THE DESIGN AND DEVELOPMENT OF A MULTI-AGENT BASED RFID MIDDLEWARE SYSTEM FOR DATA AND DEVICES MANAGEMENT

Libe Valentine Massawe

Thesis submitted in fulfilment of the requirements for the

DOCTOR TECHNOLOGIAE:

ELECTRICAL ENGINEERING

in the

School of Electrical and Computer Systems Engineering

of the

Faculty of Engineering, Information and Communication Technology

at the

Central University of Technology, Free State

Supervisor:	**Prof. J. D. M. Kinyua**
Co-supervisor:	**Prof. H. J. Vermaak**

Bloemfontein

August 2012

Declaration of Independent Work

I, **LIBE VALENTINE MASSAWE**, passport number **AB061874** and student number **207071675**, do hereby declare that this research project, being submitted to the Central University of Technology, for the degree **DOCTOR TECHNOLOGIAE: ELECTRICAL ENGINEERING**, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology. It has not been submitted before to any institution by myself or any other person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

23 August 2012

…… ……

SIGNATURE OF STUDENT **DATE**

Abstract

Radio frequency identification technology (RFID) has emerged as a key technology for automatic identification and promises to revolutionize business processes. While RFID technology adoption is improving rapidly, reliable and widespread deployment of this technology still faces many significant challenges. The key deployment challenges include how to use the simple, unreliable raw data generated by RFID deployments to make business decisions; and how to manage a large number of deployed RFID devices.

In this thesis, a multi-agent based RFID middleware which addresses some of the RFID data and device management challenges was developed. The middleware developed abstracts the auto-identification applications from physical RFID device specific details and provides necessary services such as device management, data cleaning, event generation, query capabilities and event persistence. The use of software agent technology offers a more scalable and distributed system architecture for the proposed middleware. As part of a multi-agent system, application-independent domain ontology for RFID devices was developed. This ontology can be used or extended in any application interested with RFID domain ontology.

In order to address the event processing tasks within the proposed middleware system, a temporal-based RFID data model which considers both applications' temporal and spatial granules in the data model itself for efficient event processing was developed. The developed data model extends the conventional Entity-Relationship constructs by adding a time attribute to the model. By maintaining the history of events and state changes, the data model captures the fundamental RFID application logic within the data model. Hence, this new data model supports efficient generation of application level events, updating, querying and analysis of both recent and historical events.

As part of the RFID middleware, an adaptive sliding-window based data cleaning scheme for reducing missed readings from RFID data streams (called WSTD) was also developed. The WSTD scheme models the unreliability of the RFID readings by viewing RFID streams as a statistical sample of tags in the physical world, and exploits techniques grounded in sampling theory to drive its cleaning processes. The WSTD scheme is capable of efficiently coping with both environmental variations and tag dynamics by automatically and continuously adapting its cleaning window size, based on observed readings.

This work is dedicated to the memory of my loving son **Solomon Frederick Msae**; your memories linger on forever

Acknowledgements

First of all, I would like to thank God, for his grace, mercy and blessings to me, for giving me ability and perseverance to complete this thesis. Through Him All Things are Possible.

I would like to express my deepest gratitude to my supervisors Prof. Johnson Kinyua and Prof. Herman Vermaak; their continuous support, supervision, perceptive criticism, unfailing patience and guidance were invaluable. My thanks also go to Prof. Aghdasi Farhad for introducing me to RFID research and to Dr. M. M. Kissaka, my Head of Department at UDSM for his unwavering support and advice.

I would also like to acknowledge with appreciation my gratitude to the University of Dar es Salaam, the German Academic Exchange Service DAAD and South Africa's NRF, for scholarship and financial support received for this research work. Special thanks to the University of Dar es Salaam for granting me an opportunity to study as part of their staff development program.

Many thanks also go to my colleague and office mate Mr. Philibert Nsengivumva for his valuable discussions; all staff of the Department of Electrical and Computer Engineering at CUT and the members of Exchange Student House at CUT for their valuable co-operation and assistance throughout my studies.

A special thanks to my dear friends Mr Salim Kombe, his wife Mariam Kombe and their kids Michael, Malik and Mishaal. I will always cherish, appreciate and admire their kindness and hospitality to me and my family during my study period in Bloemfontein.

I would like to thank my family - especially my mother, Afres Eliamen; my sisters Clotilda and Noela; my brothers Elias and Gilbert; and all my relatives and friends for their constant prayers and support.

Last but not least, I would like to thank my husband, Frederick Joachim Msae, for his constant love, support, encouragement and understanding throughout my studies. Thank you for bearing my long absence away from home; your love has always been my pillar, my joy, and my guiding light - I thank you.

Table of Contents

List of Acronyms and Abbreviations

802.x	Set of IEEE Standards for LAN Protocol
ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
AOSE	Agent Oriented Software Engineering
API	Application Programming Interface
C1G2	Class-1 Generation-2 (EPC protocol)
CASE	Computer-Aided Software Engineering
COD	Communication Ontology Description
CUT	Central University of Technology
DAML	DARPA Agent Mark-up Languages
DARPA	Defence Advanced Research Project Agency
DF	Directory Facilitator
DOD	Domain Ontology Description
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPC	Electronic Product Code
ER	Entity Relationship
FIPA	Foundation for Intelligent Physical Agent
GPS	Global Positioning System
GUI	Graphical User Interface
HF	High Frequency
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
ID	Identification Data
ISM	Industrial, Scientific and Medical (portion of the radio spectrum)
ISO	International Organization for Standardization
JADE	Java Agent Development Environment
JPA	Java Persistent API
JPQL	Java Persistence Query Language
JRE	Java Runtime Environment
LEAP	Lightweight Extensible Agent Platform
LF	Low Frequency

MAS	Multi-Agent system
MASD	Multi-Agent Structured Definition
MySQL	My Structured Query Language
NFC	Near Field Communication
NRF	National Research Fund
OWL	Web Ontology Language
PASSI	Process for Agent Societies Specification and Implementation
PU	Persistent Unit
PVC	Polyvinyl Chloride
RDB	Relational Database
RDDM	RFID Device and Data Management
RDF	Resource Description Framework
RF	Radio Frequency
RFID	Radio frequency identification technology
RJ45	Registered Jack -45 (8 wire connector used in networking)
RPC	Remote Procedure Call
RS232	Recommended Standard 232 (Computer serial interface, IEEE)
SASD	Single-Agent Structure Definition
SMURF	Statistical sMoothing for Unreliable RFid data
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control protocol
UHF	Ultra High Frequency
UML	Unified Modelling Language
USB	Universal Serial Bus
WORM	Write-Once Read-Many
WSTD	Window Sub-range Transition Detection
XML	Extensible Mark-up Language

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

Radio frequency identification (RFID) is a technology that allows an object, a place or a person to be automatically identified without physical or visual contact. One simply needs to place a transponder (tag) in or on the object one intends to identify and query it remotely using a reader. The recent upsurge in interest in RFID technology stems from the development of low cost passive RFID tags with very small size and vigorous RFID standardizations efforts. With potentially significant applications [1] and the continuous decrease in both size and the cost of the passive RFID tags, it is predictable that every object could be tagged in the near future. This will enable item-level tracking replacing the omnipresent barcode labels that are used today in consumer products, opening a new range of pervasive computing applications.

While it is important to both select tags and readers including finding the right arrangement of antennas to recognize tags, we still need to deal with the issue of what happens with the data collected by the readers. How can the data be used to provide reliable information to an enterprise information system? Where does the data go next? RFID middleware addresses these questions. The middleware technologies can be categorizes into three levels [2]:

1. Software applications that ***solve connectivity problems and monitoring*** in specific vertical industries;
2. Application managers that ***connect disparate applications within an enterprise***; and
3. ***Device brokers*** that connect applications to devices.

The RFID middleware is the software subsystem that bridges the gap between the RFID hardware infrastructure which collects the data and the enterprise applications that wishes to utilize RFID data. It is an interface between the software components and the hardware components of the RFID system, and it is often referred to as the intelligent portion of the RFID system because it manages and coordinates it. The RFID middleware, therefore, provides the following benefits:

It insulates applications from the RFID hardware

The RFID middleware provides the application with standardized interface to access the RFID hardware. This standard interface acts like an adapter allowing an application to read data from many different readers even though they may be made by different manufacturers and may all have different interfaces. The application developers do not need to worry about writing special code to read from each of the readers because the middleware will handle this heterogeneity. A software layer of the RFID middleware incorporates all the device drivers of different hardware and exposes to the application a standard set of interfaces to access any hardware. The middleware reduces the work that must be performed by the application developers from building multiple interfaces to just one.

It processes raw data from RFID readers for consumption by applications

The middleware is responsible for receiving simple tag data generated by multiple readers, filtering that data, smoothing and aggregating it to produce useful events prior to their integration into existing information systems. In most cases, the amount of raw tag data read by the readers is large. As an example, consider that an RFID reader has the potential to acquire a tag every time it looks for a tag. If the reader is looking for a tag repeatedly at very small intervals (possibly ten or more times per second) it may actually report the same tag multiple times. However, most of the applications are most likely concerned with only one or possibly two "events", the first event is the acquisition of a new tag (the tag's entry into the reader field) and the second is the tag's exit from the reader field. To produce an event, the middleware may also need to associate the received data with a location and an activity. RFID middleware is responsible for determining what qualifies as an event and determines how it is reported to the interested application.

It provides an interface to manage readers

The middleware provides a standard interface to monitor and manage RFID readers existing in the RFID system. It allows a remote monitoring application to communicate with the readers in order to recognize their operational status, verify their correct operation and generate alerts when devices do not operate properly or links to devices are interrupted.

It provides an application-level interface for querying filtered RFID events

The RFID middleware also provides a standardized interface enabling an application to register for and receive filtered RFID events independent of which physical devices were used to collect the data or how it was processed.

1.2 Challenges in the Development of RFID Middleware

Despite the diversity of RFID applications, RFID systems have some common characteristics which are unique to the RFID domain [3]-[18] and they impose some significant challenges in the design of RFID middleware. These characteristics can be clustered into two groups; issues concerning data and issues concerning devices.

1.2.1 Characteristics of RFID Systems Data

Streaming and large volume

RFID data is generated quickly and automatically, and accumulated for tracking and monitoring. The data generated can be enormous, which requires a scalable storage scheme, to achieve efficient queries and updates. Also, due to the nature of RFID applications which demand track and trace queries for individual items and the large volume of data, the lineage tracking problem is more critical and challenging in RFID than in traditional data warehousing [4], [6].

Implicit semantic data

Raw data generated from an RFID system can be seen as stream of tuples comprised of the reader ID, observed tag ID, and the timestamp when the observation occurred [7]. This data carries implicit information about business processes such as changes of states, change of locations, and containment relationship among objects. For example, the detection of number of tags at the dock door over a certain period of time should be automatically translated as “shipment arrived” event. Extracting this implicit information from these raw data is the most interesting and challenging issue in an RFID middleware system. Therefore, a framework is needed in order to automatically transform the simple observed data into business logic data that enterprise applications such as inventory tracking and resource planning can use.

Inaccurate data

Missed reads are also an unfortunate reality with RFID systems [3]-[5], [9]-[11]. While reader performance is improving, cost pressures will always dictate that some RFID systems be used at the limited performance. In addition, problems such as reader interference and multipath fading will also cause many reads to be missed, "false negative reading". Moreover, the location tolerance that makes RFID tags easy to read also makes them difficult to understand; for instance, whether a tag is in fact in the reader's prescribed zone, or whether the read tag is simply passing by [8], "false positive reading". Such erroneous and unreliable data must be semantically filtered online before it is transformed into business logic data.

Data Redundancy

RFID data on average is less useful than other data streams [12]. For example, in traffic monitoring and financial applications, every record might be useful for further analysis. On the other hand, in the case of RFID data, we should be able to identify data that has been read multiple times. The less useful part of RFID data is the data that are continuously reported after the initial reading. For instance, in supply chain management, a tagged item can move to the shelf and sit the whole day on the shelf and send the data to the RFID management system constantly after every 10 minutes. But, from the management point of view, the most useful information for event detection is when the tagged item moves to the shelf and when the item is removed from the shelf. Data redundancy can also be caused by an item being in vicinity of more than one reader [13]; as a result its data is read by more than one reader. Therefore, it is necessary to have a filtering mechanism to reduce RFID data redundancy before processing the observed raw data.

Dynamic Data

There are two basic categories of data in RFID systems, static data and dynamic data [7]. Static data are related to commercial entities and product/service groups such as location information, product level and serial level information. While the entities are themselves static in the business processes, they dynamically interact with each other and generate event and state changes such as object location change, containments relationship change, etc. It is, therefore, essential to model all such information in an expressive data model suitable for application level interactions such as tracking and monitoring.

Spatial and Temporal

Most RFID based applications are, in general, not interested in individual readings in time or individual devices in space, but rather in an application-level concept of *temporal* and *spatial granules* [14]. These granules define the lowest-level, atomic unit of both time and space in which an application is interested [14]. For instance, in a retail scenario when an application continuously monitors the count of items on each shelf, the temporal granule would be each 5 seconds. Many applications are also interested in the information such as when a certain object was at a certain location. A spatial granule groups items based on some spatial categories which will be the lowest level of spatial granule at which the application operates, such as a shelf in a retail scenario. Therefore, the RFID middleware needs to have explicit temporal and spatial data models for RFID data to support tracking, tracing and monitoring application queries.

RFID data dissemination and Integration

The raw data generated from the RFID network itself is not valuable unless it is correlated to other information. For example, when the dock door RFID reader registers the arrival of a new pallet, it is vital to be able to correlate this arrival event of a pallet and all uniquely tagged cases to a purchase order, an invoice or advanced shipment notice. This means that RFID data have to be integrated with existing legacy enterprise applications. In addition to that, the information captured by a reader is usually of interest not only to a single application, but to a diverse set of applications across an organization and its business partners. The captured RFID data must thus be broadcasted to the entities that indicated an interest in the data. Due to the event-driven nature of many processes observed with the help of RFID systems, there is a need to support asynchronous messaging as well as a query-response model [15], [16]. Different applications also require different latencies. Applications that need to respond immediately to local interaction with the physical objects require a short notification latency that is comparable to the observation latency. Legacy applications that are not designed to handle streaming data might need to receive batched updates on a daily schedule [15]. This requires an RFID middleware to be easily configured for different applications.

1.2.2 Characteristics of RFID Systems Devices

Limited communication bandwidth

RFID systems rely on the availability of unlicensed frequency bands with a limited number of channels. For example, the 13.56MHz band has only one channel, and UHF frequency band in Europe allows for fifteen 200kHz-wide channels between 865.0 MHz and 868.0 MHz [17]. Readers need to listen for other transmitters using the channel before beginning to communicate with the tags. Since large distribution centres might need to run as many as 100 readers, it follows that readers need to co-ordinate their activities somehow to avoid reader interference and missing tags that pass by while the reader is not operating. Another constraint is the limited bandwidth available per channel, which limit the data transmission rate between readers and tags. Therefore, an efficient device management scheme is necessary to squeeze the maximum read rate out of larger RFID deployments [8].

Diversity of tag capabilities

Different RFID applications deploy different types of tags with different capabilities. For example, the memory on a microchip embedded in the tag usually contains a unique identifier but some microchips also feature small amounts of additional random access memory. Due to the increased power required to write to the EEPROM on the microchip, the maximum distance between reader and tag for a "write" operation is a fraction of that for a "read" operation [17]. Therefore, since RFID middleware is supposed to be application-agnostic, its design should take into consideration the diversity of tag capabilities.

Heterogeneous reader landscape

The diverse computational and networking capabilities of readers are also characteristic of RFID networks. Low-cost readers usually support only a single antenna and a serial RS232 interface. More sophisticated readers support several antennas, a TCP host interface, and ample computing resources for on-device data processing. For this reason, middleware should be flexible and scalable to accommodate a diversity of reader capabilities and reader density.

Sensor and actuator support

In many applications it is not sufficient to only identify objects, but the current state of the objects in the physical world has also to be detected as well. For example, a

perishable goods chain monitoring system should be able to monitor temperature data along the chain. The middleware has thus to provide the means to integrate sensors such as temperature, humidity or shock sensors and make their data accessible by the applications. In addition, in many applications it is not mandatory to operate RFID readers continuously due to the limited bandwidth available, and it is even undesirable to have readers transmit while no tags are present. To initiate the tag inventory process at a reader when there are tagged objects arriving in the read range, external sensors, such as motion sensors, should thus be able to trigger the RFID readers. In addition to sensors, applications often have to quickly interact with the physical world using different kinds of actuators such as locks or even simple traffic lights to signal an application state to an operator [18].

From the above analysis of RFID data and devices, it is apparent that the true benefit and complexity of RFID system does not come from reading the tags alone, but also from getting correct information from those reads to the right place in a usable form. This demonstrates that the middleware is a crucial component of RFID system and that it serves an invaluable purpose.

1.3 Statement of the Problem

Most of the commercial middleware solutions offered by major IT vendors such as Sun Microsystems, IBM, Oracle, Microsoft and SAP are proprietary in nature; they are costly and heavily dependent on the support software [19]. Their major intention is to extend their existing platforms and middleware to accommodate RFID data. They simply route RFID data to business applications by providing support for limited RFID rules; in fact, they only support primitive events or their simple combinations leaving the hard task of detecting complex events to client applications [19]. One disadvantage of such a system is that client applications are flooded with too much unnecessary data which it does not need.

Another issue is that most commercial RFID middleware solutions use a *fixed temporal smoothing filter*, a sliding window over the reader's data stream that interpolates for lost readings from each tag within the tag window, as a standard data-cleaning mechanism. Typically, the RFID middleware system requires the application to set the cleaning-window size. However, setting a cleaning-window size is non-trivial task that requires

careful consideration of both deployment environment characteristics and RFID devices dynamic patterns [10]. In fact, ascertaining for the environment characteristics and hardware and software configurations in order to get the required performance represents a significant portion of monetary and time cost associated with RFID deployments [20].

The research presented here attempts to tackle some of the middleware development challenges described in section 1.2. Specifically, this study addresses event processing concerns within the middleware, realizing an adaptive data-cleaning mechanism which minimizes software configurations and provisioning of a scalable distributed architecture.

1.4 Motivation for the Research

RFID middleware is among the cornerstone subsystems of every non-trivial RFID deployment. This is because RFID middleware typically undertakes the important tasks of interfacing to various RFID readers, filtering RFID data streams, processing tag streams, generating business and application events, while also interfacing RFID data with the enterprise information systems. RFID middleware makes it possible to realize the true benefits offered by the deployments of RFID technology.

RFID middleware affects the following techno-economic factors of the whole RFID technology deployment:

Total cost of ownership (TCO) of RFID deployment: The cost of RFID middleware is a critical component of the overall TCO of the RFID solution. Its contribution to the TCO varies according to the characteristic of the deployment. Depending on the amount envisaged for consumables and hardware, it may represent a significant cost component. The quality of RFID middleware also affects other components of the TCO such as maintenance and support costs.

Technical quality of the RFID deployment: The technical characteristics of an RFID deployment (including performance, functionality, and scalability) are largely dependent on the quality, robustness and versatility of the RFID middleware solution.

Affect other elements of the deployment (notably the hardware): The compatibility between hardware and middleware, as well as the effective operation of the hardware are highly dependent on the middleware used.

The above factors illustrate the importance of developing efficient and effective RFID middleware solutions that will enhance the overall cost and performance of RFID systems deployment. In general, this research is motivated by the need to provide a middleware solution, which will facilitate widespread deployment of RFID systems. Within the middleware proposed, novel solutions to some of the middleware development challenges discussed in section 1.2 are also addressed and provided.

1.5 Research Objectives

The main objective of this research work is to address some of the RFID device and data management challenges described in section 1.2. The specific objectives of this research work are to address the following challenges within the middleware:

1. To develop data cleaning or filtering techniques to clean the unreliable readings generated from the RFID data streams.
2. To develop a high-level data model to support efficient generation, updating, querying, and analyzing of both recent and historical RFID events.
3. To develop a scalable and distributed RFID middleware system architecture.
4. To develop an application-independent domain ontology for RFID devices.

To fulfil these objectives, a data model, a data cleaning scheme, a domain ontology and an RFID middleware prototype for managing RFID readers and processing the captured RFID data were designed and implemented.

1.6 Limitation of Study

One of the functionalities of the middleware system is to integrate the RFID system with the legacy enterprise applications. However, this functionality is not implemented in the middleware prototype under discussion. The current prototype uses a graphical user interface in which a user can configure the type of data desired to receive from the RFID system. Also, the implemented prototype middleware uses passive read- only tags. However, the proposed cleaning scheme and RFID data model can be used with other types of RFID tags and systems.

1.7 Research Methodology

The research methodology consist of three major activities as discussed below.

- **Development of a high-level data model to support efficient processing of the RFID events**

A literature study on the RFID data modelling and RFID event processing was done in order to get a thorough understanding of the subject. A new temporal-based RFID data model was then developed by extending the conventional Entity-Relationship (ER) model to include temporal information. ER constructs are made temporal by changing their semantics; that is, the ordinary relationship types are given temporal semantics making their instances record variation over time, rather than just single states.

- **Development of multi-agent based RFID middleware for data and devices management**

The key technologies used in the development of the multi-agent based RFID middleware are software agents, ontologies and relational databases. All the tasks involved in the middleware are assigned to different software agents. Agents within the middleware acts as a distributed community of data processing units able to: capture, manipulate, make decisions, and propagate information efficiently. The prototype middleware system is designed using the Process for Agent Societies Specification and Implementation (PASSI) methodology and implemented using the Java Agent Development Environment (JADE) platform and MySQL database. The choice of using PASSI methodology is driven by the step-by-step requirement-to-code guidance, and CASE tool support provided by this methodology. Also, PASSI is oriented towards a FIPA compliant implementation platform. JADE is the most popular and matured open-source based platform, which is well-supported by its developers and users. It is important to note that JADE is fully compliant with FIPA specifications.

Ontologies play a vital role in the development of multi-agent systems. All the information to be exchanged between the agents must conform to a common ontology. The ontology used for our middleware system is developed using Protégé ontology editor by following Ontology Development 101 methodology for ontology development. This methodology provides a simple guide that can be used even by non-expert in knowledge

representation field to develop an ontology. The JADE compliant ontology Java code is generated using Protégé ontology *Bean Generator* plug-in.

Persistence storage is an essential part of the proposed middleware system. It is required for storing of the ontological information, including RFID static and dynamic data, using the developed temporal-based data model in order to provide an efficient querying and analysis of both recent and historical RFID events data. A Relational Database (RDB), and in particular MySQL, is used for persistent storage in the developed prototype middleware system. RDBs are the most commonly used type of data-persistence mechanism. Since the developed middleware is focused on integrating the RFID data with other legacy enterprise applications, a RDB, which is more likely used by many enterprises, was chosen. MySQL is used because it is open source, free software and it is good enough for our prototype design, even though any type of RDB can be used.

While the middleware system is implemented using the Java-based JADE platform, which is an object oriented technology, the data storage mechanism is based on relational technology. Within the prototype middleware, the Java Persistent API also referred to as JPA is used as a persistent framework strategy to implement mapping between Java based objects and relational database. JPA is a Java programming language framework for managing *object relational mapping* in applications using Java platforms. JPA fully encapsulates database access from the application objects. It reduces the coupling between the object schema and the data schema in such a manner that simple changes in a data schema do not affect the application code.

- **Development of data cleaning technique**

A thorough review and analysis of RFID tag-reader performance and RFID data cleaning techniques in the existing literature was conducted. Based on the results of the analysis, the observed RFID readings were modelled as an unequal probability random sample of tags in the physical world. A new, adaptive window-based data cleaning scheme called WSTD was developed based on the binomial sampling and π-estimator statistical techniques. The WSTD cleaning scheme contains a novel method of detecting tag transition by comparing the two cleaning-window sub-range observations or estimated tag counts.

The performance of the developed WSTD cleaning scheme in comparison to other window-based cleaning schemes was then evaluated. The data sets for our experiments were generated by a synthetic data generator that simulates the operation of RFID readers under a wide variety of conditions using MATAB. The generator is composed of two components. The first component simulates the movement of tags and the second component simulates tag detection by the reader.

Finally, the implemented middleware prototype was updated to include the newly developed WSTD cleaning method.

1.8 Thesis organisation

The remainder of this thesis is organized as follows:

Chapter 2 provides a literature review and introduction of the concepts and technologies used in this research. These include an introduction to RFID technology, Multi-agent technology and Ontology technology.

Chapter 3 deals with RFID data modelling for efficient event processing. Readers' observations generated from RFID deployments are raw data which provide no explicit semantic meanings. In order for these data to be useful, they need to be transformed into semantic data properly represented in their own data models before they can be integrated into applications. In this chapter we present a temporal-based RFID data model which considers both applications' temporal and spatial granules in the data model itself for efficient event processing.

Chapter 4 presents an analysis and design of our prototypical agent-based RFID middleware system and **Chapter 5** presents the middleware system implementation. The middleware abstracts the auto-identification applications from physical RFID device specific details and provide necessary services such as device management, data cleaning, event generation, query capabilities and event persistence.

