number of objects allocated.

The metric value is the current usage of the heap memory in bytes. Further, it counts both the memory occupied by the active objects and garbage objects that have not been collected yet. As a result, constant values along the time is the ideal scenario meaning that there are no memory leakages. On the contrary, an increasing value means that somewhere there is a memory leak. A value going up and down at some points means that there are some objects being created but also being collected by the Java Garbage Collector (GC), as expected. Nevertheless, the value itself is interesting to monitor because it can tell if the system is well dimensioned.