Unreliability of the data streams generated by RFID readers is among the primary factors which limits the widespread adoption of the RFID technology. RFID data cleaning is, therefore, an essential task in the RFID middleware systems in order to reduce reading errors, and to allow these data streams to be used to make a correct interpretation and analysis of the physical world they are representing. **Chapter 6** covers the proposed

adaptive sliding-window based data cleaning scheme for reducing missed readings from RFID data streams and **Chapter 7** presents the experimental evaluation of the cleaning scheme performance.

Chapter 8 is the last chapter of the thesis and gives the concluding summary of the work described in this thesis; highlights the contributions of the thesis; and gives recommendations for future work.

Chapter 2: Literature Review

2.1 RFID Technology

2.1.1 Introduction to RFID

RFID stands for Radio Frequency Identification and it is a type of automatic identification systems. RFID uses radio signals to acquire data remotely from tags within read or interrogation range. The data is then used for variety of purposes such as tracking an object, paying tolls or opening a door.

RFID technology allows an object, a place or a person to be automatically identified without physical or visual contact. RFID represents a major step forward in relation to the laser-based barcode system, which is the most popular automatic identification system used today. Compared to a barcode system, an RFID system offers several advantages. The first advantage is that the *RF tags do not require line of sight* to be read. While a barcode must be scanned directly by a laser beam and cannot be read if something opaque stands between the reader/scanner and the label, the RF tags can be read through many materials, including boxes and other radiolucent products.

The second advantage is *increased read range*; the effective read range of a laser-based barcode system is limited, because with increased distance comes an increased chance of materials passing between the reader's laser and the barcode label. For example, attempts in the past to use barcodes for toll-way use or railcar identification failed because the vehicle speed combined with the increased likelihood of rain, snow or debris interrupting the laser's line-of-sight at the crucial moment of passage rendered the technology highly unreliable for these applications.

The third advantage is the *ability to update the data*; barcode data is fixed the moment the label is printed. It cannot be changed unless a new label is printed and attached. On the other hand, many RFID tags can be reprogrammed in the field to reflect current information such as storage location or date placed in service. More sophisticated RFID tags can also be integrated with sensors to record dynamic conditions such as temperature or meter usage as they change, and then transfer the current conditions or a record of conditions to a reader upon request.

Other advantages are that RF tags can *store more data* than barcodes and multiple RF tags can be scanned at once in contrast to barcodes which require an individual label to be scanned at a time. In short, RFID raises the standard for automatic identification technology and allows it to perform more valuable functions than have been possible with barcodes.

2.1.2 The RFID System Components

The RFID system is comprised of at least three basic components that include tags, readers and a host computing device, as shown in Figure 2-1. An RFID tag or transponder is an identification device attached to an object to be identified while an RFID reader or interrogator is the device which recognizes the presence of RFID tags and read or write data to them. The reader can then inform a computer about the presence of the tagged items. The computer with which the reader communicates usually runs software that stands between readers and applications. This software is called middleware, and it is responsible for controlling the readers, receiving and managing the data generated by the readers and interacting with other enterprise applications interested in collected RFID data.

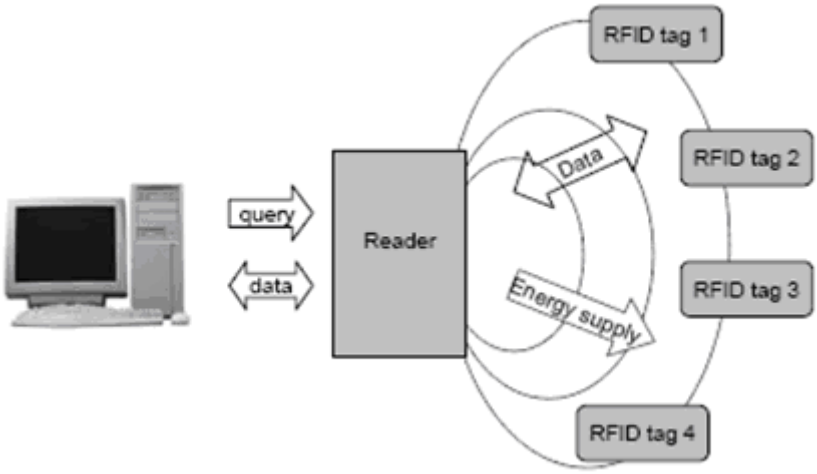


Figure 2-1: Basic components of RFID system

In addition to these basic components, RFID systems can also include other input and output devices such as sensors and actuators. These additional components act as triggers providing a certain level of automation to RFID systems.

2.1.3 RFID Tags

RFID tags are devices containing identification and other information that can be communicated to a reader from a distance. Basically an RF tag consists of an integrated circuit (a small silicon chip), an antenna and an optional memory component. The antenna is used to communicate with the reader by coupling the tag chip to the RF signal generated by the reader antenna. RF tag antenna designs vary based on environment, operating frequency and applications of the tag.

Different operating frequencies have different properties. Lower frequency signals are better able to travel through water, while higher frequencies can carry more information and can be read over a relatively longer distance. In the low frequency bands (LF and HF), the tag is in the near-field region of the reader antenna resulting in a short reading distances and magnetic coupling with inductors which are used for energy and signal transfer. Tag antennas for use in LF and HF bands are thus formed as coils (inductive loops). In the Ultra high frequency (UHF) and microwave bands, the tags are located in the far field region of the reader antenna resulting in a relatively larger reading distance; in this case dipole antennas are used for electromagnetic backscatter coupling for energy and signal transfer.

Although the RF spectrum is regulated mostly on a county-by-country basis, there are frequency ranges that are widely adopted around the world. Table 2.1 shows different frequency bands allocated for RFID applications, their availability, properties and limitations together with some typical RFID application areas [21]-[23].

Table 2.1: Radio Frequency bands: properties and applications

Frequency range	**ISM frequencies for RFID**	**Properties and Limitations**	**Some typical applications**
LF: 30 kHz – 300 kHz	9 - 135 kHz (Worldwide)	Reading distance less than 50 cm; Usable in metallic surroundings and on items with high water contents;	Animal identification; Central locking systems for automobiles
HF: 3 – 30 MHz	6.78MHz, 13.56MHz, 27.125MHz (Worldwide)	Reading distance less than 3 m; Works on metallic surfaces but shielding is required	Individual items; smart cards; access control; ticketing; e-passport; books
UHF: 300 MHz – 3 GHz	433 MHz (Worldwide) 860 – 868 MHz (Europe) 902 – 928 MHz (USA, Canada) 840 – 845 MHz /865 - 868 MHz / 920 – 925 MHz (China) 865 – 868 MHz / 915 – 921 MHz (South Africa) 950 – 956 MHz (Japan) 918 – 926 MHz (Australia)	Reading distance less than 9 m; High reading speed, but works poorly on metallic surfaces, under moist conditions and at close range	Boxes, cases and pallets in supply chain; aircraft luggage
Microwave: > 3 GHz	2.45 GHz, 5.8 GHz (Worldwide)	Reading distance greater than 10 m; Used in WLAN and WiFi systems	Road toll and vehicle identification of all sorts; container tracking systems

RF tags may have just enough memory to hold only the simplest information such as an identification code (little more than the amount of data on the average barcode label), or may have as much memory and processing power as a small computer.

RFID tags can be classified in terms of their energy source, storage capacity and whether the stored data can be modified (rewritten) [21]. Classification of RFID tags is described below followed by a description of their physical characteristics and factors affecting their read range.

2.1.3.1 Active tags

Active tags are tags with a built-in internal power source in the form of battery or a wired powered connection that provides the necessary power for tag operation and communication with the reader. The active tag can be designed with a variety of specialized electronics including microprocessors, different types of sensors or input/output devices. They can process and store the input data from the sensor for immediate or later retrieval by a reader. Active tags in general have a range of about 3 meters to more than 100 meters.

Active tags can normally be read and written and configured in various modes. In standard mode, they transmit their ID code at a defined interval, possibly accompanied by other data. In order to save energy, active tags can also be put in quiescent mode, which they can only exit in response to a specific request from the reader or on the occurrence of a defined event generated by the sensor data, such as harmful vibration, deviation from specific temperature range or hazardous operating conditions.

The primary advantages of the active tags are their higher reading range and reliability. Because they are powered by an internal battery, their field strength is higher, making it possible to be read from further distances and they do not need a continuous radio signal from the reader to power its internal circuit - making them more reliable. The limitations of active tags include limited lifetime of the battery, the cost and the size of the tags - which are directly linked to the inclusion of the battery in the tag.

2.1.3.2 Passive tags

Passive tags are tags that do not have a built-in power source; they are completely powered by the incoming RF signal from the reader. The incoming RF signal from the reader induces in the tag antenna a tiny but significant electrical current to activate the tag. They transmit data by modulating the RF signal of the reader antenna and they have a short read range of only a few meters (approximately 9 meters).

Primary advantages of passive tags are their smaller size, cheaper price and unlimited lifetime because they do not use batteries. The limitation of passive tags is that the range of operation in terms of distance from the reader is limited to only a few meters.

2.1.3.3 Semi-active tags

Semi-active tag is a combination of active and passive tag; it has a battery like active tags, which it uses to power its internal circuit, but it still uses the reader's power to communicate with the reader - just like passive tag. Semi-active tags have the read reliability of the active tag with slightly improved read range of up to 30 meters. In addition, since semi-active tags become active only when they enter the RF field of the reader antenna, they have a longer lifetime than a fully active tag.

2.1.3.4 Read-only tag (RO)

These tags are assigned multi-bit (64-bit or 96-bit) data during the tag manufacturing stage (they are also referred to as factory programmed) and the data in the tag cannot be modified. RO tags are the simplest and most economical tags. Another variant of read-only tags is Write Once, Read Many (WORM) tags. WORM tags enable users to encode tags at the first instance of use with non-modifiable data, which cannot be changed but can be read as often as desired.

2.1.3.5 Read/Write tag

Read/Write tags can be reprogrammed in the field either by a dedicated programming device or by the reader itself. They have additional memory space where data can be written and modified. User data, handling instructions, sensor data or process data, can be stored in these tags. The memory space in these tags varies from just few bytes to hundreds of kilobytes. Writable tags can store data that can later be read by the authorized reader. This function is especially useful in situations where access to the database containing object data is not possible.

2.1.3.6 Physical characteristics of RF Tags

Because RFID tags must be physically attached to items of different shapes and sizes in different environments, they come in a wide assortment of shapes and sizes. The smallest tag, as of today, is the μ-tag produced by Hitachi [24], which only measures 0.4 millimetres including the antenna. Furthermore, they may be housed in many different kinds of materials. Some of the physical characteristics of various tags include [22]:

- PVC or plastic buttons and disks, usually including a central hole for fasteners. These tags are durable and reusable.
- RFID tags shaped like credit cards, which are called “contact-less smart cards”.
- RFID tags made into layers of paper in a label, called “smart labels”. These may be applied with automated applicators similar to those used for bar code labels.
- Small tags embedded in common objects such as clothing, watches, and bracelets. These small tags may also come in the form of keys and key chains.
- Tags in glass capsules, which can survive even in corrosive environments or in liquids.

RF tag antennas are often etched or printed metallic pattern on a circuit board or a thin film inside a small case, or sandwiched between layers of printed label.

2.1.3.7 Tag Read Range

Tag read range depends on much more than just the characteristics of the tag. Reader power and sensitivity, antenna range and polarization, and the reading environment can all affect the range at which a given tag may be successfully read. Certain attributes of the tag itself and its immediate surroundings also help determine a tag’s full read range, including:

- Tag power source (battery-powered tags typically have greater range than those powered exclusively by the RF beam).
- Type of materials between and around the tag and the reader.
- Tag position relative to the antenna’s preferred orientation.
- Relative tag speed (amount of time the tag is within read range, if either the tag or the reader is moving relative to the other).
- Amount and rate of data to be exchanged between tag and reader and the overhead involved in error correction and other quality processes.
- The tag antenna design.

Tags, like every other element in an overall system design, affect system performance and should be configured to optimize the specific applications they are to be used for.

2.1.4 RFID Readers

An RFID reader or interrogator is an electronic device which recognizes the presence of RFID tags and read or writes data to it. RFID readers have three fundamental building blocks; the HF interface system, control system and antenna (e) [25]. These parts interact with each other and with an external host system as shown in Figure 2-2.

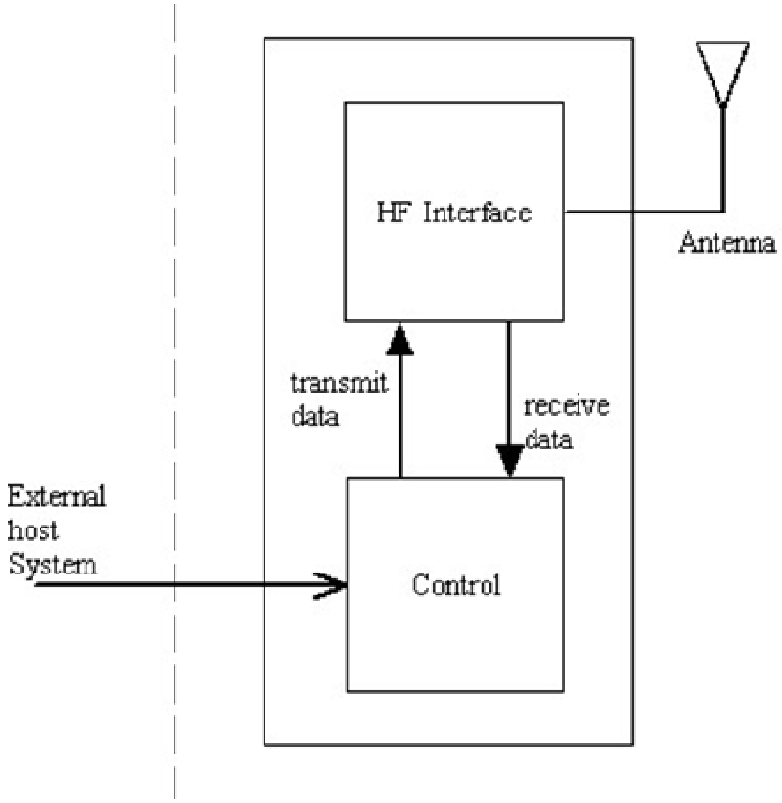


Figure 2-2: General RFID reader block diagram [25]

2.1.4.1 HF interface

The HF interface system provides an interface to the antenna which transmits and receives signals and data to and from the tag. It generates the high frequency power which activates the tag and supplies it with power. It is also responsible for modulation/demodulation of data transmitted/received signal to/from the tag.

2.1.4.2 Control system

The control system provides essential communication links between the reader and the external entities. It controls all communication protocols with the tags, performs signal

coding and decoding and communicates with other reader input and output devices such as back-end host computer, sensors and actuators. Control systems incorporate different types of communication interfaces such as serial port connection RS232 or RS485, Ethernet RJ45 adapter, 802.x wireless interface and universal serial bus (USB).

2.1.4.3 Antennas

Antennas are required to transmit and receive RF signals to and from the tag. In the LF and HF band antenna is a coil similar to a tag antenna. It is shaped to achieve the best possible coupling with the antennas of the tag to be read. In UHF band there is wide variety of antenna designs and they are tuned for specific environments in which they will be deployed. Depending on the type of the reader, an antenna can be integrated within the reader or external to the reader, also a reader can have one or more antennae connected to it. The critical importance of the use of the appropriate antenna in a specific environment cannot be underestimated for the success or failure of the communication between the tag and reader. RFID readers can be categorized based on their communication interface or by its mobility. A brief description of each category is given below.

2.1.5 Air interface and Standards

2.1.5.1 The Air Interface

Air interface is the RF field that forms the link between the reader and the tag. The air interface specification is sets of rules which describes the operating frequency, the tag's power source, type of coupling, communication mode, type of modulation, data-encoding methods used as well as the structure of transferred commands between reader and tag. These commands includes commands for reading and writing data, controlling the anti-collision protocols, blocking individual memory cells and disabling tags.

2.1.5.2 RFID Standards

In the past few years, several organizations have attempted to create a single standard for communications in RFID that is compatible across various tags and readers to facilitate the rapid growth of RFID solutions. In the RFID field two main families stand out: ISO (International Organization for Standardization [26]) standards and EPC (Electronic Product Code [27]) standards. These standards address many, if not all, aspects of RFID

communications, from how the reader and tag communicate with each other (air interface specification) as well as specific uses of RFID technology. Standardization is thought to be the only means in which RFID can penetrate industrial solutions on a major scale. Standards ensure interoperability of various components of RFID systems that may be provided by different manufacturers'. Standards also ensure the development of dual frequency and wide frequency bands RFID tags (tags that can be used over the entire UHF band) for open RFID system applications (such as supply chain management) to be read worldwide without breaking radio regulation laws in different countries e.g. ISO 10374 tags supports both 850 – 950 MHz and 2.4 – 2.5 GHz frequency bands for freight container identification.

2.1.5.2.1 ISO standards

In ISO family, the key defined RFID standards categorized based on specific applications are: 11784/5 and 14223 for animal identification [28], 14443, 15693 for contact-less smart cards [28]; 10374, 17363, 18185 for freight containers [29]; and 17358, 17364–17367 for supply chain applications. Standards which describe the communication protocols between reader and tags are described in ISO 18000 series standards, while the data organization inside the tag independent of actual technology is specified in the 15961-15963 standards. Procedures for measuring the efficiency of devices and air interface communications to verify the conformity of a device with standards are described in 10373, 18046 and 18047. Finally, ISO 21481, 23917 and 22536 describes standards for Near Field Communication (NFC) protocols.

2.1.5.2.2 EPC Standards

EPC standards are focused primarily on UHF frequencies in the 860 – 960 MHz band with electromagnetic coupling between tags and readers. EPC standards classify different classes of tags according to their functionalities, whereby each successive class is more sophisticated than the one below it. Table 2.2 shows an overview of EPC tag classification. Passive tags are classified in classes 0 to 3, Class 4 describes active tags, and Class 5 is reserved for tag readers and active tags that can read data from other tags.

In mid-2006, ISO approved the Generation 2 UHF Air protocol as part of its ISO 18000-6 standard, as amendment 18000-6c, which is the first common standard of ISO and EPCglobal. The inclusion of the EPC standard in the global ISO standard is seen as an

important step towards a broad implementation of Class 1 Generation 2 RFID systems worldwide, as the World Trade Organization has guidelines about following standards endorsed by ISO and other global standards bodies [30].

Table 2.2: Overview of EPC Tag Classification [31], [32]

Class	**Functionalities and properties**	**Operating Frequency and Communication**
Gen. 1, Class 0	Passive identity tags; Read-only; Stores up to 96 bits Most basic and thus cheapest type of tags	860 – 930 MHz Backscatter
Gen. 1, Class 1	Passive tags; Read/Write Once; Stores up to 96 bits	860 – 960 MHz Backscatter
Gen. 2, Class 1	Generation 2: uniform specification that merge Generation 1 classes 0 and 1 with improved features; Passive tags with at least 256 bytes of memory; Read/Write; Adopted as ISO standard 18000-6c	860 – 960 MHz and 13.56 MHz Backscatter
Class 2	Passive tags with supplementary functions, such as data storage for encryption (security features)	860 – 930 MHz Backscatter
Class 3	Semi-passive tags; Read/Write; < 100 Kilobytes of memory; security and sensors features	860 – 930 MHz Backscatter
Class 4	Active tags; Read/Write; security and sensors features; They may be capable of broad-band peer-to-peer communication with other active tags in the same frequency band, and with readers.	860 – 930 MHz Active Transmission
Class 5	Class 5 tags are essentially readers. They can communicate with all other classes and with each other.	860 – 930 MHz Active Transmission

2.2 Multi-Agent Technology

The RFID middleware prototype system developed in this research work employs software agents' engineering methodologies. Hence, this section provides an introduction to the software agents' technology and the motivation for using software agents' technology for the middleware development.

2.2.1 Definition of Software Agent

There is no universally accepted definition of "software agent", but the definition proposed by Jennings *et al*. in [33] has received much recognition from the researchers in the field. A software agent, which is henceforth referred to as 'an agent' is a software-based computer system, *situated in some environment* that is capable of *flexible autonomous* action in order to meet its design objectives [33].

Agents are characterized by following properties that are presented here in the form of definitions:

- *Autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
- *Social ability*: agents interact with other agents (and possibly humans) via some kind of agent communication language.
- *Reactivity*: agents perceive their environment and respond in a timely fashion to changes occurring therein.
- *Pro-activeness*: in addition to acting in response to their environment, agents are able to exhibit goal-directed behaviour by taking initiative.

To support interaction with its environment, an agent is equipped with a sensory system for receiving external information and a knowledge base about its environment. The actions of an agent alter its environment and thereby influence its future decisions. The prerequisite for this interaction is that the agent has suitable information about its environment. Performing actions and communicating with other agents are not possible in an unknown environment [34]. The common representation of the environment is known as ontology and it is discussed in section 2.3.

2.2.2 Multi-Agent Systems

A Multi-agent system (MAS) is a system that is designed and implemented as several interacting agents. It can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver [35]. Agents communicate with each other and thus perform cooperative activities in their environment. According to Jennings *et al*. [33], MAS is ideally suited for representing problems that have multiple problem solving methods, multiple perspectives and/or require multiple problem-solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions. Examples of common types of interactions include:

- Cooperation - working together towards a common aim;
- Coordination - organizing problem-solving activity so that harmful interactions are avoided or beneficial interactions are exploited; and
- Negotiation - reaching an agreement acceptable to all the parties involved.

It is the flexibility and high-level nature of these interactions that distinguishes multi-agent systems from other forms of software and provides the underlying power of the paradigm.

MAS has the following advantages over single agent and centralized systems:

- MAS distributes computational resources and capabilities across a network of interconnected agents. Whereas a centralized system may be plagued by resource limitations, performance bottlenecks, or critical failures, MAS is decentralized and thus does not suffer from the "single point of failure" problem associated with centralized systems.
- MAS allows for the interconnection and interoperation of multiple existing legacy systems. By building an agent wrapper or transducer agent around such systems, they can be incorporated into an agent society.
- MAS allows for efficiently retrieving, filtering, and globally coordinating information from sources that are spatially distributed.
- MAS models problems in terms of autonomous interacting component-agents, which is proving to be a more natural way of representing task allocation, team planning, user preferences, open environments, and so on.

- MAS provides solutions in situations where expertise is spatially and temporally distributed.
- MAS has the potential to enhance overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.

A detailed survey of research on programming languages, methodologies and development tools for MAS can be found in [36]-[38].

2.2.3 Motivation for using MAS for RFID Middleware

Agents are being used in an increasingly wide variety of applications, ranging from comparatively small systems such as email filters to large, open, complex, mission-critical systems such as air traffic control. A comprehensive review of commercial and industrial applications using multi agent-based systems can be found in [39].

There is a strong correlation between RFID system deployments and the types of applications supported by multi-agent systems. Most RFID system deployments are distributed in nature; RFID readers installed in different strategic locations within the organizations are linked together creating a distributed network of readers. Also, in most cases information collected from the RFID system is expected to be communicated to the other existing legacy organization applications. MAS, on the other hand, is characterized by the ability to solve problems in which data, expertise or control is distributed and it also allows for an easy integration of the new system with the existing legacy system. MAS offers system scalability and load balancing, which are the features desired in RFID system. These are the main factors for our decision to develop our RFID middleware prototype as a multi-agent system.

The middleware system in this study is developed using agent oriented software engineering methodology called PASSI [40] and implemented using Java Agent Development Environment (JADE) platform [41]. Our choice of using PASSI methodology is driven by the step-by-step requirement-to-code guidance, and CASE tool support provided by this methodology. Also, PASSI is oriented towards a FIPA [42] compliant implementation platform. JADE is the most popular and matured open-source based platform, which is well-supported by its developers and users. Most importantly, though, JADE is fully compliant with FIPA specifications.

2.3 Ontology Technology

For agents to interoperate, cooperate and coordinate, they need a common understanding of the domain they are working in. That common representation of the objects, concepts, entities and relationships within the domain is refereed as an ontology [43]. Ontology plays a vital role in the development of a MAS. This section introduces the notion of an ontology; its benefits in the design and development of multi-agent systems, and lastly describes different types of ontologies.

2.3.1 Definition of an Ontology

Many definitions of ontologies have been given in the last two decades, but the most-cited definition in the knowledge-sharing community is given by Gruber [43]:

"An ontology is an explicit specification of a shared conceptualization".

A ***conceptualization*** refers to an abstract, simplified view of the world that we wish to represent for some purpose. The world view is often conceived as a set of concepts (e.g. entities, attributes, processes) their definitions and their inter-relationships. ***Explicit*** means that the type of concepts used and the constraints on their use are explicitly defined. ***Shared*** reflects the notion that an ontology captures consensual knowledge, that is, it is not restricted to some individual, but accepted by a group. An ontology provides a shared vocabulary, which can be used to model a domain that is, the type of objects, and/or concepts that exist, and their properties and relations. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some ontology, explicitly or implicitly [43].

2.3.2 Motivation for Ontologies in MAS

The literature is currently rich with discussion of ontologies' importance [44], such as in the areas of knowledge engineering [45], information retrieval [46], database design and integration [47] and natural language processing. This section focuses on the importance of ontologies in the context of MAS. Within this context, ontologies have been widely recognized for their significant benefits to *interoperability*, *reusability*, *MAS development activities* and *MAS operation* [48]-[52]. These benefits are actually inter-related with each other, as will be mentioned throughout the discussion.

2.3.2.1 Benefits of ontologies to interoperability

Interoperability refers to the ability of heterogeneous components to interact and work with each other to achieve shared or individual goals [52]. Interoperability involves not only communication between the heterogeneous components, but also the ability of these components to use the exchanged information. In MAS, interoperability issues may arise between heterogeneous agents or between heterogeneous non-agent resources (such as knowledge sources and legacy application systems). Two prominent interoperability issues are [53]-[55]:

- Semantic heterogeneity issue: occurring when the knowledge base of each agent, or the information/application of each resource, uses a different vocabulary to express the same information (e.g. "Price" versus "Cost") and/or uses the same vocabulary to express different information (e.g. concept "Employee" in one agent/resource means anyone currently on payroll but in another agent/resource means anyone currently receiving benefits, thus including retirees). Another example of semantic heterogeneity is the scaling conflict, where the same concept refers to the different scales or references of measurement (e.g. concept "Price" may be measured in dollar in one agent/resource but in euro in another); and
- Structural heterogeneity issue: occurring when the knowledge base of each agent, or the information/application of each resource, uses a different conceptual schema to represent its data. For example, concept "Customer-Name" is represented as an object in one agent/resource but as an attribute in another.

Both of these heterogeneity issues can be addressed by the use of ontologies [45], [50], [54]. Specifically, when the knowledge bases of heterogeneous agents and the information/applications of heterogeneous resources are explicitly conceptualized by ontologies, the structural and semantic interoperability between these agents and resources can be achieved by mapping between these ontologies. Such a mechanism is known as "ontological mapping", i.e. specifying the semantic correspondences between the concepts of one ontology with those of another. An in-depth discussion of on ontological mapping can be found in [44], [45], [50].

2.3.2.2 Benefits of ontologies to reusability

The capability of ontologies to enhance reuse has earlier been acknowledged and exploited by the Knowledge Engineering community in the development of knowledge

based systems [43]. An ontology was employed to capture domain knowledge of a system, while the system's problem solving knowledge, which specifies the domain-independent reasoning steps to solve the problem, was stored separately in a Problem Solving Method. Consequently, each knowledge-based system was designed as being composed of two components: a Problem-Solving Method and ontology [52], [56], [57]. This modularity in knowledge modelling, which was made possible by ontologies, enables the reuse of Problem-Solving Methods across different problem domains and the reuse of domain knowledge across different problems [44], [48], [54].

In the context of MAS development, the above ontology-based mechanism of reuse could still be applied, since each agent in MAS is basically a knowledge-based system. Each agent can be modelled as being composed of two major knowledge components: the behavioural knowledge component, which captures the problem-solving knowledge of an agent in the form of plans, reflexive rules and/or actions that guide the agent's behaviour in achieving its goals, and the (local) domain knowledge component, which contains the ontologies defining the domain related knowledge requirements of the agent's behaviour. Given this approach of agent knowledge modelling, an agent's behavioural/problem-solving knowledge can be reused across agents with similar behaviour/goals in different domains, and its domain related knowledge can be reused across agents within the same domain area.

2.3.2.3 Benefits of ontologies to MAS development activities

Two major activities of MAS development that can be greatly facilitated by the use of ontologies are system analysis and agent knowledge modelling.

System analysis involves the formulation of the problem to be solved (e.g. elicitation of system goals) and/or the representation of the application's domain knowledge [55].

- With regard to the problem formulation, the availability of an ontology which holds explicit, comprehensive knowledge about the target domain will greatly promote the developer's understanding of the application, thereby facilitating its elicitation of the system goals and responsibilities. In fact, a weak ontological analysis often leads to an incomplete or inaccurate understanding of the application, thereby leading to an incoherent system [45]. The first step in developing an effective knowledge-based system has been recommended to be an effective ontological analysis [56]. Moreover,

when the target application covers multiple domains, the mappings between domain ontologies will help the developer to grasp the associations amongst these domains. These associations are particularly important if the development project involves multiple developers working on different domains [44].

- With regard to the representation of the application's domain knowledge, ontologies offer a structured, explicit, human-readable mechanisms for representing domain knowledge. These characteristics promote the readability of an ontology, hence making it a reusable enhanced representation mechanisms.

Agent knowledge modelling refers to the specification of local knowledge of each agent in the MAS, including problem-solving knowledge and local domain-related knowledge. Just as for application's domain knowledge, an ontology can be used as an effective representation mechanism for agents' local domain-related knowledge (which is typically a portion of the application's domain knowledge) [54]. Different parts of ontologies can be assigned to different agents to represent the agents' different views of the world [48]. In addition, as previously discussed, ontologies offer a mechanism for decoupling the modelling of agent domain-related knowledge from its problem-solving knowledge, hence promoting the reuse of agent knowledge modules.

2.3.2.4 Benefits of ontologies to MAS operation

Ontologies are beneficial to two major aspects of MAS operation: communication and agent reasoning.

- Communication in MAS may occur between agents, between agents and non-agent resources, and between agents and human users.

  Regarding **inter-agent communication**, even though sharing a common agent communication language (ACL) will allow agents to exchange messages (thanks to the common communication syntax), it does not ensure that the communicating agents will interpret the exchanged messages in a uniform and consistent manner, i.e. to share the same understanding of the semantics of the messages [44], [48]. Successful agent communication requires "ontological commitment" of the agents, i.e. an agreement between agents to share an ontology during communication [58]. This shared ontology provides the agents with a set of common vocabulary for formulating and interpreting the content of the exchanged

messages. This means that the local knowledge of each agent should contain the common ontology that is used for communication. This requirement indicates the inter-dependency between the ontology's role in agent communication at run-time and the modelling of agent knowledge at design-time.

- Regarding **agent-resource communication**, non-agent resources are normally accessed by agents via "wrappers", i.e. specialized agents that provide interface to the resources [59]. Client agents can relay ACL queries and commands to the wrapper agents, which in turn translate and invoke them onto the underlying resources (Figure 2-3).

Figure 2-3: Agent-resource communication

Ontologies can be used to conceptualize the resources' internal data and/or application, thereby allowing the wrapper agents to determine which vocabulary they should use to formulate input queries/commands to the resources and interpret outputs, without having to access the resource's internal structure [58].

- Regarding **human-agent communication**, ontologies can be used to facilitate the formulation of user queries and the representation of queries' results. When a query/command needs to be formulated, the human user can consult the ontology committed by the agent receiving the query and use the vocabulary defined in that ontology as query terms [51], [60]. A query composed this way will be directly understood by the queried agent without any need for further query processing. When the results of the query are found, they can be represented using the same ontology as that previously used for query formulation. This allows the human user to receive a single representation scheme of the results, even if the results have been gathered from heterogeneous resources with different local representation schemes [51].

- **Agent reasoning** at run-time processes the problem-solving knowledge of the agent, and uses the domain-related knowledge held by the agent as inputs [46]. If the domain-related knowledge has been modelled as an ontology during agent

knowledge modelling at design time, with all relevant domain concepts and relationships being explicitly defined the agent reasoning process can easily utilize this knowledge and make the most out of it. The following are a few examples of how ontology-based knowledge can facilitate agent reasoning.

- The taxonomy of concepts in an ontology can help agents to process a user query by decomposing it into sub-queries.
- Mappings between ontologies may help agents to make useful inferences.
- Mappings between ontologies of heterogeneous resources and a common ontology may help agents to determine the appropriate resources to use without having to access each resource's internal data.
- Axioms, rules and assertions that specify constraints on concepts and relations (if any) may help agents to reason.

2.3.3 Classification of ontologies

There is no universally accepted classification of ontologies, but the most common taxonomy for classifying ontology is by their level of generality [48], [61]-[63]: *generic/top-level/upper-level* ontologies, *domain* ontologies, *task* ontologies and *application* ontologies as shown in Figure 2-4. Generic/top-level/upper-level ontologies specify the general knowledge about the world, providing basic notions and concepts for things like space, time, matter, object, event and action. These concepts are independent of a particular problem or domain and can be re-used across applications. Domain ontologies describe concepts which are specific to particular domain. Domain ontologies may be reused across applications that belong to the same domain. Domain ontologies can be developed by refining generic ontologies. Task ontologies describe the entities relevant to problem-solving tasks and methods. Task ontologies can be also developed by refining generic ontologies. Application ontologies describe concepts that are specific to an application. Since each application is typically characterized by both a particular domain(s) and a particular task(s), Application ontologies are thus a synthesis of domain ontologies and task ontologies that have been specialized to model the application's specific knowledge needs.

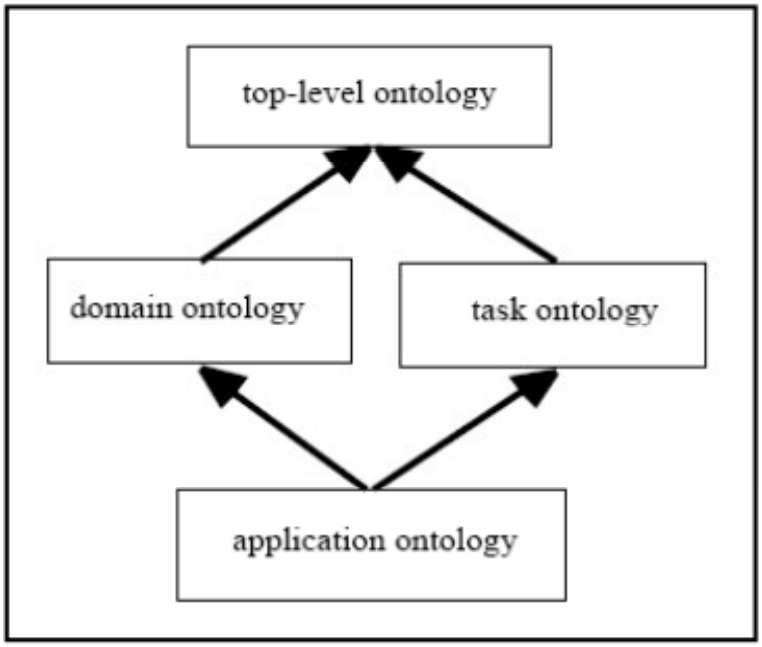


Figure 2-4: Types of ontologies [61]

A more comprehensive review of ontologies including their representation languages, development tools and methodologies can be found in [63]-[66].

Ontology plays a vital role in the development of MAS. Because of no existing developed RFID ontology, the gap was filled by developing the ontology for RFID devices used in the MAS based middleware system discussed. RFID device ontology includes a description of devices which are specific to RFID domain. This ontology can also be reused or extended to be used in other applications. The development of the ontology which is combination of RFID device ontology and task an ontology is covered in Chapter 5, Section 1.

Chapter 3: RFID Data Modelling

The RFID middleware is the software subsystem which bridges the gap between the RFID hardware infrastructure which collects the data and the enterprise applications that wishes to utilize the collected RFID data. It addresses questions such as; where does the collected data go next, and how can the data be used to provide useful information to the enterprise information systems? Therefore, a proper data model is vital for the efficient functioning of the middleware.

RFID data modelling is the process of translating a physical RFID deployment world into its corresponding virtual world. Readers' observations generated from RFID deployments are raw data which provide no explicit semantic meanings. In order for this data to be useful it needs to be transformed into semantic data properly represented in its own data model before it can be integrated into applications. Hence, RFID data modelling is the first and essential step for managing RFID data and for supporting business applications.

The work by Harrison [67], summarizes the characteristics of RFID data, and provides some reference relations to represent the data. In their model, RFID data is modelled as events, thus state history and its temporal semantics are not explicitly modelled. This data model is not effective in supporting complex queries such as object tracking. Complex queries often need to be divided into numerous steps that are indirect and inefficient [67]. Wang and Liu [4] propose a data model for RFID application which models both RFID events and states. Their model provides a more efficient support for complex queries. The model by Wang and Liu is based on the event observations generated from a single reader. However, as was pointed out above, in general, most of RFID based applications are not interested in individual readings in time or individual devices in space, but rather in an application-level concept of *temporal* and *spatial granules* [14]. These granules define the lowest-level, atomic unit of both time and space in which an application is interested [14]. The model presented here builds on the Wang and Liu model [4], but considered the application of temporal and spatial granules in the data model itself. In the data model under discussion, event observation is a combination of observations from one or more readers that cover the observed location of interest. Also, taking into consideration the temporal granule aspect of RFID application events and the need to reduce the volume of data generated by the RFID system as well as the way of dealing

with erroneous data, observations are modelled as macro-events with start and end time instead of instant events. In this section we present our proposed data model and discuss how the model is implemented in middleware.

There are two basic categories of data in RFID systems, static data and dynamic data [7]. Static data are related to commercial entities and product/service groups such as location information, product level and serial level information. These are static attribute data about an object such as product description, which are more likely to remain the same throughout the lifecycle of the product. For example, the previous modelled RFID device ontology describes the static attributes of the RFID devices.

3.1 Static RFID Data

Although many entities may exist in RFID application, only some of them are directly related to RFID. These fundamental entities in RFID applications follow from an examination of RFID systems:

Tags – refers to the RFID tags also known as RF transponders. Tag has unique identification code stored in its memory and it is attached or embedded into the object for the purpose of identification using radio waves.

Objects – An object is referred to as any entity tagged with an RFID tag. RFID tags act as a proxy of their corresponding objects. Observations of tags represent observations of the corresponding objects.

Reader – These are RFID readers which use radio frequency signals to communicate with RFID tags and read data from tags or write data into tags. Each reader is also uniquely identified by its RFID Tag.

Sensors – These can be sensors embedded in the RFID tags or standalone sensors such as temperature sensors, motion sensors, or location sensors. Tag-embedded sensors measure a target and then write the measurement to its master RFID tag.

Location – Location can be geographical location or symbolic location [67]. Symbolic location is way of indicating that an object is in/on/at a particular discrete location. Location can contain another location. Different applications may have different location semantics; hence, granularity of symbolic location is defined according to application need. Continuous geographical locations can be identified by a local sensor (e.g. GPS) or

a local position radar combined with reader as demonstrated in [68]. Some hybrid, active tags with integrated GPS are also available [68].

Containment - Another concept in RFID application related to location is the concept of containment. Containment determines the hierarchical relationship among objects. For example, the tagged shelf containing tagged books, and the tagged case containing tagged items. The containment relationship implies important logic information; for example, a container object and its containing objects share the same location. Another similar concept is *association*, where tagged objects are associated with certain relationships. For example, a room may be associated with the list of furniture contained in the room, and/or a toolbox associated with the list of tools.

Logical readers – logical readers are abstract names given to one or more readers that have a single logical purpose. For example, in a large warehouse, there may be five loading dock doors each with three readers for a particular door. It is more likely that an application trucking the flow of goods into trucks would want to read each individual door readers observations combined as a single event rather than combining all the warehouse door reader observations into single event. The logical reader name approach is more desirable than using reader identities - especially in the applications deploying fixed readers for the following reasons:

- If the reader is changed, the unique identity of the reader will change, forcing the application configuration to be changed.
- If the number of readers must change, e.g. because it is discovered that four readers are required instead of three to obtain enough coverage of a particular dock door, then the application configuration must be changed.

The logical reader concept is synonymous to location. In the prototypical middleware discussed here, the location and logical reader are used interchangeably.

Operation - This refers to the situation where by the reader acts as a proxy of operation or operator. For example, the detection of number of tags by the dock door reader may mean "shipment arrival" operation. In a movable reader deployment an operator can either be a human operator or a motion control system. An operator with a reader performs certain operations or processes to targeted tagged objects. For example, a nurse wearing a wearable reader may interact with syringes, medicine and patients. Thus, the

interactions between the reader (operator) and object or combination of objects represent a certain operation. In this case, reader observations are used to track operations in which they imply the start or the end of operations.

Transaction – Transaction can be viewed from two perspectives. The first one refers to the business transaction in which the tagged object is involved. For example, the checkout involves a credit card transaction with many tags readings. The second perspective refers to the predefined properties of the objects, such as "processing steps", which can be written in writable tags. Here the latter perspective will be called object transaction. Hence, object transaction refers to the data written in the writable tags.

While the fundamental entities in RFID applications are themselves static, relationships among these entities can be either static or dynamic. Static relationships are spatial and temporal independent and they are similar to the traditional Entity-Relationship (ER) model [70]. For example, the relationships between an object and its corresponding proxy tag, an object and its on-board sensor, location and its sub-locations (location containments) are static relationships. Once set, they rarely change. However, most of the entity relationships in RFID applications are temporal and spatial dependent, which lead to dynamic data.

3.2 Dynamic and historical RFID Data

Dynamic entity relationships between fundamental entities in RFID applications generate movement, workflow, operations, and business logic [4], [67]. These interactions can be categorized as either an occurrence of an *event* or a *state* change. According to definitions provided in [71], an event is an instantaneous fact, i.e. something occurring at an instant, while state is something that has extent over time. Events delimit states. The occurrence of an event results in a fact becoming true; later, the occurrence of another event renders that fact no longer valid. Hence, events and states are dual; states can be represented by their delimiting events, and events are implied by states.
In RFID systems events are generated when entities interact with each other. RFID events include:

- **Observations** – these are generated when readers interact with objects.

- **Sensor Measurements** – these are generated when the sensor senses a target(e.g. motion, temperature).
- **Object transaction**– these are generated when a reader writes the data into the writable object's tag. For example, the reader may write the processing steps performed on the object to the tag attached to the object; in this case the transaction is the processing step.
- **Business Transacted items** – these are generated when an object participates in a business transaction.

On the other hand, state changes are generated when the entity relationships changes. State changes include:

- **Change of object location** – change of relationship between tagged object and location represents an object movement. Location changes come with the movement of objects in the business process.
- **Change of Object aggregation or containment relationships**
- **Start or end of an operations** – an RFID observation can be used to signify the start or end of an operation.
- **Change of reader locations** – change of relationship between reader and location. For example, movable readers enter/leave a location.

3.3 Modelling Static RFID Data

Static RFID data are modelled following the conventional ER model. Static entities and static relationships are modelled as a single state relation (a two-dimensional relational database table with the tuples as rows and the attributes as columns). For simplicity an *entity description* is used to refer to all static-related attributes of an entity. Each object, reader, sensor and location static entity is associated with RFID tag, which uniquely identifies it.

From the above list of static RFID data we have the following static entity tables.

- Object table – this table captures the unique identification code of the tagged object, and the object description.

- Reader table – this table captures the unique identification code of the reader and the reader description.
- Sensor table – this table captures the unique identification code of the sensor, and the sensor description.
- Location table – this table captures the unique identification code of the symbolic location, and the location description. A location containment table captures containment relationship between locations. For example, a large warehouse may have five loading dock doors, with each door identified as a separate location.
- Logical reader table – this table captures the unique identification code of the logical reader, unique ID of the location, and the logical reader description
- Transaction table – this table captures the unique identification of the business/object transaction, and the transaction description.
- Operator Table – this table captures the unique identification of an operator, the unique identification of the reader which acts as the operator's proxy and the operation description.

In this static model, if nothing changes in the reality, the relations remains unchanged; otherwise the state of the relation is updated using data manipulation operations such as insertion, deletion or replacement, which takes place as soon as they are committed. In this process, past states of the entity updated is discarded and forgotten. This type of model produces a snapshot database in which the state of the database at any particular point in time is modelled to reflect the current status or the last-known status of the reality. However, one of the essential goals of an RFID-enabled application is to trace objects, track objects or to monitor the system at any location, at any time, or both. For example, an RFID tracing application will require looking back in time at an historical list of all observations of an object. Tracking application will require looking forward in time and obtaining the most recent observation, while monitoring application looks at the current state of the system. RFID applications track and trace objects by keeping track of their movements among different locations through RFID observations. Tagged objects moving in the RFID deployed environment are automatically sensed and observed with their identifications, locations and movement paths recorded. While these movements are observed and recorded as sequence of observations, they signify process/movements in applications. This is another example of implicit semantic of RFID data which need to be

extracted and explicitly expressed in the data model. It also illustrates the importance of the data model which preserves the entity events and states histories for the successful execution of RFID applications.

In the static model, the history of events and states of the entities and their relationships are not considered. Therefore, in addition to the static model we also need a temporal based model which will effectively map the RFID observation into application logic in the real world while preserving the history of events and states generated from these observations.

3.4 Modelling Dynamic RFID Data

Time is an essential part of information about the constantly evolving real world. As time passes, researchers are interested in monitoring what is true at the present time, as well as in the past and future. The key observation is that while the present is continuously changing, most of the world description is remaining the same and the casual relationships among events and states of entities are embedded in temporal information. Therefore, facts or data need to be interpreted in the context of time. A time-based model provides a support to handle historical queries about past status, trend analysis for decision support systems as well as the way to represent retroactive and proactive changes occurring in the system [72], [73].

One way of dealing with temporal information is to add one or more time attributes to a model [72]. This approach was used here and, therefore, dynamic data in the middleware are modelled by extending the conventional ER model to include temporal information. ER constructs are made temporal by changing their semantics, i.e. the ordinary relationship types are given temporal semantics making their instances record variation over time, rather than just single states.

Since states have duration and events do not, the valid time of states is thus modelled using pair of time attributes $start\_time$ and end_time which represents the duration of the state. The valid times of events are modelled using an attribute $Tstamp$ which represent the occurrence timestamp of the event.

3.4.1 Temporal Events

Once again, in general, most sensor-based applications are not interested in individual readings in time or individual devices in space, but rather in an application-level concept of *temporal* and *spatial granules* [14]. These granules define the lowest-level, atomic unit of both time and space in which an application is interested [14]. Although RFID readers are able to produce data at a very high rate, applications are often concerned with data from a large time duration, or temporal granule. For instance, in a retail scenario when an application continuously monitors the count of items on each shelf, the temporal granule might be 5 seconds. Therefore, the "event" of interest in the application often means macro-event, which is defined as a holistic fact with duration, i.e. something occur over an interval of time taken as whole [71]. To support the notion of temporal granules in our model, observations are modelled as macro-events with start and end time. Within macro-event duration, the observations are aggregated and compared to detect obvious outliers. Modelling temporal granule aspect within the data model helps to reduce the volume of data generated by the RFID system and also provides a way of dealing with erroneous data.

Observation Events – Observation tables capture the raw read data generated by the readers at a certain location within specified time duration. For simplicity, single surrogate primary keys are used to identify the entities.

a) **In the read-only tag deployments** - It includes unique ID of the observation (objev_id), unique ID of the object (obj_id), start and end time of the macro-event, number of times the same tag was read (tag_count), and the location unique ID (location_id) which represent where the tag was read.

   $OBJECT\_EVENT$
   (objev_id, obj_id, start_time, end_time, tag_count, location_id)

   As explained before, the macro-event contains one or more instant tag reads, meaning that within that time the same tag can be read more than once by either the same reader or by more than one reader forming one logical reader in the location of interest. The attribute tag_count represent the number of times the same tag was read during the macro-event duration. This coalescing of the instant read tag values over the macro-event duration eliminates duplicate reads while reducing the amount of data accumulated by the

application and hence reduces the storage requirements. As will be explained later, tag_count also helps to detect erroneous tag reads.

b) **In the read-write tag deployments** – where reader can write data to the tag. The writing and reading actions are done independently, and writing can be more frequent than reading. When a reader interacts with an object for reading, the reader observes both the tag ID and the logged object transaction history. In this case, we have an additional table which keeps record of the logged object transactions, called the $OBJECT\_OTRANSACTION$ table. To link the object event instance with its list of object's object transaction instances, the object event unique ID is added into the list of object_otransaction entity table attributes. This attribute relates the object event table with the object's object transaction table in a one-to-many relationship with each other.

$OBJECT\_EVENT$

(objev_id, object_id, start_time, end_time, tag_count, location_id)

$OBJECT\_OTRANSACTION$

(object_transaction_id, objev_id, writer_id, value, Tstamp)

Each object's object_transaction table keeps record of its corresponding object event (objev_id), object transaction ID (object_transaction_id), the written data (value), the reader which wrote the information in the tag (writter_id), and the time when the information was written (Tstamp). It is important to keep the record of the reader that wrote the information, because the reader that wrote the data into the tag might not be the same reader that read the data. Also, the writer reader might be a proxy to operator, and such information can be used to add more semantics to the data collected.

c) **In the sensor-embedded tag deployment** – where the sensor embedded in the tag periodically senses the target object and writes the sensor measurements to the tag. When a reader interacts with an object, the reader observes both the tag ID and the logged sensor measurements history. In this case, there is an additional table that keeps records of the logged sensor measurements, called the $SENSOR\_DATA$ table. To link the object event

instance with its list of sensor data instances, the object event unique ID is added into the list of sensor data entity table attributes. This attribute relates the object event table with the sensor data table in a one-to-many relationship with each other.

$OBJECT\_EVENT$
(objev_id, object_id, start_time, end_time, tag_count, location_id)

$SENSOR\_DATA$ (sensor_id, objev_id, value, Tstamp)

The $SENSOR\_DATA$ table keeps records of the sensor measurements, including the primary key of its associated object event instance (objev_id), sensed value (value), and the time of sensing (Tstamp). If the sensor type is a location sensor then the sensor data is recorded in the $LOCATIONSENSOR\_DATA$ TABLE. This table keeps a record of the object's physical location coordinates (x, y, z), unique ID of the sensor which read the data (sensor_id), its corresponding object event instance (objev_id) and the time at which the data was read (Tstamp).

$LOCATIONSENSOR\_DATA$ (sensor_id, objev_id, x, y, z, Tstamp)

- **Business Transaction Events** – The business transaction event table keeps records of information about the business transaction in which the object participated. Its attributes include business transaction unique ID (business_transaction_id), object unique ID (obj_id) and time when the transaction occurred (Tstamp).

$BTRANSACTION\_OBJECT$ (business_transaction_id, obj_id, Tstamp)

3.4.2 Temporal States

As described before, state changes are generated when the entity relationships change over time. Also, state is associated with duration.

- **Change of object location.** The $OBJECT\_LOCATION$ table – this table keep record of object location history as the object moves from one location to another. Its attributes include object unique ID (obj_id), location unique ID (location_id), and the time duration [start_time, end_time] during which the object stays in that location. The combination of (obj_id, location_id, start_time) can be used as a

primary key to uniquely identify an individual object location instance or another single surrogate primary key can be added.

$$OBJECT\_LOCATION\ (\text{obj\_id}, \text{location\_id}, \text{start\_time}, \text{end\_time})$$

- **Change of containment/Association relationship.** The $CONTAINMENT$ table – this table keep records of object containment history as their containment or association relationships changes over time. Its attributes includes object unique ID (obj_id), parent object unique ID (parentObj_id), and the time duration [start_time, end_time] during which the object is contained in its parents object.

$$CONTAINMENT\ (\text{obj\_id}, \text{parentObj\_id}, \text{start\_time}, \text{end\_time})$$

- **Operation.** The $OPERATION$ table – this table keeps records of the operations performed by an operator. Its attribute includes the unique ID of the operator who performed the operation (operator_id), the object being operated (obj_id) on, and the life span of the operation [start_time, end_time].

$$OPERATION\ (\text{operator\_id}, \text{obj\_id}, \text{start} - \text{time}, \text{end\_time})$$

- **Change of reader locations** – The $READER\_LOCATION$ table – this table keeps record of reader location as it is moved from one location to another. Its attributes includes unique reader ID (reader_id), unique location ID (location_id), and the duration of the reader in that location [start_time, end_time].

$$READER\_LOCATION\ (\text{reader\_id}, \text{location\_id}, \text{start} - \text{time}, \text{end\_time})$$

The above presented model extends the conventional ER construct by adding time attribute to the model. The ordinary relationship types are given temporal semantics making their instances record variation over time, rather than just single states. By maintaining the history of events and state changes, the data model captures the fundamental RFID application logic into data model itself. Although state changes information can be derived from events data, explicitly modelling of the state changes information into the data model provides a better support for complex queries. Most of the RFID queries are time based queries with temporal constraints such as history, temporal snapshot, temporal slicing, temporal joins and temporal aggregates. History queries retrieve the history information of an object; for example, location history, temperature measurements history, and objects transaction. Snapshot queries retrieve the snapshot information of an object; for example, location at time t, and temperature at time

t. Temporal slicing queries retrieve the information of the object during the time interval (t1, t2). Temporal join queries find information by joining multiple relations on a certain constraint. Temporal aggregation queries summarize aggregation information at certain snapshot or interval. Examples of basic queries can be found in [67].The above model efficiently supports all these types of queries. The semantics of the data model is generalized from RFID data, thus fits directly with RFID data.

Table 3.1 summarizes the RFID data model tables for different types of RFID deployments. The same information is also summarized in Figure 3-1, Figure 3-2, and Figure 3-3 which shows the event and state dynamic relationships generated by interaction of static entities within the RFID system. Note that not all tables are necessary for a given deployment.

Table 3.1: RFID Data Model Tables for Different Deployment Types

Deployment Type / **Table**	**Read-only tags**	**Read-write tags**	**Read-write tags with onboard sensor**
Static Entity Tables:			
OBJECT	X	X	X
READER	X	X	X
LOCATION	X	X	X
LOCATION_CONTAINMENT	X	X	X
OTRANSACTION		X	X
BTRANSACTION	X	X	X
SENSOR			X
OPERATOR	X	X	X
Dynamic Event Tables:			
OBJECT_EVENT	X	X	X
OBJECT_OTRANSACTION		X	X
SENSOR_DATA			X
LOCATIONSENSOR_DATA	X	X	X
BTRANSACTION	X	X	X
OPERATION	X	X	X
Dynamic State Tables			
OBJECT_LOCATION	X	X	X
CONTAINMENT	X	X	X
READER_LOCATION	X	X	X

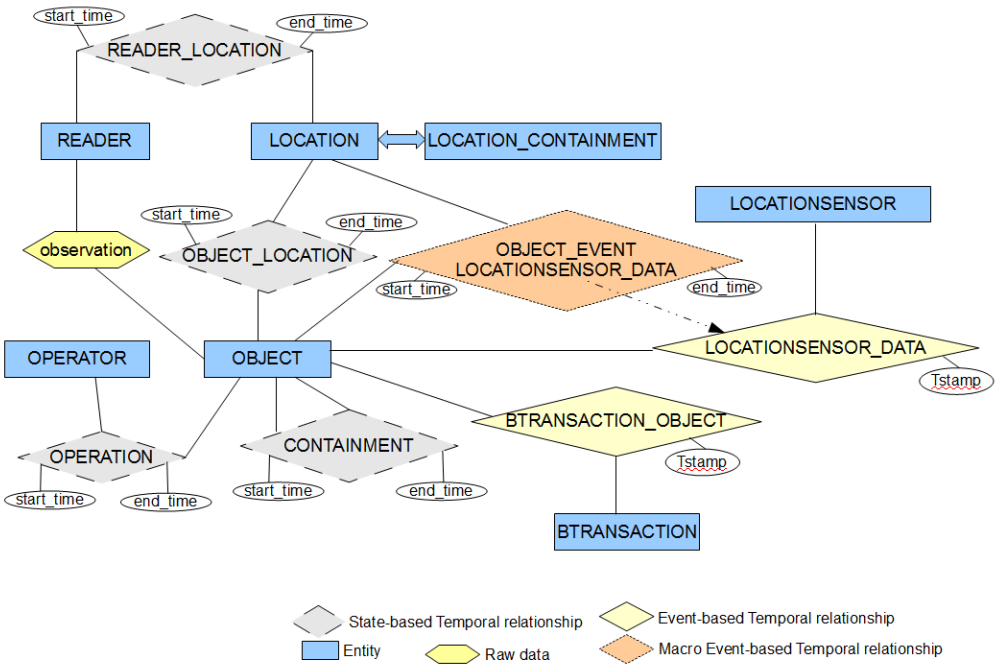


Figure 3-1: Data Model for read-only tags RFID deployment

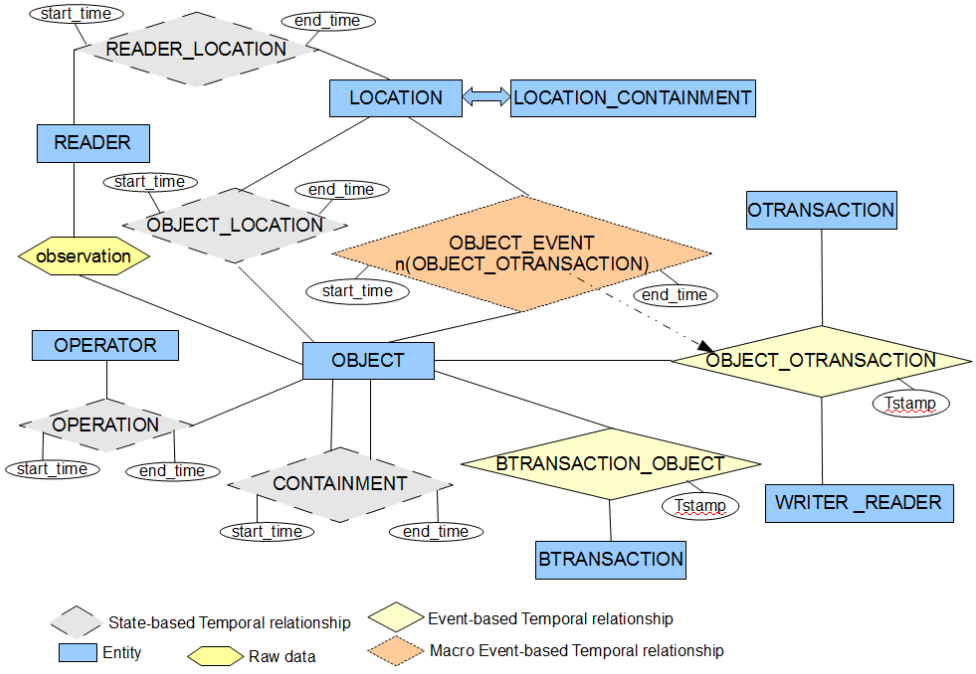


Figure 3-2: Data Model for read-write tags RFID deployment

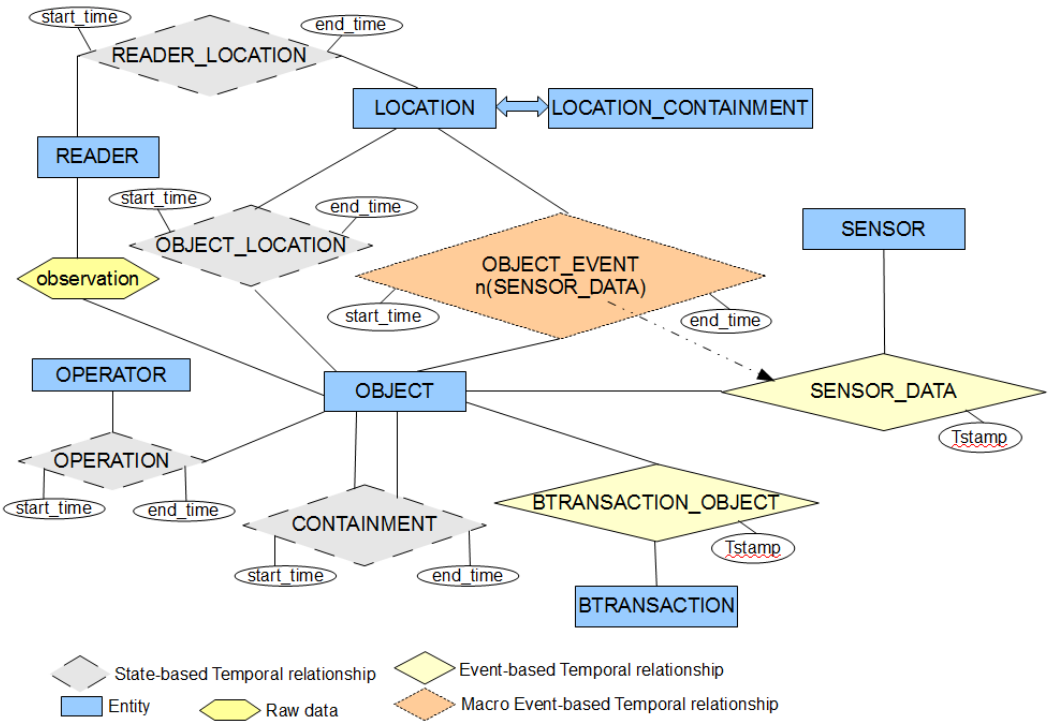


Figure 3-3: Data Model for read-write tags with onboard sensor RFID deployment

3.5 Application Level Events Processing

The middleware developed uses the above data model and some rules to automatically transform the raw data generated by RFID system into higher semantic data which are able to support most of the RFID applications queries. The extracted higher semantic data such as location change, containment change or operation are captured into the data model as state change information.

An assumption is that the user specifies the temporal granule of the event as the macro-event duration (e.g. read tags for 1 minute), as well as the location or the logical reader at which the observation has to be performed. As explained before, a logical reader/location can consist of one or more readers.

3.5.1 Observation Events

Readers vary in their capabilities and functionalities. Some readers are capable of performing some aggregation and filtering functionalities while others simply read the

tags and report the raw read data without any aggregation or filtering. Therefore, in order to cater for low range readers, the middleware provides a functionality which performs the aggregation and filtering of tag reads within the macro-event duration.
The following algorithm for aggregating tags collected during the micro-event duration is explained in section 3.5.1.1 below.

3.5.1.1 Aggregation algorithm – for a one reader

```
Initialize count = 0 // number of tag reads received from the reader
Initialize tagEventList = new ArrayList() // list of tag events
While (micro-event duration) {
    read tag
        create a new tag event object
        set scanned input value as tag_id
        readTime = new Date() // capture the read time
        set the tag discovery time as readTime
        set the tag last seen time as readTime
    update tag event
        if (count>0)
        //check if the tag already exists in the list
            If (not in the list)
                initialize its tag_count = 1
                add tag to the list
            else
                increment its tag_count -> tag_count + 1
                update its lastSeenTime -> readTime
        else (count = 0) → this is first tag add it in the list
            initialize its tag_count = 1
            add tag to the list
        increment count
    }
```

After the micro-event duration each reader produces data with the reader unique ID and list of its tag events

Reader observation → (reader_id, tagEventList)

Where,

$$\text{tagEventList} = \{\text{tgev1}, \text{tgev2}, \text{tgev3}\ldots\}$$
$$\text{tagev} = (\text{tag\_id}, \text{discoveryTime}, \text{lastSeenTime}, \text{tag\_count})$$

tag_id	Is a unique ID of a tag
discoveryTime	Is the first time the tag was read by the reader
lastSeenTime	Is the last time the tag was read by the reader
tag_count	Number of times in which this tag was read by reader during event duration

3.5.1.2 Aggregation for more than one reader

The data from readers which form a logical reader or the readers which are deployed in the location of interest are then combined together to generate event data. The assumption here is that the client of the data is interested in the location where the object was found rather than the reader which observed the object.

$$\text{Event\_Observation} \rightarrow (\text{location\_id}, \text{combined\_tagEventList})$$

If there are two or more reader data instances with the same tag_Id in their tag event list, their tag_count are added to form new aggregated tag_count, last seen time is set to the greater last seen time, and discovery time is set to be the minimum discovery time.

$$R_x \rightarrow \text{tagEvent}_x(\text{tag\_id}_x, \text{discoveryTime}_x, \text{lastSeenTime}_x, \text{tag\_count}_x)$$
$$R_y \rightarrow \text{tagEvent}_y(\text{tag\_id}_y, \text{discoveryTime}_y, \text{lastSeenTime}_y, \text{tag\_count}_y)$$

$$\text{If tag\_id}_x = \text{tag\_id}_y$$
$$\text{Then} \rightarrow \quad \text{tag\_count} = \text{tag\_count}_x + \text{tag\_count}_y$$
$$\text{discoveryTime} = \min(\text{discoveryTime}_x, \text{discoveryTime}_y)$$
$$\text{lastSeenTime} = \max(\text{lastSeenTime}_x, \text{lastSeenTime}_y)$$

Since readers, containers and even some locations can also be attached with tags to uniquely identify them; the RFID Tag table has an attribute called $identifierType$ of enumeration data type which describes what type of entity it identifies. The enumeration data type is:

ENUM('OBJECT', 'READER', 'SENSOR', 'LOCATION', 'CONTAINER', 'OBJECTandCONTAINER').

Where,

OBJECT, READER, SENSOR and LOCATION identify their respective entities; and

$CONTAINER$ – identifies any object which can contain another object e.g. toolbox; and

$OBJECTandCONTAINER$ – identifies any entity which can be both container and object. For example, cases and pallets: case contains items which are objects; when loaded into a pallet, cases are viewed as objects and pallet as container, and when pallets are loaded into truck they are viewed as objects while truck becomes a container of the pallets. This concept can be referred to as nested containment.

This attribute identifies which type of entity is observed and it also aids in deciding state changes to be triggered by the occurrence of an event.

Note that nested containments are difficult to resolve; for example, where more than one container object is observed and the list of objects, it is difficult to know which container an object belongs to. This implementation assumes that the containment concept is used in a structured environment such that only one container is observed at a time with no nested containment.

From the event observation ($location\_id, combined\_tagEventList$), object events are generated from the tag events whose tag's identifierType is either OBJECT or CONTAINER. These events are stored in the OBJECT_EVENT table of the data model discussed previously.

OBJECT_EVENT
($objev\_id, object\_id, start\_time, end\_time, tag\_count, location\_id$)

Where,

$Object\_id = tag\_id$ ($with\ tag's\ identifierType = OBJECT \mid\mid CONTAINER$)
$Start\_time = discoveryTime$
$End\_time = lastSeenTime$

3.5.2 State changes Events

As explained before, events delimit states. The occurrence of an event results in a fact becoming true; later, the occurrence of another event renders that fact no longer valid. Hence, occurrence of an event triggers state changes.

a) **Location change** – Any tag event in the event observation ($location\_id, combined\_tagEventList$) whose tag's $identifierType$ is OBJECT or CONTAINER triggers an update of object location. If tag's $identifierType$ is READER it triggers an update of reader location and if it is a SENSOR it triggers an update of sensor location.

For example, if the last object location is the same as the event location, update end_time of the location to the end_time of the event, otherwise new object location tuple is added to the OBJECT_LOCATION table with $start\_time$ and end_time corresponding to event start and end times.

For each new OBJECT_EVENT tuple

→UPDATE: OBJECT_LOCATION

If object last location = event location

→Update object location end time

$$object\_location.end\_time = event.end\_time$$

Else add new

$object\_location(obj\_id, location\_id, start\_time, end\_time)$

b) Containment change – From the event observation ($location\_id, combined\_tagEventList$), if there is any tag event whose tag's identifierType is CONTAINER, it triggers the object containment change with parent object being the container object and the children object being other tag's in the event whose $identifierType$ is OBJECT.

For each new event observation with tag event whose tag's

$identifierType = CONTAINER$

→UPDATE: CONTAINMENT ($obj\_id, parentObj\_id, start\_time, end\_time$)

If object last parent object = event container

→Update containment end time

$$containment.end\_time = event.end\_time$$

Else add new containment

$$(obj\_id, container\_id, start\_time, end\_time)$$

c) **Operation** – From the reader observation $(reader\_id, tagEventList)$, if this reader is associated with OPERATOR entity, this observation triggers changes in OPERATION table, with operation objects being the objects in the tag event list with the tag whose $identifierType$ is OBJECT or CONTAINER.

For each new reader observation whose reader is associated with operator

→UPDATE: OPERATION $(operator\_id, obj\_id, start\_time, end\_time)$

Chapter 4: Middleware System Design

What is presented in this chapter is the prototypical design and analysis of the proposed agent-based RFID middleware system, called RDDM (**R**FID **D**evice and **D**ata **M**anagement) middleware. The middleware abstracts the Auto-ID applications from physical RFID specific details and provides necessary services such as device management, data cleaning, event generation, query capabilities and event persistence.

The prototype system is designed by using the agent oriented software engineering methodology PASSI [40], and implemented using the Jade Platform [41] and a MySQL relational database. One attractive feature of the software agents and multi-agent systems (MAS) is their ability to represent a complex software system as a modular society of cooperating autonomous problem solvers. MAS is a mature technology with a proven potential to enhance overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse [33]. On the other hand, most of RFID system deployments are distributed in nature, RFID readers or tagged devices installed in different strategic locations within the organizations are linked together creating a distributed network of RFID devices. We argue that using of MAS distributed architecture approach for the middleware provides a scalability and load balancing through distribution of the computing resources and capabilities across a network of interconnected agents. The MAS modular approach also provides an adequate level of reuse of the middleware or its components to facilitate development of the Auto-ID applications. Using MAS for a middleware system development is what makes this middleware system different from other research-based RFID middleware systems. A detailed review of existing RFID middleware solutions can be found in [74] and [75].

This prototype system assumes the read-only deployment using read-only passive RFID tags with no other types of sensors. Only the unique identifier of the tagged object is stored in the tag and all object related data are stored in the back-end information systems. The main advantage of such a system is that these types of tags are relatively cheaper and there is no need to encrypt the simple identifier because access to the data on the network is restricted. Because of its modular temperament, the system can be extended to include other data entities and accommodate different types of RFID

deployments and applications. For example, to add support for another type of sensor a proxy agent for that sensor should be modelled and added to the system. The middleware also uses a discrete (symbolic) location model in which the location of the object can either be identified by a fixed reader's location or with some special tags which are fixed to represent a location. Fixed reader's locations can be used to identify the position of moving object, while fixed location tags can be used to identify location in the case of a mobile reader.

4.1 System Requirement Analysis

Based on an analysis of different RFID applications and the study of other work on RFID middleware [3], [5], [6], [8], [15], [16], an RFID middleware system should meet the following requirements:

- Support for heterogeneous reader landscape.

  The diverse computing and networking capabilities of readers is also an important RFID consideration when developing RFID infrastructure support. Low cost readers usually support only a single antenna and a serial RS232 interface. These reader types are connected to a computer which hosts the application directly or forwards the captured data over a network connection. More sophisticated reader devices support several antennas, a TCP host interface, and ample computing resources for on-device data processing, such as filtering and aggregation.

- Fault and configuration management.

  The proliferation of readers mandate fault and configuration management. This includes monitoring the health of RFID readers and accessing the RFID reader configurations remotely.

- RFID data filtering.

  A common feature of all applications that make use of the captured data is the desire to receive filtered RFID events rather than all RFID data captured. Different applications are interested in a different subset of the total data captured, based on the reader, reader antenna, and tag involved.

- RFID data aggregation.

RFID systems generate significant amount of data that can be aggregated in a number of different ways. RFID data can be aggregated in the time or space domain, e.g., by combining data from different readers and reader antennas that observes the same location or by detecting the movement of a tagged objects. Since RFID permits identification at the instance-level rather than at the class-level, there is also the possibility to report the quantity of objects belonging to a specific category.

- RFID data interpretation.

From an application perspective, it is also desirable to provide a mechanism that interprets the captured RFID data in a given business context and that turns the low level RFID event into the corresponding business events. For example, the detection of a tag by reader R1 followed by reader R2 can be translated as a person leaving the office.

- RFID data dissemination.

The information captured by a reader is usually of interest not only to a single application, but to a diverse set of applications across an organization and its business partners. The captured RFID data must thus be broadcast to the entities that indicated an interest in the data. Due to the event-driven nature of many processes observed with the help of RFID systems, there is a need to support asynchronous messaging as well as a query-response model. Different applications also require different latencies. Applications that need to respond immediately to local interaction with the physical objects require a short notification latency that is comparable to the observation latency. Legacy applications that are not designed to handle streaming data might need to receive batched updates on a daily schedule.

- Persistency of event data.

The data received by RFID readers need to be stored persistently to allow applications to query the data. For example, an application that wants to retrieve the track and trace information about a certain object needs to query all related historical RFID events related to the object.

- Location information.

  The system has to provide a mechanism to enrich RFID data with location information since applications are interested when a certain object was at a certain location. For many applications, symbolic location names are sufficient in business context definitions. For example, the query of a track and trace application should return a list of times, locations and object states for a given object.

To model this system, the first step was identifying and analyzing the system stakeholders and their intentions. Stakeholders are modelled as social actors who depend on each other for goals to be achieved, tasks to be performed and resources to be furnished. Intentions are modelled as goals which are later grouped together to form system functionalities.

4.1.1 System Stakeholder

The main stakeholders for the developed prototype system are:

- Client applications – these are enterprise software applications which need the data captured by the RFID system.
- Administrators – these are individuals who are responsible for installation and configuration of the middleware system, as well as monitoring the performance of the system.
- Readers – these are infrastructure components responsible for acquiring raw data from RFID tags and forwarding them to the middleware system.

Figure 4-1 shows the actor diagram for the RDDM middleware system with their primary goals. A reader's goal is to collect data from RFID tags and forwarding it to the middleware; client applications want to receive RFID event reports and administrators want to configure the system.

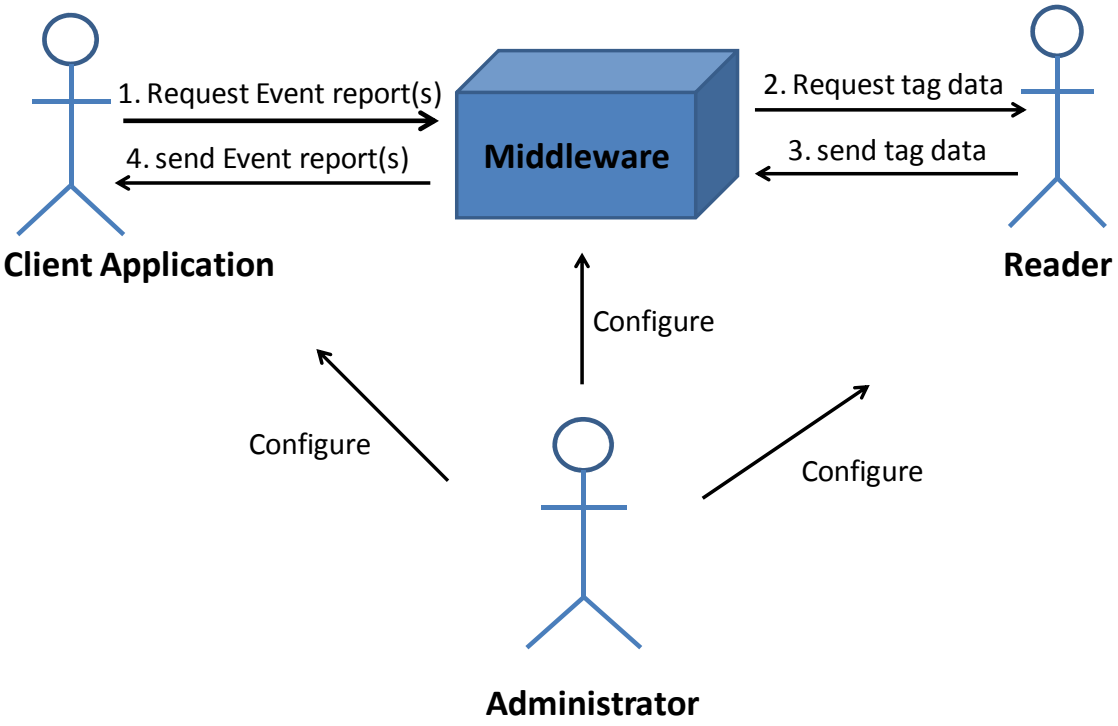


Figure 4-1: Actor diagram modelling the key stakeholders of the RDDM middleware system

4.1.2 System Architecture

Figure 4-2 shows the proposed RFID middleware architecture with four main layers: device management layer, data management layer, business process interpretation layer and an interface layer. The architecture is motivated by the analyzed system requirements and system stakeholders.

Device Management Layer

The device management layer deals with management of different types of RFID devices deployed in the system. The device adapter acts as a device proxy in the software and implements the communication protocols of the reader. The device adapter communicates with the physical RFID readers to collect the read data. The collected data is then filtered to reduce erroneous readings before being transferred to the data management layer for further event processing. The device monitor is used to monitor the pertinent health information of the connected devices. The parameters to be monitored can include IP address, port number, connectivity status and operation status. The device configuration manages the information about the devices which are deployed and registered in the middleware.

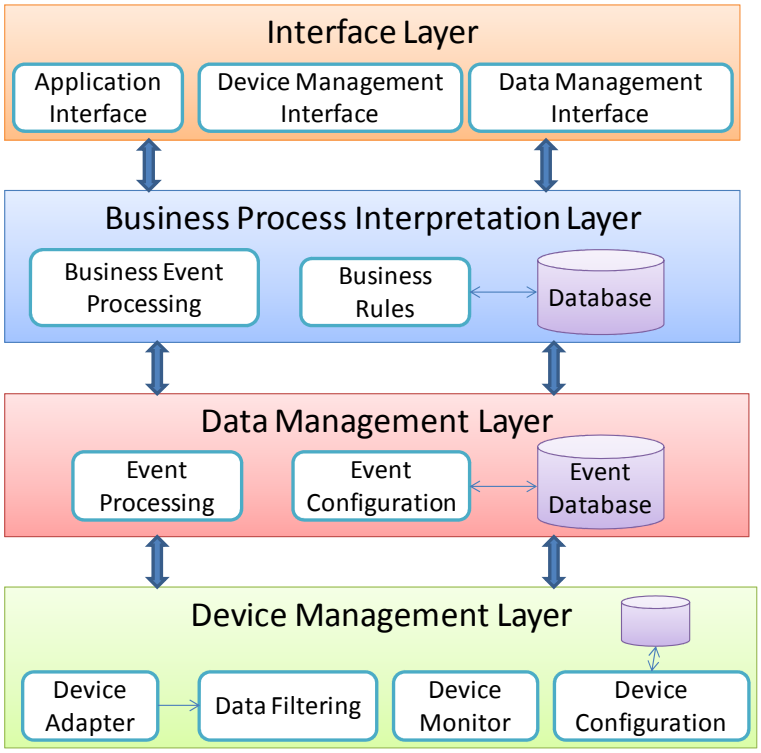


Figure 4-2: System Architecture

Data Management Layer

The data management layer is responsible for processing the data collected by multiple RFID devices. This layer contains several data processors such as filters and aggregators which are used to generate events according to events configuration as requested by the data client. Filter operations performed in this level are of much higher level than the ones performed in the physical level. The physical layer filter involves reducing the read errors while the data layer filter involves selecting a particular type of object based on given conditions. In this level the data can be filtered based on the reader, reader antenna or the tag involved. Data can be aggregated based on time, space or object type. The generated events are forwarded to the next layer for further processing and also persistently stored in the database to facilitate application query processing.

Business Process Interpretation layer

This layer is responsible for extracting meaningful business events from the raw RFID data event. The event data from the lower layer is associated with existing business domain logic whereby the data is aggregated, transformed and enriched with business context information. To facilitate generation of business events the status and tagged

object related information such as object location, aggregation information, the environment of the tagged object and inference and association rules are maintained in the database.

Interface layer

The interface layer connects the system with the real world; it provides a graphical user interface GUI allowing the system administrator or user to interact with the system and also it provides a means for integrating the middleware system with other enterprise applications interested in the data. By using the device management interface the user can specify, configure and monitor the set of devices connected to the middleware. By using the data management interface the user can define the rules for processing the incoming data collected by the devices and subsequently define where to send the information after processing. The application interface is responsible for integrating the middleware with other enterprise applications interested in RFID data.

4.2 System design

This middleware system is designed following agent oriented software engineering (AOSE) methodology. AOSE is concerned with the engineering of software that has the concept of agents as its core computational abstraction. Several methodologies exist for the analysis, design and implementation of agent oriented software. An author of [37] offers a very good guide to the broad body of literature on AOSE and comparison of various methodologies can be found in [38]. Among the AOSE methodologies which exist we choose to use PASSI methodology [40] for our system development. This choice is driven by the availability of step-by-step requirement-to-code guidance, documentations and easy-to-use design tools offered by this methodology. PASSI has also been used successfully in several projects in both robotics and information systems [76].

PASSI is comprised of five models (System Requirements, Agent Society, Agent Implementation, Code Model and Deployment model) which include several distinct models as shown in Figure 4-3. PASSI methodology includes the use of ontologies and communications with a FIPA compliant structure.

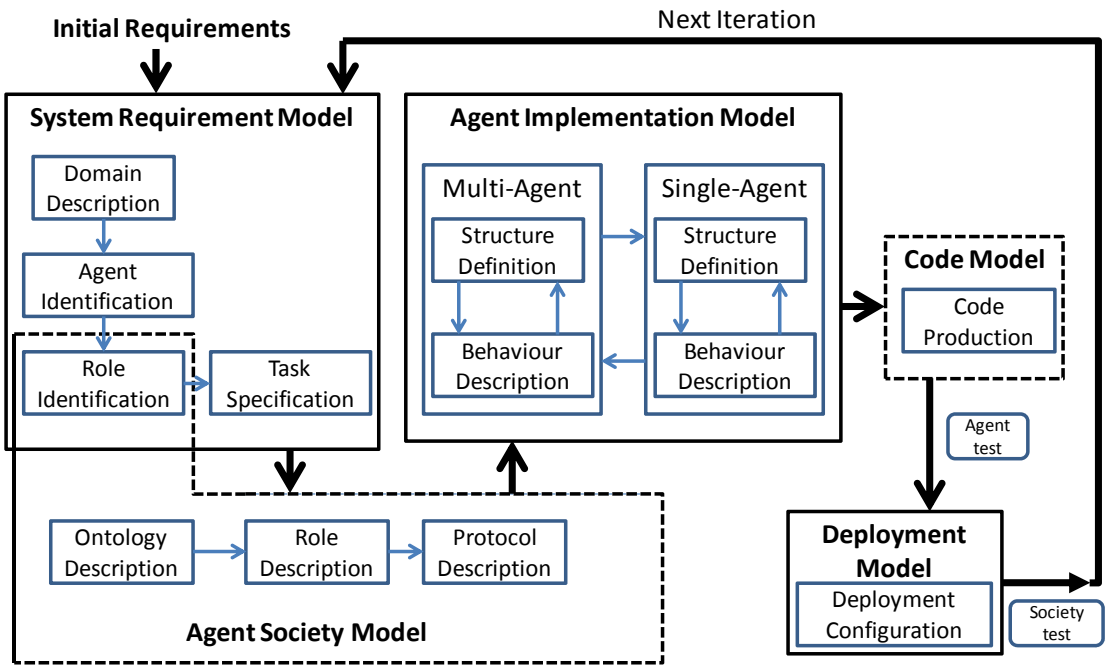


Figure 4-3: The models of the PASSI methodology [40]

4.2.1 Domain Description

The functionality of the system is first captured through a series of use case diagrams employing classical object oriented methods or through the informal application of scenario-based methods [40]. From our system requirement analysis conducted on the previous section, we modelled our RDDM middleware system functionalities in the domain description diagram shown in Figure 4-4. The stakeholder's goals are delegated to the system-to-be functionalities as it was elicited in the requirement analysis phase.

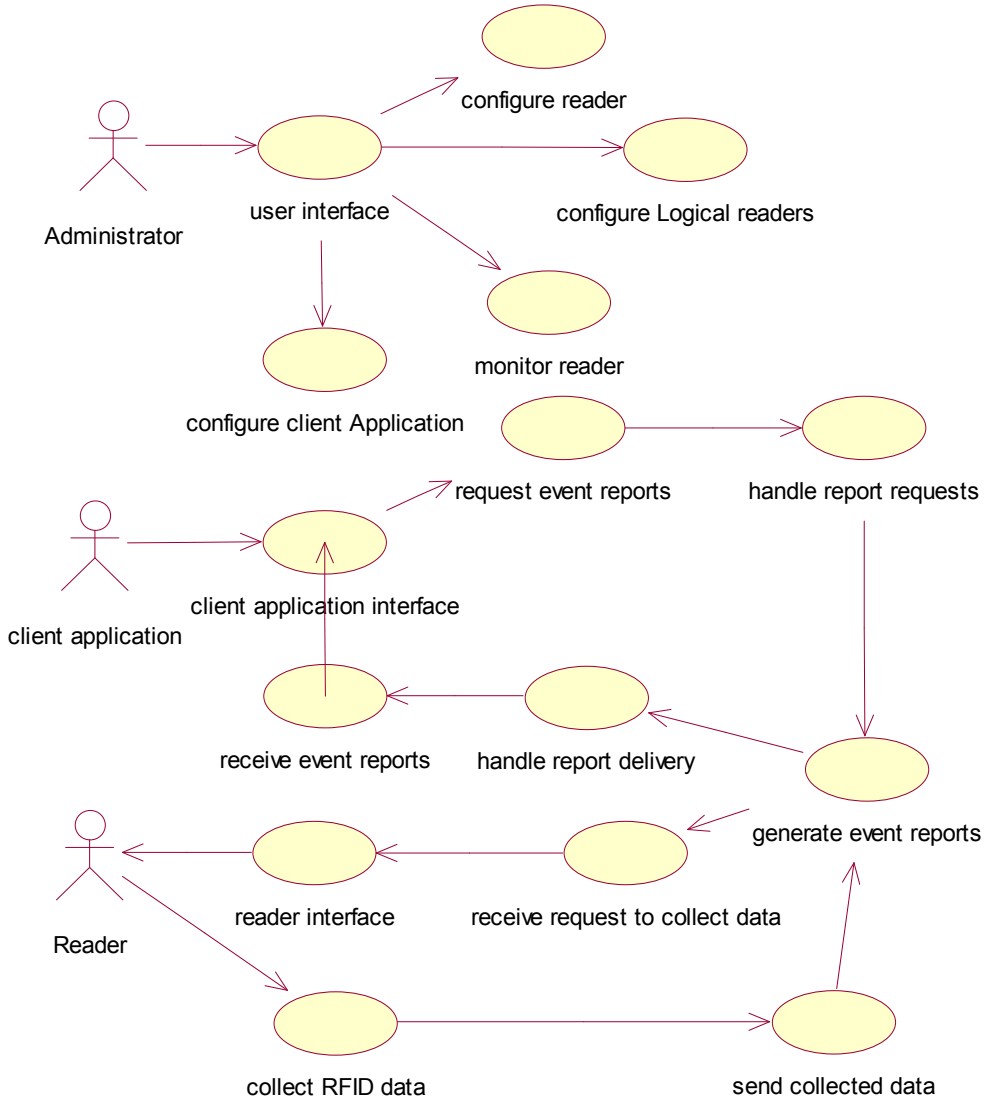


Figure 4-4: Domain Description diagram - functionalities of the RDDM middleware system

4.2.2 Agent Identification

Following PASSI methodology agent identification is performed early in the development process. This decision is motivated by the perception of MAS as a heterogeneous society of intended and existent agents that according to Jackson's terminology can be "bidden" or influenced but not deterministically controlled [76]. From this point of view, it is therefore more reasonable to divide the description of required system functionalities into units of responsibility from the start. Hence, agent identification is performed by grouping some system functionality into one agent.

According to definition of agent by Jennings *et al*. in [33], it is possible to see an agent as a use case or package of use cases [40]. Therefore, agent identification starts from the use case diagram (Figure 4-4) of the previous step which describes the decomposition of system functionalities. One or more use cases are grouped into different packages forming a new diagram (Figure 4-5), called an agent identification diagram. Each package defines the functionality of a specific agent with the package name representing an agent name, while the system stakeholders (which are external entities interacting with our system, i.e. administrator, client application and reader) are represented as actors. The relationships between use cases of different agents are stereotyped as "communication".

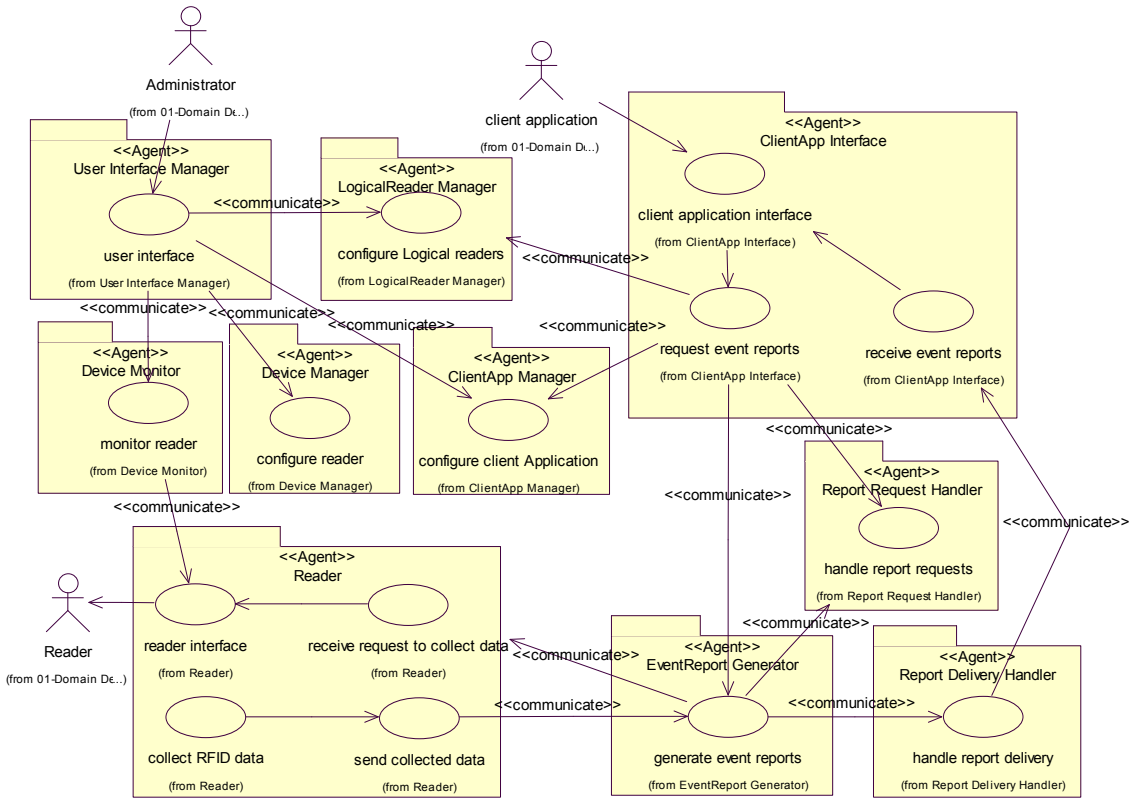


Figure 4-5: Agent identification diagram

User Interface Manager agent

The *User Interface Manager agent* is an interface agent which interfaces the administrator or system user with the RDDM middleware. It provides a graphical user

interface for a user to configure logical readers, set reader configuration, monitor operation status of the device or configure client applications.

Device Manager Agent

The *Device Manager agent* is responsible for managing persistent device-specific information about all the devices that are deployed and registered in the middleware. Examples of such device-specific information include the device name, model number, protocols that the device support, operating frequency, network protocols, power level, connectivity, active antennas and additional information for operating a particular device.

Device Monitor Agent

The *Device Monitor agent* is responsible for monitoring the operational status of the device together with network connectivity. It keeps real-time pertinent heath information of each reader connected on the system. Parameters monitored include: reader connectivity status; automatically discovery of the connected devices with their IP addresses or port numbers; tag ID, discovery time and tag read count; and process events and alarm generated in case of unusual operational status.

Logical Reader Manager Agent

The RFID readers generally have multiple antennas connected to it. These antennas may be placed in groups at different locations for tracking purposes. A group of antennas at one tracking location may belong to single or multiple readers. There may also be readers whose antennas are part of various groups allowing a single reader to be a part of multiple tracking locations. Our middleware renders this functionality by introducing a notion of Logical Reader and modelled as *logical reader manager agent*, which allows us to combine multiple reader-antenna pairs with a single logical purpose. The Logical Reader is a virtualization of a tracking location and is used to associate multiple reader-antenna pairs to the tracking location.

Client Application Manager Agent

The *Client Application Manager agent* is responsible for managing persistent information about client applications that interact with the middleware. The information includes the name, notification address and message format.

Client Application interface Agent

Each enterprise application interested in receiving RFID event data from the middleware is represented with a transducer agent referred to as *Client Application interface agent.* The *Client Application interface Agent* is responsible for interfacing the enterprise application with the middleware. It encapsulates the client application APIs and provides client application with the common APIs to specify, in high-level and declarative way what type of data they are interested in receiving and to notify the client when the data is delivered. In this way the middleware can receive requests using different communication protocols (such as XML-RPC or SOAP-RPC) and translate those requests into agent communication language (ACL) to be processed by the other agents within the middleware.

Report Request Handler Agent

The *Report Request Handler Agent* is responsible for handling the report requests made by client applications. It maintains the client subscriptions and the current status of the request. An event report request also referred to as $eventSpec$ defines how the event report is to be generated. It contains the list of logical readers whose data read are to be included in the event data, a specification of how the boundaries (beginning and the end) of the event are to be determined and the specification which describe how the report should be generated from the event data. During its execution cycle, the $eventSpec$ transits in three states: unrequested, requested and active. Once defined, the $eventSpec$ state is said to be "unrequested"; when the client subscribes to that $eventSpec$, the status changes from "unrequested" to "active" state; and in this last state the read data from readers accumulates into event data based on an event boundary specification (duration time). When the event duration time elapses the accumulated event data is sent to a generate report and the $eventSpec$ status changes from the "active" to "requested" state until the repeat period of the event boundary specification elapses, in which case the status changes from "requested" to "active". The accumulation of event data from readers then starts again based on the time duration.

Report Delivery Handler Agent

The *Report Delivery Handler Agent* is responsible for handling the delivery of generated event reports to the client in an appropriate format. It receives the event reports in ACL

and translates them to the format understandable by the client application before sending it to the client. It also monitors the delivery of the reports and handles exceptions.

Event Report Generator Agent

The *Event Generator Agent* is responsible for generating event reports requested by client applications. The agent contains several data processors such as filters, aggregators and buffers. These data processors with appropriate processing rules and algorithms are responsible for receiving tag reads from readers, filtering the incoming detected raw data, aggregating multiple incoming events into one higher–level event, temporary storage of observed events and translation of cleaned filtered and aggregated events into appropriate format as requested by the client.

Reader Agent

Each reader device to be included in middleware is represented with a transducer agent referred to as *reader agent.* The *reader agent* acts like an interface between a reader device and other agents in the middleware system. The *reader agent* encapsulates the reader manufacturer's APIs and provides a common API to access the reader device. Therefore, adding support for a new type of a reader is a matter of adding new reader agent in the middleware and does not require modification to any other part of the middleware.

4.2.3 Role Identification

In PASSI methodology, the role identification phase is considered to be part of both the System Requirements and Agent Society models (as shown in Figure 4-3). Role identification is a functional/behavioural description of the agent and therefore part of the System Requirement model, but it is also a representation of its relations with other agents and as a consequence a part of the Agent Society model [40].

A sequence diagram in UML is a kind of interaction diagram that shows how processes operate with one another and in what order. They are also called event diagrams, event scenarios and timing diagrams. We use sequence diagrams to identify the roles played by agents when interacting with other agents within the system.

In general, one sequence diagram is drawn for each inter-agent communication depicted in the agent identification use case diagram (Figure 4-5). These sequence diagrams describe the important use case scenarios occurring within the system. Several roles that an agent can play are introduced as objects in the appropriate sequence diagram using the syntax: <name of the role> :< name of the agent>. An agent may appear in several scenarios playing distinct roles in each, and may even appear playing more than one role in the same sequence diagram. The message exchanged between different roles in the sequence diagram represents either an event generated by the external environment or system, or parts of communication between the roles of one or more agents. A message specifies what the role is to do and possibly the data to be provided or received. Data and tasks that are mentioned in the sequence diagram are specified in more detail later in the Domain Ontology Description (DOD) and Role description diagrams, respectively. Hence, for each message in a sequence diagram, a corresponding relationship should appear in the DOD diagram together with a description of knowledge exchanged or shared.

Figure 4-6 and Figure 4-7 shows examples of scenarios and sequence diagrams used to identify the agent roles of the middleware system.

Scenario 1: Query device operation status

This scenario shown in Figure 4-6 illustrates how the system user interacts with the system agents playing different roles to query for the current operation status of the device.

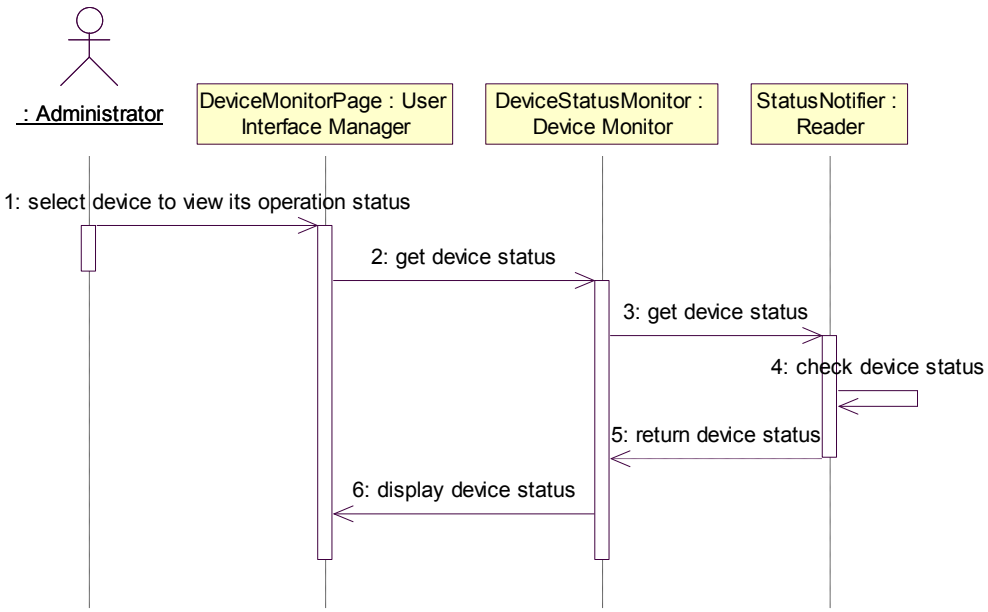


Figure 4-6: The Role identification Diagram for the scenario in which the administrator wish to view the operation status of the device

Scenario 2: Unsubscribe from receiving event report

This scenario shown in Figure 4-7 illustrates how the system client interacts with the system agents playing different roles to unsubscribe from receiving event reports.

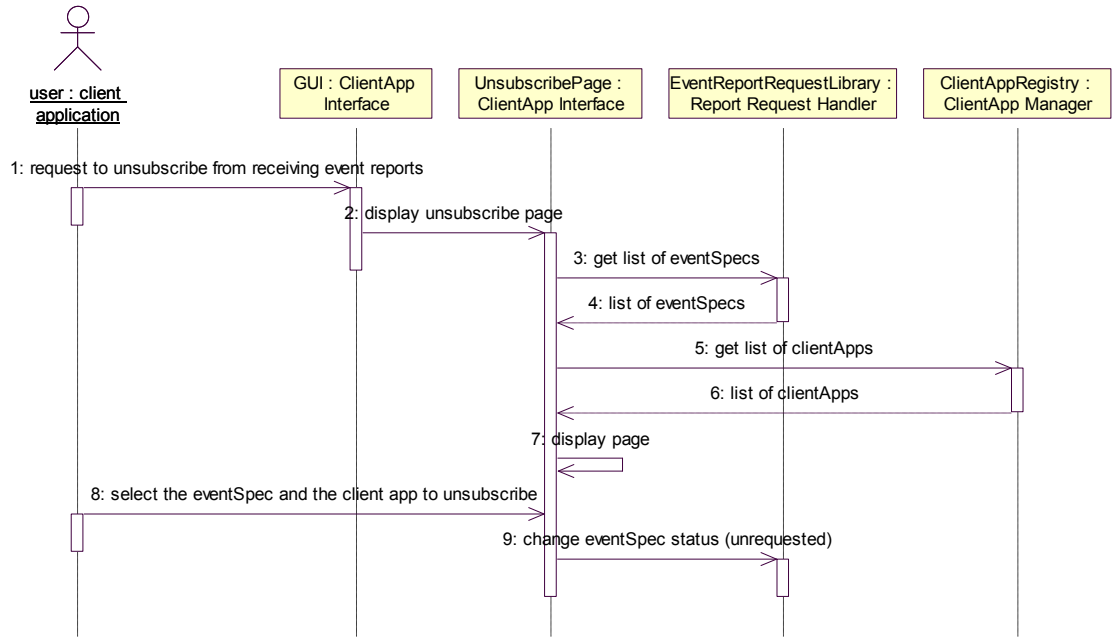


Figure 4-7: The Role identification Diagram for the scenario in which the client wishes to unsubscribe from receive event reports

4.2.4 Task Specification

In the task specification phase, the focus is on each individual agent's behaviour by decomposing it into tasks that encapsulate some functionalities described in previous steps. Task specification diagram summarizes what an agent is capable of doing, ignoring information about the roles that an agent plays when doing particular tasks. The scope of the diagram is limited to capabilities of the single agent viewed as a social problem solver. Therefore, for each agent we develop an activity diagram containing all activities carried out by that agent and its interaction with other agent's tasks to reach its goal.

The relationship between activities signifies either messages between tasks and other interacting agents' tasks or communication between tasks of the same agent. Communications between tasks of the same agent are not speech acts, but rather signals addressing the necessity of beginning an elaboration; that is, triggering a task execution or delegating another task to do something.

Information needed to produce the agent's task specification diagrams comes from role identification sequence diagrams. We examine all the agent's role identification diagrams and explore all of the interactions and internal actions that the agent performs to accomplish a scenario's purpose. From each role identification diagram we obtain a collection of related tasks, and grouping them together appropriately results in the task specification diagram.

Figure 4-8 shows the task specification diagrams of the *Event Report Generator* agent. Each task specification diagram is divided into two sections by means of two swim-lanes: the right one contains the specific agent's tasks, and the left one contains the tasks of the other interacting agents in order to represent relationships of this agent with the others.

One listener task is introduced in every agent in order to pass incoming communications to appropriate tasks. A task is also introduced for each incoming and outgoing message in the Role identification sequence diagrams and is connected to a listener. Other tasks arise from the decomposition of agent's functionalities to either facilitate reuse, perform calculations or access to external devices.

For example, Figure 4-8 shows the tasks that constitute the *Event Report Generator* agent capabilities committed to the process of generating event reports. The

$receive\_immediate\_request$ task handles the incoming message from the $ClientApp\ Interface$ agent that requests to receive the event report immediately, while the $receive\_subscribe\_request$ tasks handles the request from $ClientApp\ Interface$ agent requesting to subscribe to receive event reports after every certain period of time. The $request\_reader\_to\_collect\_data$ task communicates with the *Reader* agent requesting it to collect tag data for a specific duration of time; this request is received by *listener* task of the *Reader* agent. The *Reader* agent communicates with the physical reader and collects data for a specified duration of time and uses $its\ send\_collected\_data$ task to send collected event data back to the *Event Report Generator* agent. The $generate\_report$ task generates reports as specified in the $eventSpec$ and the *send-reports* task sends the generated reports to the *Report Delivery Handler* agent for formatting and dispatching it to the appropriate $ClientApp$ agent. These tasks are used whether the event reports are requested immediately or by subscription. However, for subscription requests there are additional tasks required; for example, $initializing\_subscription\_repeat\_period$ and $check\_subscription\_repeat\_period$ tasks, which are used to ensure that the process of event report generation repeats after every certain period of time. The $check\_status\_of\_eventSpec$ task is used to make sure that the process only repeats if the $ClientApp$ agent has not unsubscribed from receiving the reports.

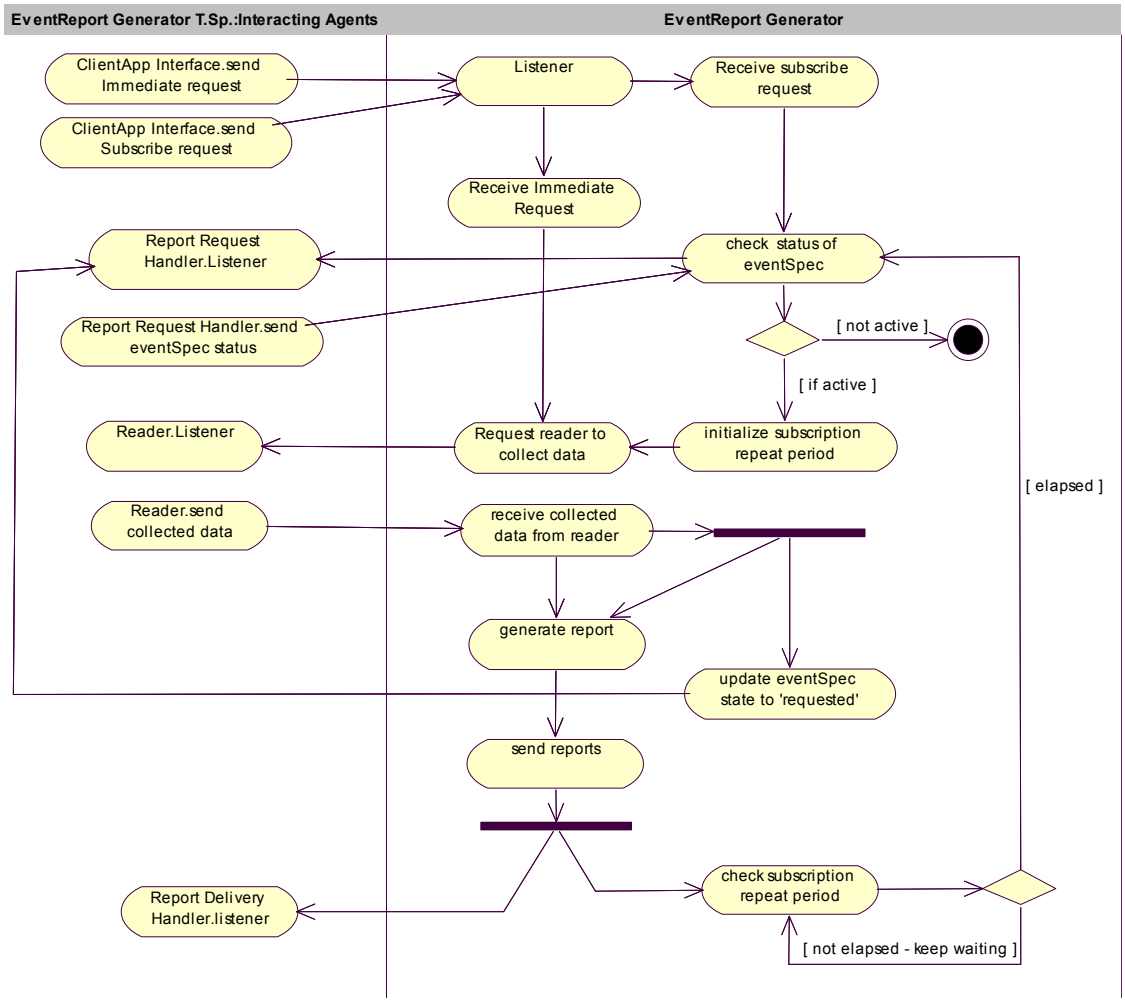


Figure 4-8: Tasks of the Event Report Generator agent

4.2.5 Ontology Description Phase

For the agent-based system to achieve its semantic richness and for the agents within the system to be able to interoperate, cooperate and coordinate with each other, they need to have a shared understanding of their working domain. That common representation of the domain-specific terminologies and theories represented as objects, concepts, entities and their relationships within the domain is referred to as an ontology. The PASSI methodology models an ontology from two points of view: the Domain Ontology Description (DOD) and the Communication Ontology Description (COD).

In DOD, the ontology of this middleware system was described by representing the pertinent domain-specific concepts and actions as classes and their relationships with each other as shown in Figure 4-9. Stereotype is used to differentiate between concepts and agent actions. The COD describes the exchange of information between the agents. The COD diagram has two types of classes: agent classes and communication classes as shown in Figure 4-10.

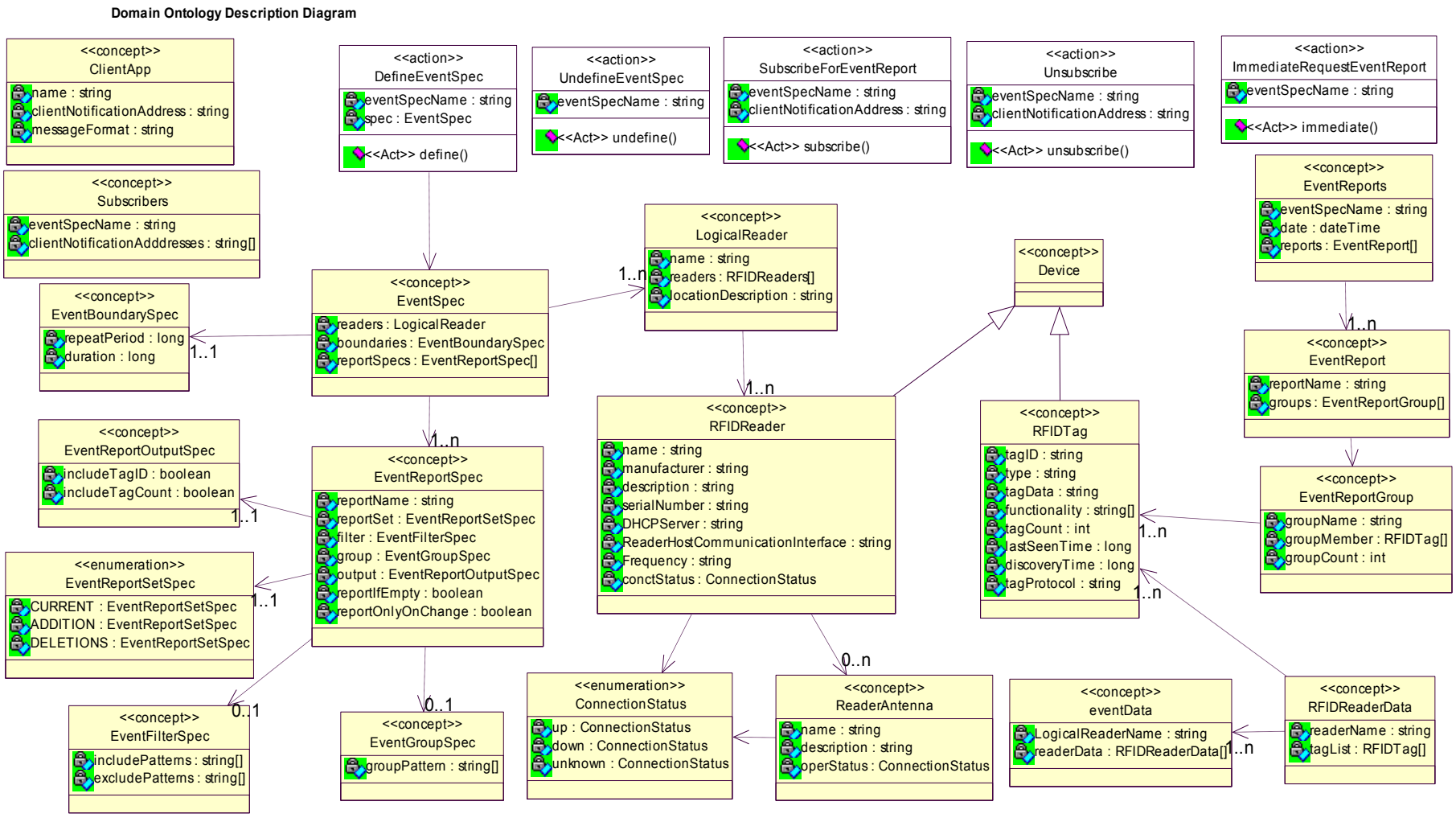


Figure 4-9: Domain Ontology Description Diagram

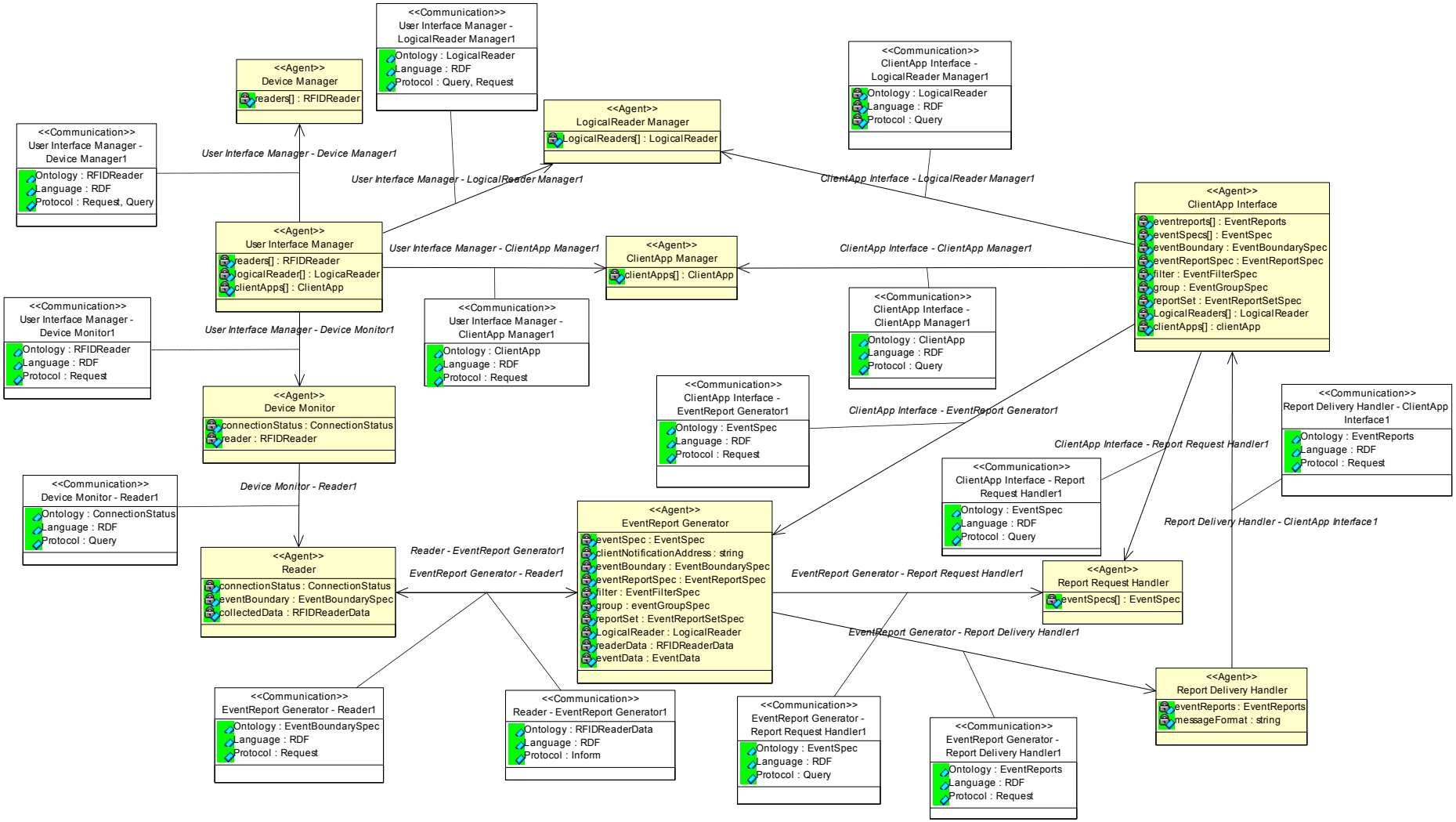


Figure 4-10: Communication Ontology Description Diagram

The attribute compartment of the agent class contains the agent's knowledge, which is obtained from the DOD diagram (Figure 4-9) with the same name as ontology's entity class name from the DOD. Communication classes represent the association relationship between the two communicating agents. The association is drawn from the initiator of the conversation (i.e. agent starting the conversation, the sender) to the responder (i.e. agent engaged in conversation after being contacted by some agent, the receiver). Conversations between agents are based on speech act theory [78] which explains that messages are actions, or communicative acts, as they are intended to perform some action by virtue of being sent. Conversations often fall into typical patterns. In such cases, certain message sequences are expected, and, at some point in the conversation, other messages are expected to follow. These typical patterns of messages are known as interaction protocols [79]. In addition, in order for the receiver agent to be able to interpret the meaning of the message content, the ontology and language used need to be specified. So, if agents are to communicate in the way that makes sense to them, they must share the same language, ontology and protocol. Hence the communication class in the COD has three attributes; ontology, language and interaction protocol. For example, in Figure 4-10, the $ClientApp\ Interface$ agent starts a conversation with $EventReport\ Generator$ agent through the communication class $ClientApp\ Interface - EventReport\ Generator1$. The communication class contains the $EventSpec$ ontology, *RDF* language and *Request* protocol. As explained before, the PASSI methodology includes the use of ontology and communications with a FIPA compliant structure. All the interaction protocols used in our model are FIPA standard protocols [79].

The DOD is then further developed using Protégé ontology editor and the FIPA/JADE compliant ontology Java code is generated using Protégé ontology *bean generator* plug-in. The *bean generator* automatically creates the ontology definition class and the agent actions and concepts classes. The ontology definition class describes the terminology of concepts and actions used by the agents in their space of communication, and the nomenclature of relationship between these concepts. The agent actions and concepts classes describe their semantics and structure. In terms of ontology classification, we can classify the DOD output as task ontology.

4.2.6 Role Description Phase

In the role description phase the lifecycle of each agent was modelled, taking into consideration the roles it can play, the collaboration that it needs, and the communications in which it participates.

In the Role Description diagram the roles are represented as classes grouped together in packages representing agents. Figure 4-11 shows the portion of role description diagram, illustrating the roles played by the reader agent. The information presented in the role classes are derived from the output of the Role Identification and Task Specification phases. In the Role Identification phase (section 4.2.3), information about the existing roles of an agent and changes in the roles were deduced, and in the Task Identification phase (section 4.2.4), the names of the tasks belonging to each role were obtained. The Role Description diagram, therefore, presents the information that is already in the design, but which is now assembled from a different point of view. The role is the focus and related data are: the agent class playing the role; the tasks involved in it (it is possible for the same task to be used in different roles of the same agent); the communication between the roles, some of them involving messages between different agents, and others messages within the same agent; and the dependencies between different roles.

Dependencies are direct consequences of MAS cooperation. However, since agents are autonomous entities, they could refuse to provide a requested service, and as a result these dependences do not always hold when MAS runs. It is therefore important to analyze these dependences and if possible to provide alternative ways of achieving the goal. Two types of dependencies were considered, namely service dependency and resource dependency. The *service dependency* means that the role of the sender agent depend on the role of the receiver agent to achieve a goal or perform an activity, and the *resource dependency* means that the role of the sender agent depends on the role of the receiver agent for the availability of an entity. Each communication between roles of different agents is marked with dependency type stereotype.

For example, in Figure 4-11 the $RequestListener$ role of the *Reader* agent receives a communication from $ReportRequestProcessor$ role of the $EventReport\ Generator$ agent. The *listener* task of the $RequestListener$ role passes the messages to an appropriate task, which is $ReceiveCollectDataRequest$ task. The

$ReceiveCollectDataRequest$ task triggers the agent role change to $DataCollector$ role in which the $collectdata$ task is executed to collect the data from the physical reader. The $ReportRequestProcessor$ role of the $EventReport\ Generator$ agent depends on $RequestListener$ role of the *Reader* agent to perform an activity and hence the *service* dependency stereotype on its communication ($EventReportGenerator - Reader1$).

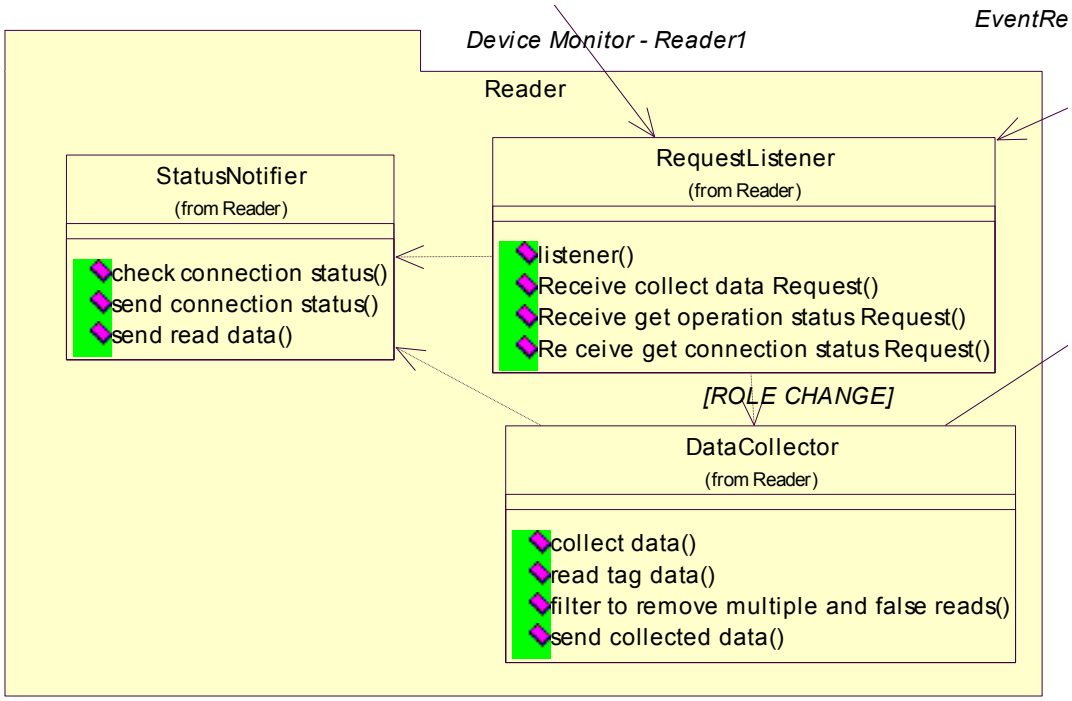


Figure 4-11: Role Description diagram for the Reader agent

4.2.7 Agent structure Definition Phase

In agent structure definition phase the focus is on the structure of the MAS, and it is composed of several class diagrams. The class diagrams are divided into two logical levels of abstraction: the multi-agent and the single-agent. In multi-agent the focus is on the general architecture of the system and therefore in it we find agents and their tasks. In the single-agent, each agent's internal structure is considered, showing all attributes and methods of the agent class and of its internal task classes.

4.2.7.1 Multi-Agent Structure Definition (MASD)

MASD diagram represents the designed multi-agent system as a whole, whereby one class is introduced for each agent identified during the agent identification phase. Figure 4-12 shows a portion of the modelled middleware MASD diagram. The attributes of the agent class represent the knowledge defined during the ontology description phase, and the methods of the agent class represent the tasks identified during the task identification phase. The relationship between two agents represented as association in the diagram represents the communication existing between the two agents and they are derived from the previous role description phase.

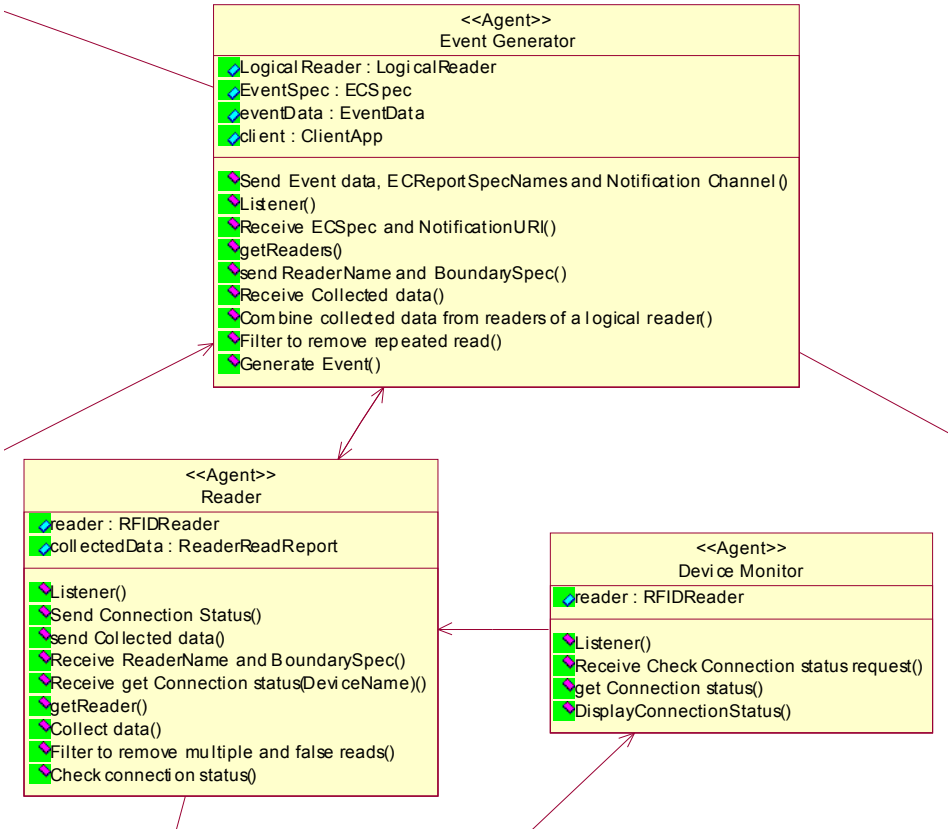


Figure 4-12: Portion of the Multi-Agent Structure Definition (MASD) diagram

4.2.7.2 Single-Agent Structure Definition (SASD)

In the SASD diagram the internal structure of each individual agent is addressed. We developed one diagram for each agent, in which we introduced one agent main class and the inner task classes. Figure 4-13 shows a single-agent structure definition diagram of the *Device Monitor* agent. The agent main class is inherited from the JADE agent class and it includes the attributes and the methods such as constructors, destructors and all methods required to register the agent in the white/yellow pages directories of the Jade environment. Each task of the agent is represented as a class inherited from the Jade behaviour class with methods needed specific to deal with series of activities and to handle communication events.

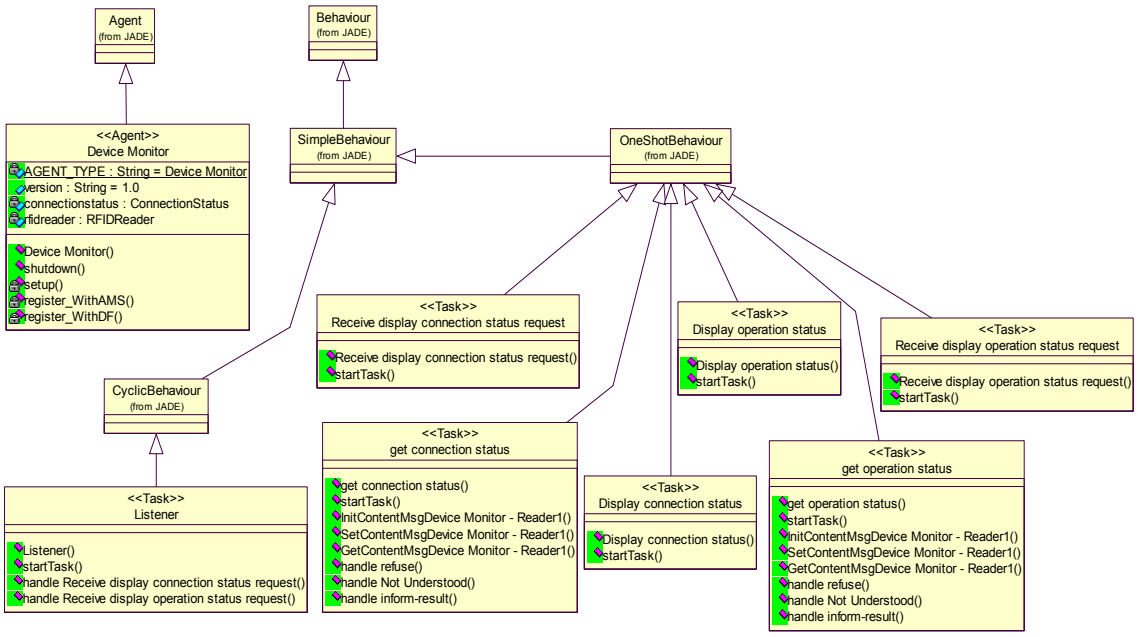


Figure 4-13: Single-Agent Structure Definition diagram for the Device Monitor agent

At this level of detail, the structure of the middleware software i.e. classes, methods and attributed had been described in sufficient detail to implement it. In the next chapter, the implementation of the ontology using Protégé editor together with implementation of the agent classes using the Jade platform and MySQL database will be discussed.

Chapter 5: Middleware System Implementation

In Chapter 4, PASSI methodology was employed to analyze and design the middleware system. In this chapter, the artefacts produced in Chapter 4 are reviewed and employed to implement the system using JADE platform and MySQL database.

Figure 4-5 in section 4.2.2 shows the designed multi-agent system whereby each class models a single agent in the system. When this diagram is reviewed, it can be seen that there are several agent classes (*Logical Reader Manager*, *Client App Manage*, *Device Manager*) with their main tasks being to interact with the database. These classes are merged together to form a database agent class. The *Report Delivery handler* and the *Event Report Generator* agent classes, which deal with event reports, are merged to form an event processing agent. Therefore, the implemented prototype system is comprised of the following agents: User Interface Agent; Client App Agent; Reader Agent; Database Agent; Event Processing Agent; Report Request Handler Agent; and Device monitor Agent.

An ontology is an essential component of any multi-agent system; it is used as the content of the agent messages; in this way all the agents using the same ontology have a common understanding of the domain concepts. Section 5.1 presents in detail the implementation of the ontology used in this middleware prototype system. Persistence storage is discussed in section 5.2 while section 5.3 describes the implementation of the designed middleware prototype.

5.1 Ontology Development

In the Ontology Description Phase in section 4.2.5, the concepts within the domain under discussion and their relationship were modelled and two artefacts produced. These artefacts are the *domain ontology description* diagram (Figure 4-9) and the *communication ontology description* diagram (Figure 4-10), which are both UML class models. These two artefacts are used as the basis for implementing the ontology discussed. They are considered to be the bases, because refinement was done on these initial designs whereby some concepts, predicates and actions were added, while others were removed or their attributes changed as the development process evolved.

One of the key functionality of the RFID middleware is to act as a device broker connecting applications to RFID devices through standardized interface. This allows applications to read data from many different types of readers even though they are made by different manufacturers and may have different interfaces themselves to access its functionalities. Another function is to manage the network of connected RFID devices in the deployment. To achieve these functionalities the RDDM middleware ontology contains the domain device ontology, whereby all the meta-information and knowledge about RFID devices and the types of devices are stored. In the RFID data modelling in Chapter 3, three types of devices were identified that are directly related to RFID in the RFID deployments, namely RFID tags, RFID readers and sensors. In the prototype implementation discussed here, sensors are not included and, hence, the ontology for two types of devices, RFID tags and RFID readers, are considered.

The best point to start at when thinking of using an ontology in an application is to find out if the ontology in mind already exists so that one can use it, modify it or extend it to suit one's application. Finding ontologies is important in order to avoid the creation of new ontologies where serviceable ones already exist [80]. This reuse of the domain knowledge capability is one of the major benefits of the ontology. The development of ontologies for the Web has led to the tremendous growth of services providing lists or directories of ontologies with search facilities. Such directories have been called ontology libraries [81]. The following are some of the main static libraries of ontologies: DAML Ontology Library [82] Schema Web [83] and Protégé Ontology Library [84]. Other ontology libraries have search engines that include crawlers searching the web for well-formed ontologies. The most popular ones are: Swoogle [85], the OntoSelect Ontology Library [86] and the Ontaria [87] Also, in mobile device domain the two most established databases for mobile device capabilities and features are UAProf – User Agent Profile [88]and WURLF – Wireless Universal Resource File [89].

After going through the above ontology libraries and search engines, the researcher could not find the ontology that is specific to RFID readers or RFID tags. The available device ontologies did not fully capture the properties and capabilities of these RFID devices. For example, the device ontology described in [90] is too general, while the one proposed by FIPA in [91] is more suitable for mobile devices. A new RFID Device ontology, which defines in detail the properties and capabilities of these devices, was therefore developed

in this research study. This is not application-specific ontology; therefore, it can be imported and used in any other application that is interested in the device-specific features and hardware characteristics of the RFID readers and tags. The device ontology was thereafter also extended to include the other RDDM middleware domains' concepts and actions, which are mainly application-specific to form RDDM middleware ontology.

5.1.1 Device Ontology

Like FIPA device ontology we chose to use frame-based ontology language together with Protégé platform to model our ontology. The Protégé platform can export the ontology to a variety of formats (such as OWL, RDF Schema, HTML or N-TRIPLE) if the ontology is required in other formats. The ontology is graphically visualized using Protégé's OntoViz Tab [92].

In frame-based knowledge representations the knowledge describing a particular concept is organized as a frame. A frame is a single place in which properties and axioms of a concept are specified. Frames usually contain properties called *slots*. In object oriented terms, frames can be related to classes and slots correspond to attributes. Relationships between frames are expressed by stating dependencies or restrictions between frames. Frames can have any number of slots assigned to it, and slots have some restrictions placed on them such as the data type, multiplicity, optionality, allowed values in case of enumerated data type and default values. The data type of a slot can be any of the primitive data types (Boolean, Integer, Float, or String), enumerated data type (predefined set of allowed values) or another frame (object data type). When a slot has an object data type it expresses the binary relationship between the two frames.

All slot restriction features are well-defined in the ontology, although they are not shown in some of the presented figures because of the limited representation of the OntoViz. The OntoViz produced figures capture the slot's data type, multiplicity and allowed values, but it does not show the optionality or default values of the slot.

For the OntoViz frame figures which are presented in the table format (e.g. see Figure 5-4); the top row defines the name of the frame ontology and the following rows define the frame's slots. The first column of the slots defines the slot's name and other columns define the slot data type, multiplicity and allowed values. A slot with two

columns defines a slot with primitive data types. A slot with three columns represent a slot with either object data type shown with "*Instance*" followed by their corresponding instance frame name or enumerated data type shown with "*Symbol*" followed by the allowed values. Multiplicity is represented by a symbol * whereas *Instance* defines a single instance while *Instance** defines a multiple instances.

Figure 5-1 shows the hierarchical view of the frames used in RFID device ontology. The RFID Reader, RFID Antenna, and RFID Tag frames model the three main physical devices while other frames represent concepts describing properties of those devices. Figure 5-2 shows the ontology frames and their relationships.

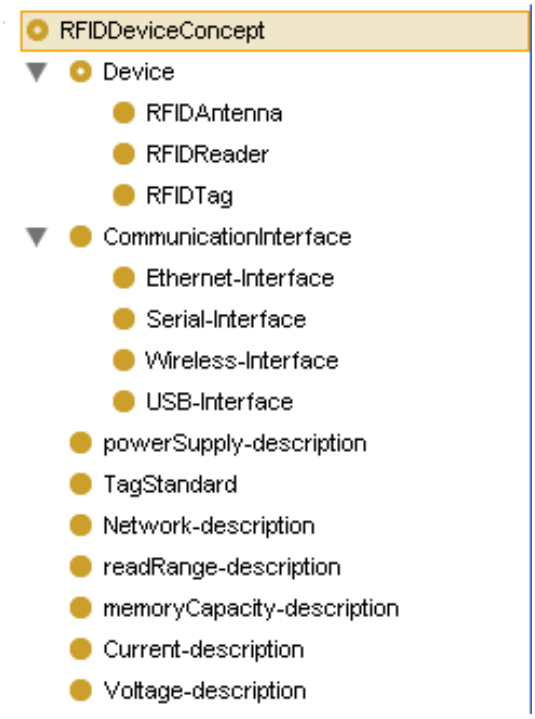


Figure 5-1: Frames in the RFID Device Ontology

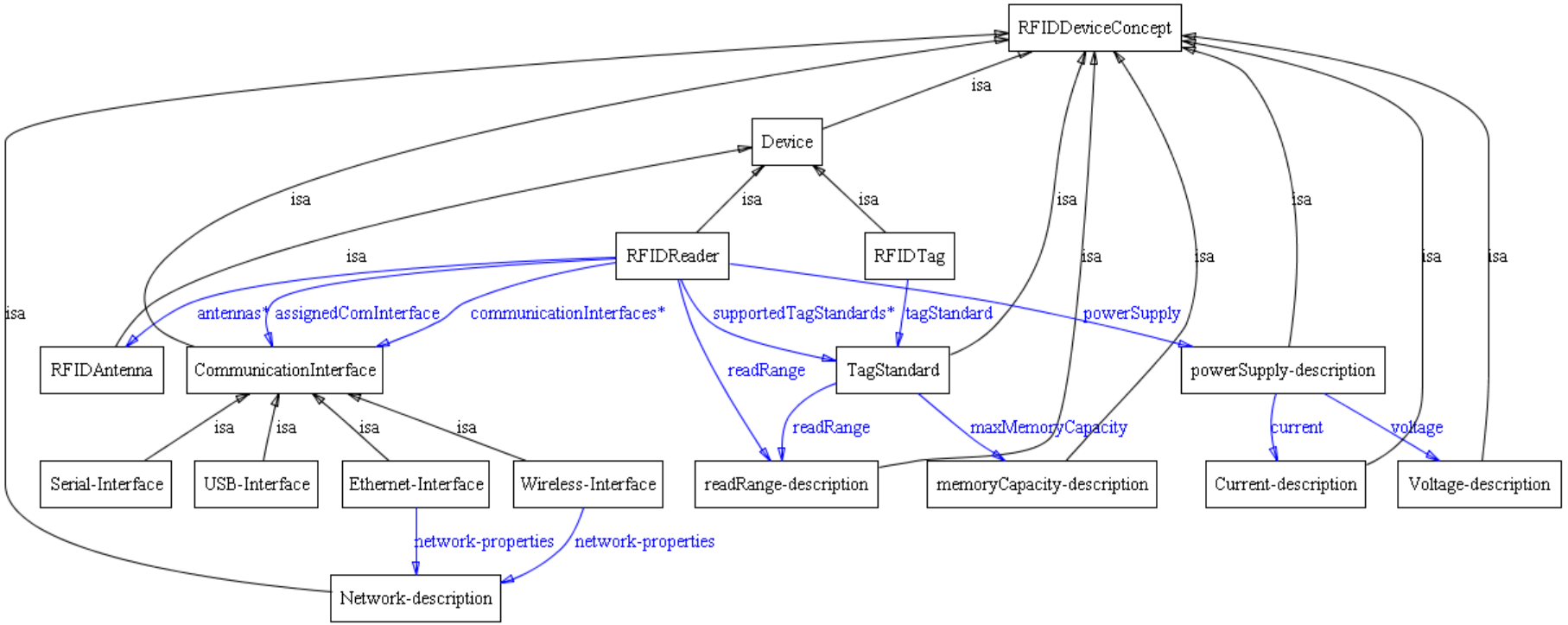


Figure 5-2: Frames in the RFID Device Ontology and their relationships

5.1.1.1 RFID Device Concept frame

This is the high-level frame of the RFID device ontology; it is an abstract frame with no slots assigned to it but encompasses all RFID device concepts. See Figure 5-2, where this frame is related to all other device concepts through "is a" hierarchical relationship.

5.1.1.2 RFID Reader frame

This frame represents the description that can be used to define most of the properties of the RFID reader. Figure 5-3 shows the description of RFID reader slots.

- The *Name* column defines the name of the slot;
- The *Cardinality* column defines the multiplicity of the slot as either "single" or "multiple", it also defines the optionality of the slot whereby the mandatory slots are preceded with "required" keyword; otherwise the slot is considered optional;
- The *Type* column defines the slot's data type; and
- *Other Facets* column defines other facets such as allowed values of the slot for enumerated data type and the slot's default value.

Template Slots

Name	Cardinality	Type	Other Facets
readerID	required single	Integer	
name	required single	String	
serialNumber	single	String	
modelNumber	single	String	
readerType	single	Symbol	allowed-values={STATIONARY,HANDHELD} value=STATIONARY
operatingFrequency	single	String	
readRange	single	Instance of readRange-description	
antennas	multiple	Instance of RFIDAntenna	
communicationInterfaces	multiple	Instance of CommunicationInterface	
assignedComInterface	single	Instance of CommunicationInterface	
supportedTagStandards	multiple	Instance of TagStandard	
logicalReader	single	Instance	
connectionStatus	single	Symbol	allowed-values={CONNECTED,DISCONNECTED} value=DISCONNECTED

Figure 5-3: Description of the RFID Reader slots

Figure 5-4 shows this frame and its slot's definition as presented by OntoViz graph. Figure 5-5 shows this frame's relationship with other frames. Relationships are shown by using a dotted line. For example, the antennas* relationship defines a relationship between RFID Reader frame and the RFID Antenna frame, stating that the reader can have multiple antennas associated with it.

RFIDReader		
readerID		Integer
name		String
serialNumber		String
modelNumber		String
readerType	Symbol	STATIONARY
		HANDHELD
operatingFrequency		String
readRange	Instance	readRange_description
antennas	Instance*	RFIDAntenna
communicationInterfaces	Instance*	CommunicationInterface
assignedComInterface	Instance	CommunicationInterface
supportedTagStandards	Instance*	TagStandard
logicalReader	Instance	LogicalReader
connectionStatus	Symbol	CONNECTED
		DISCONNECTED
readRate		String
powerSupply	Instance	powerSupply_description
manufacturer		String
IOInterface_description		String
description		String
lastUpdate		String

Figure 5-4: Description of the RFID Reader frame using OntoViz

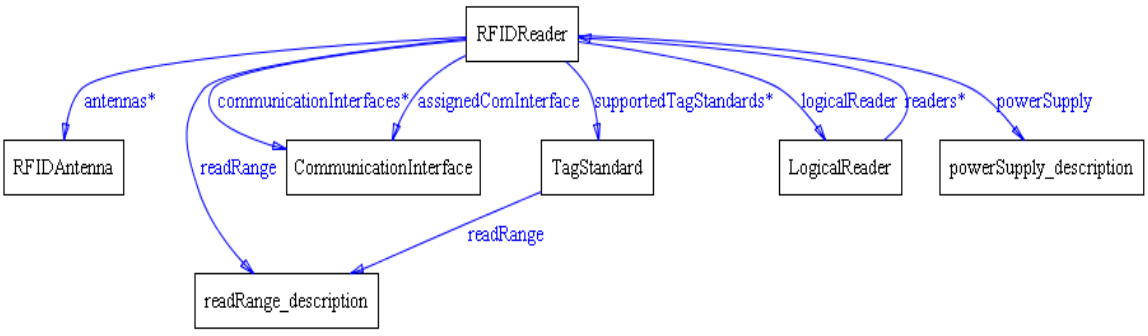
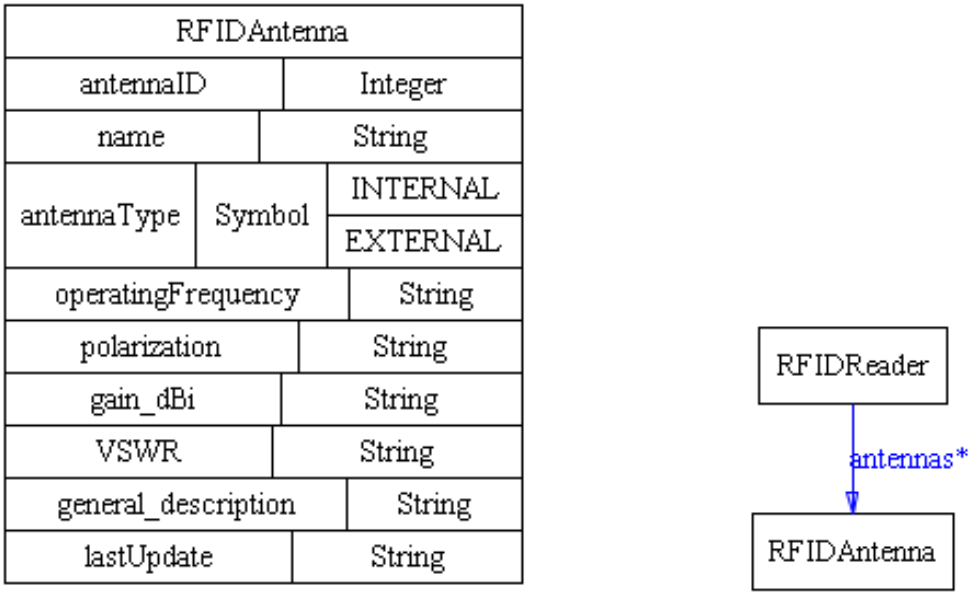


Figure 5-5: Relationship between RFID Reader frame and other frames

5.1.1.3 RFID Antenna frame

This frame represents the description that can be used to define general properties of the RFID antenna. Figure 5-6(a) shows this frame and its slot's definition and Figure 5-6(b) shows this frame's relationship with other frames.



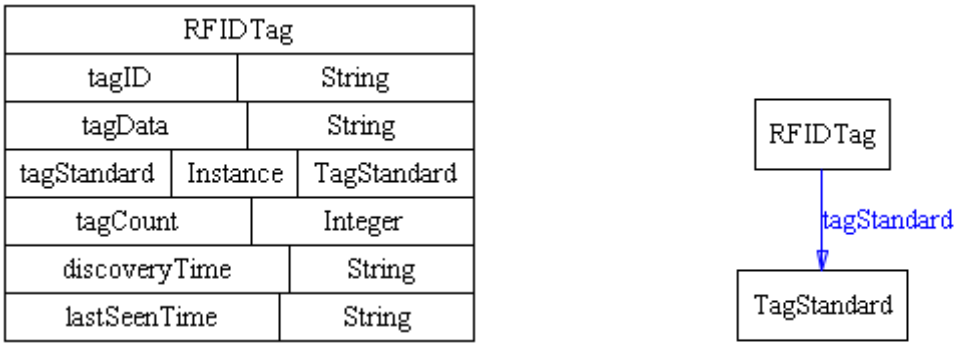
(a) (b)

Figure 5-6: Description of the RFID Antenna frame and its relationship with other frames

5.1.1.4 RFID Tag frame

This frame represents the description that can be used to define general properties of the RFID tag. Figure 5-7(a) shows this frame and its slots definition and Figure 5-7(b) shows this frame's relationship with other frames.

- *Tag ID* slot defines the RFID tag unique identification code
- *Tag Data* slot defines the extra data encoded into the tag apart from its ID code
- *Tag Standard* slot defines the RFID standard that this tag adheres to
- *Tag Count* slot; this is an application-specific slot which describes the number of times in which the tag was read by the reader within a specific period.
- *Discovery Time* slot; this is application-specific slot and it describes the time the tag was first seen by the reader
- *Last Seen Time* slot; this is application-specific slot and it describes the time the tag was last seen by the reader.



(a) (b)

Figure 5-7: Description of the RFID Tag frame and its relationship with other frames

5.1.1.5 Communication Interface frame

An RFID reader uses a communication interface to interact with the host computer. This frame represents the description that can be used to define general information of the communication interface concept. This frame is a super class of four subclasses which are different types of communication interfaces. Subclasses inherit the slots of the super-class. Figure 5-8 shows this frame and its slot's definition together with its subclasses. Figure 5-9 shows this frame's relationship with other frames. The reader can have several types of communication interfaces; hence, the relationship $communicationInterfaces$, but at one particular time it uses one type of interface to connect with the host computer or network and hence the relationship $assignedComInterface*$.

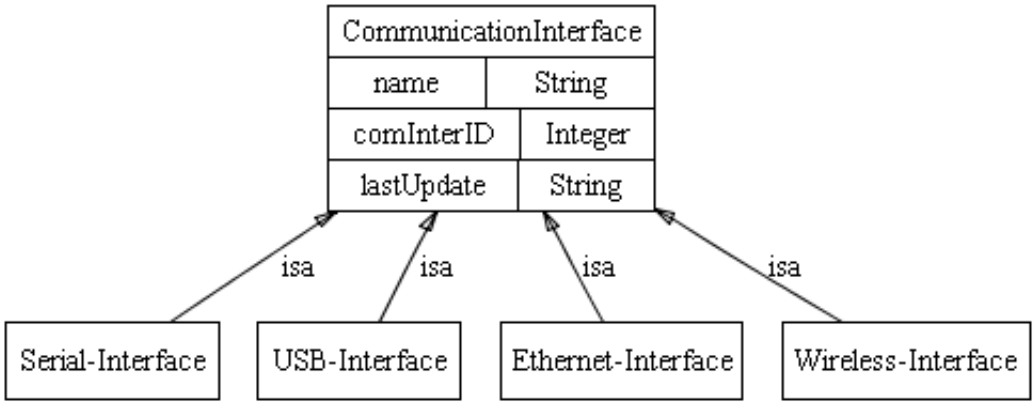


Figure 5-8: Description of the Communication Interface frame and its subclasses

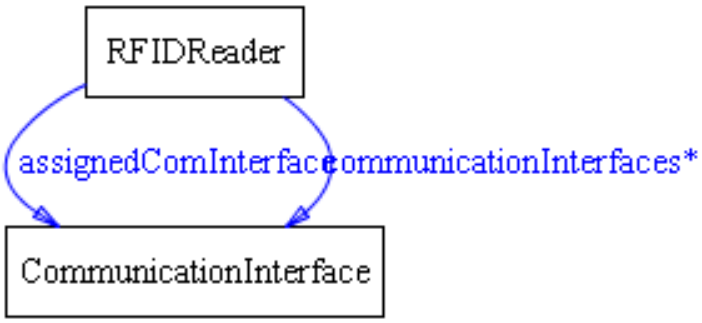


Figure 5-9: Relationship between communication interface frame and other frames

5.1.1.6 Serial Interface frame

This frame represents the description that can be used to define properties of the serial interface connection. Figure 5-10 shows this frame and its slots definition. In addition to its slots this frame also inherits all the slots of its superclass communication interface.

Template Slots

Name	Cardinality	Type	Other Facets
name	required single	String	
comInterID	single	Integer	
lastUpdate	single	String	
baudRate	single	String	
dataBits	single	Integer	
stopBit	single	Integer	
parity	single	Symbol	allowed-values={EVEN,ODD,NONE,MARK,SPACE}
flowControl	single	Symbol	allowed-values={NONE,RTSCTS_IN,RTSCTS_OUT,XONXOFF_IN,XONXOFF_OUT}
general-description	single	String	

Figure 5-10: Description of the Serial Interface frame

5.1.1.7 USB Interface frame

This frame represents the description that can be used to define properties of the USB interface. Figure 5-11 shows this frame and its slots definition. In addition to its slots, this frame also inherits all the slots of its superclass communication interface.

Template Slots

Name	Cardinality	Type
name	required single	String
comInterID	single	Integer
lastUpdate	single	String
general-description	single	String

Figure 5-11: Description of the USB Interface frame

5.1.1.8 Ethernet/Wireless Interface frame

Ethernet interface and Wireless interface frames have similar slot definitions that define the properties of the network connection. Figure 5-12 shows the Ethernet interface frame and its slot definition. In addition to its slots, this frame also inherits all the slots of its superclass communication interface.

Template Slots

Name	Cardinality	Type	Other Facets
name	required single	String	
comInterID	single	Integer	
lastUpdate	single	String	
network-properties	single	Instance of Network-description	

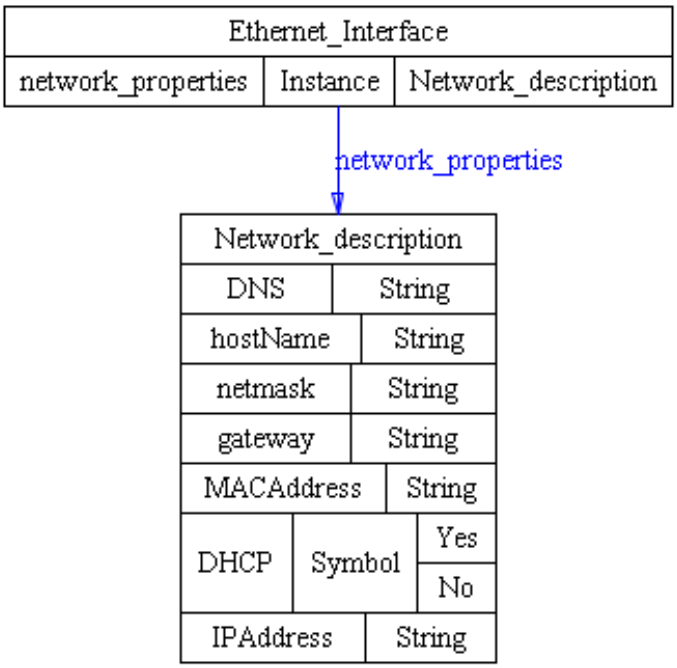
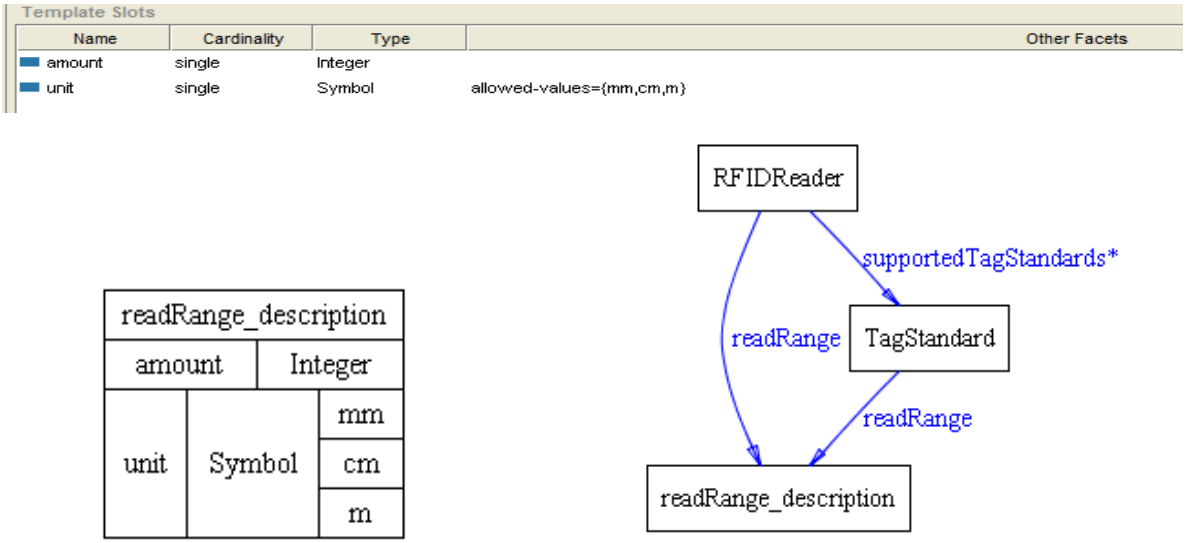


Figure 5-12: Description of the Ethernet Interface frame

5.1.1.9 Read range description frame

This frame represents the description that can be used to define the amount and unit of the maximum read range between a reader and a tag. Figure 5-13(a) shows this frame and its slot's definition and Figure 5-13(b) shows this frame's relationship with other frames.

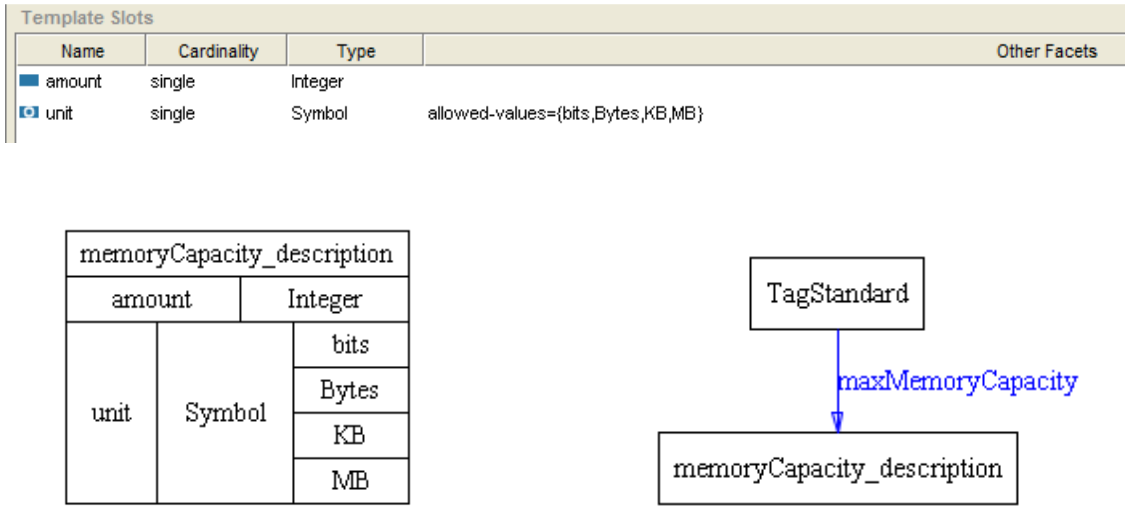


(a) (b)

Figure 5-13: Description of the read range description frame and its relationship with other frames

5.1.1.10 Memory capacity description frame

This frame represents the description that can be used to define the amount and unit of the tag's memory capacity. Figure 5-14(a) shows this frame and its slots definition and Figure 5-14(b) shows this frame's relationship with other frames.



(a) (b)

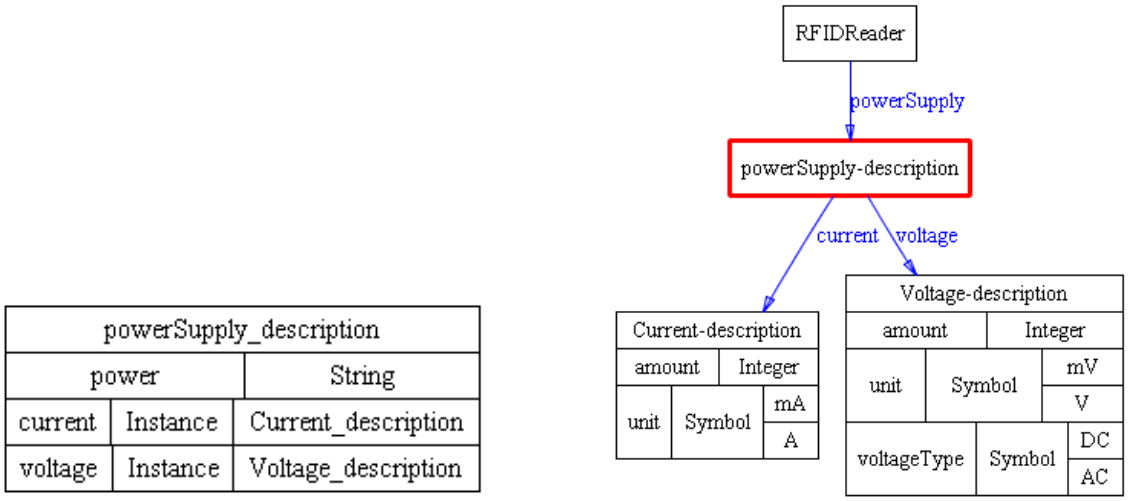
Figure 5-14: Description of the memory capacity description frame and its relationship with other frames

5.1.1.11 Power supply description frame

This frame represents the description that can be used to define the device's power supply requirement related details. Figure 5-15(a) shows this frame and its slots definition and Figure 5-15(b) shows this frame's relationship with other frames.

Template Slots

Name	Cardinality	Type	Other Facets
power	single	String	
current	single	Instance of Current-description	
voltage	single	Instance of Voltage-description	

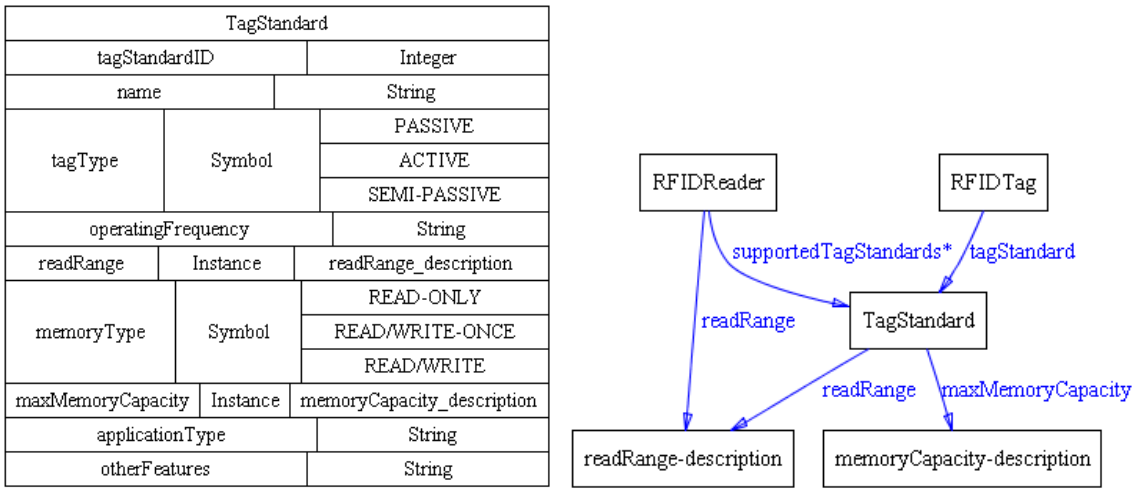


(a) (b)

Figure 5-15: Description of the power supply description frame and its relationship with other frames

5.1.1.12 Tag Standard frame

This frame represents the description that describes the main features defined in the RFID standards. RFID tags and readers adhere to specific RFID standards that describe how the reader and tag communicate with each other as well as specific uses of RFID technology. Figure 5-16(a) shows this frame and its slots definition and Figure 5-16(b) shows this frame's relationship with other frames.



(a) (b)

Figure 5-16: Description of the Tag Standard frame and its relationship with other frames

5.1.2 RDDM Middleware Ontology

JADE provides a basic ontology for agent communication. This basic ontology contains schemas for the primitive data types and the SL0 operators i.e. basic ontological elements required for minimal agent interaction. However, for RDDM middleware agents to work within the JADE platform we need to define our own application-specific vocabulary and semantic for the content of the messages that will be exchanged between its agents. JADE framework requires that application-specific ontologies should be implemented by extending its basic ontology.

The JADE basic ontology consists of three generic interface classes: *predicate* class, *concept* class, and *agent action* class.

- *Predicates* are expressions that say something about the status of the world and can either be true or false. Predicates can be used effectively as the content of an INFORM or QUERY-IF message, both of which express facts.
- *Concepts* are expression identifying entities (abstract or concrete) that exist in the domain that agents may reason about. Concepts typically make no sense if used directly as the content of the message; they are generally referenced inside predicates and other concepts.

- *Agent actions* are special concepts that indicate actions that can be performed by some agent. Agent actions can be used effectively as the content of a REQUEST messages.

For the agent deployed within the JADE framework to be able to fully exploit the JADE content language and ontology support offered by the platform, the agent developer is advised to follow the following steps [93]:

1. Define the ontology including the schemas for the types of predicates, agent actions, and concepts that are pertaining to the addressed domain.
2. Develop proper Java classes for all types of predicates, agent actions and concepts in the ontology implementing its corresponding interface.
3. Select a suitable content language among those directly supported by JADE.
4. Register the defined ontology and the selected content language with the agent.
5. Create and handle message content as Java objects that are instances of the classes developed in step 2 and let JADE translate these Java objects to/from strings or sequence of bytes that fit the content slot of ACL Messages.

Following these steps the RDDM middleware , which is an application-specific ontology that describes the elements that are used as the content of the agent messages in the RDDM middleware domain, were implemented. Like the device ontology, the RDDM ontology is also a frame-based ontology developed using Protégé ontology editor. The RDDM ontology is implemented by extending the JADE basic ontology and by implementing its three interface classes: *concept, agent action, and predicate*. The RDDM ontology's agent actions, concepts and predicates are derived from the *ontology descriptions* and *task specifications* identified earlier in chapter 4 while taking into consideration the revised agent classes. The RDDM middleware domain concepts are classified into two groups: concepts related to the devices in the domain and concepts related to data event processing. The concepts related to device are the concepts previously defined in the RFID device Ontology. Figure 5-17 shows the root class frames of the actual designed RDDM ontology with a limited view of the ontology classes. Figure 5-18 shows the hierarchical view of the frames constituting the event processing concepts.

After the RDDM ontology was defined using Protégé, the ***Ontology Bean Generator*** plug-in of the Protégé editor was then used to automatically generate the JADE compliant ontology source code. The ontology bean generator generates the ontology definition class together with Java class for all predicates, concepts and agent actions defined in the ontology. The code in Figure 5-19 shows an abstract view of the RDDM ontology definition class.

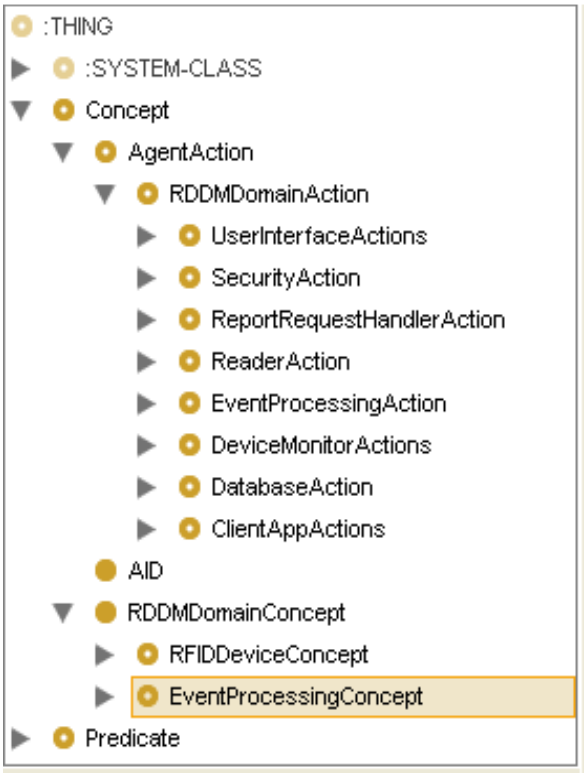


Figure 5-17: RDDM Ontology root class frames

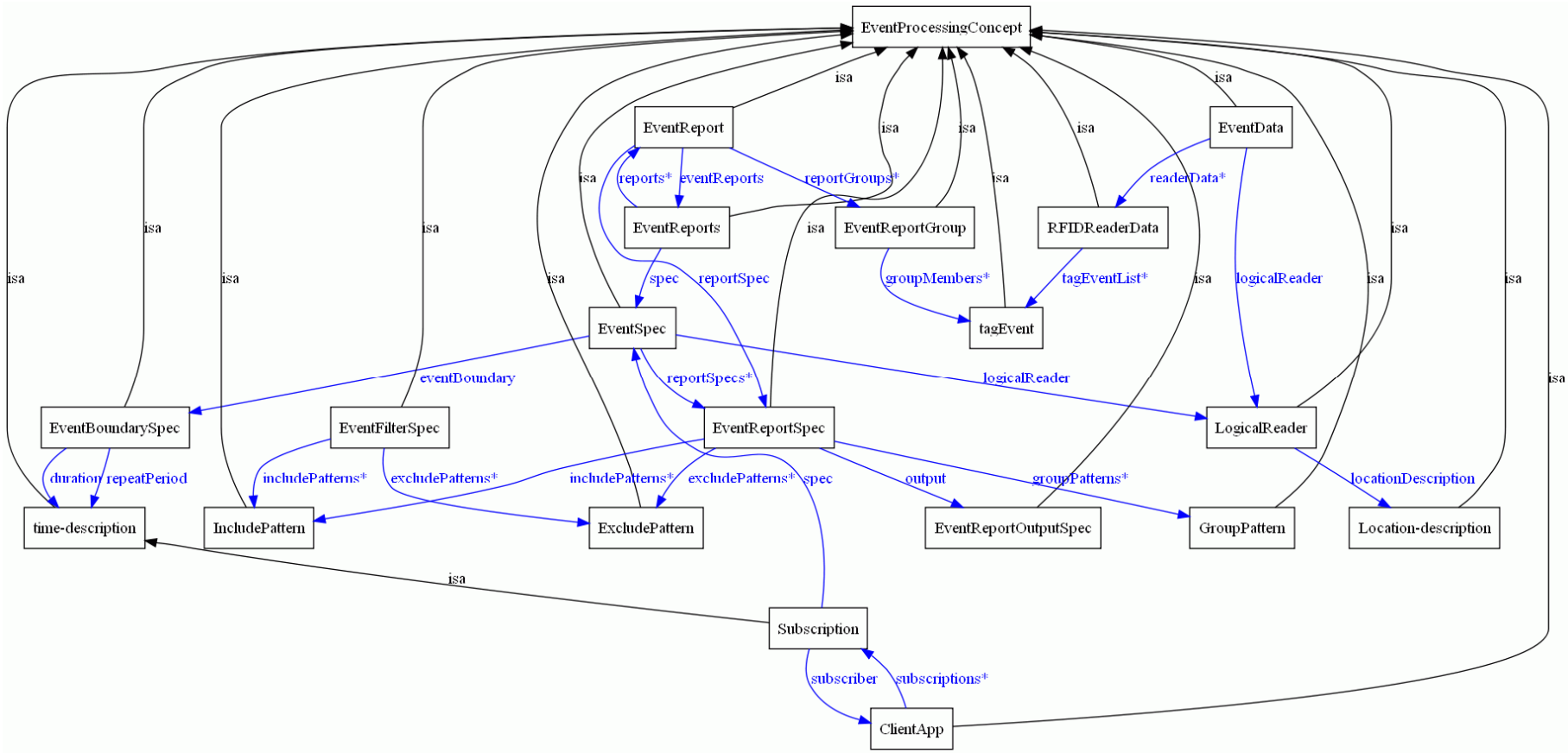


Figure 5-18: Frames of the Event Processing Concepts and their relationships

```
// file: RDDMOntology.Java generated by ontology bean generator.
package RFIDMiddlewareOntology;
//IMPORTS
import jade.content.onto.*;
import jade.content.schema.*;
...
public class RDDMOntology extends jade.content.onto.Ontology  {
 //NAME
 public static final String ONTOLOGY_NAME = "RDDM";
 // The singleton instance of this ontology
 private static ReflectiveIntrospector introspect = new ReflectiveIntrospector();
 private static Ontology theInstance = new RDDMOntology();
 public static Ontology getInstance() {
   return theInstance;
 }
 // VOCABULARY
        public static final String RFIDREADER="RFIDReader";
        public static final String RFIDREADER_NAME="name";
        public  static  final  String  RFIDREADER_OPERATINGFREQUENCY  ="  operatingFrequency
";
        public static final String RFIDREADER_ANTENNAS =" antennas ";
        public    static    final    String    RFIDREADER_    ASSIGNEDCOMINTERFACE    ="
assignedComInterface ";
        ...
 /**
  * Constructor
 */
 private RDDMOntology(){
  super(ONTOLOGY_NAME, BasicOntology.getInstance());
  try {
  // adding Concept(s)
  ConceptSchema rfidReaderSchema = new ConceptSchema(RFIDREADER);
  add(rfidReaderSchema, RFIDMiddlewareOntology.RFIDReader.class);
        ...
  // adding AgentAction(s)
  AgentActionSchema checkConnectionStatusSchema = new
                                    AgentActionSchema(CHECKCONNECTIONSTATUS);
 add(checkConnectionStatusSchema, RFIDMiddlewareOntology.CheckConnectionStatus.class);
        ...
  // adding AID(s)
        ...
// adding Predicate(s)
   PredicateSchema isLocatedSchema = new PredicateSchema(ISLOCATED);
   add(isLocatedSchema, RFIDMiddlewareOntology.IsLocated.class);
        ...

  // adding fields
rfidReaderSchema.add(RFIDREADER_NAME,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY );
```

```
rfidReaderSchema.add(RFIDREADER_OPERATINGFREQUENCY,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);

rfidReaderSchema.add(RFIDREADER_ANTENNAS, rfidAntennaSchema, 0, ObjectSchema.UNLIMITED);

rfidReaderSchema.add(RFIDREADER_ASSIGNEDCOMINTERFACE,communicationinterfaceSchema,
ObjectSchema.OPTIONAL);
        ...
  // adding name mappings
        ...
  // adding inheritance
  rfidReaderSchema.addSuperSchema(deviceSchema);
        ...
  }catch (Java.lang.Exception e) {e.printStackTrace();}
 }// end constructor
 }// end of RDDM Ontology class
```

Figure 5-19: Abstract view of RDDM Ontology class

Referring to the abstract ontology code presented in Figure 5-19, the ontology is characterized by the following features:

1. One name

   public static final String ONTOLOGY_NAME = "RDDM";

2. One or more ontology that it extends

   import jade.content.onto.;*

   public class RDDMOntology ***extends*** *jade.content.onto.Ontology{…}*

3. Ontology vocabulary – this defines all the constants used for names of concepts, predicates, agent actions and their slots. They represent the terms that constitute the specific language of the agents. For example:

   public static final String RFIDREADER = "RFIDReader";

   public static final String RFIDREADER_NAME = "name";

4. Set of element schemas - these are objects describing the structure of concepts, actions, and predicates defined in the ontology. For example;

   - This code creates the concept schema

     ConceptSchema rfidReaderSchema = new ConceptSchema(RFIDREADER);

   - This code adds the created schema in the ontology and associates it with its corresponding Java class.

$$add(rfidReaderSchema, RFIDMiddlewareOntology.RFIDReader.class);$$

- This code define the structure of the created concept schema

$$rfidReaderSchema.add(RFIDREADER\_NAME,$$
$$(TermSchema)getSchema(BasicOntology.STRING),$$
$$ObjectSchema.MANDATORY\ );$$

  The $add(\ldots)$ method of the concept schema takes three arguments: the name of the slot to be added, the schema of this slot, and optionality of the slot.

- Schemas that describe concepts support inheritance. This code defines the schema inheritance

$$rfidReaderSchema.addSuperSchema(deviceSchema);$$

Once the RDDM ontology class and its corresponding Java classes generated by the ontology bean generator were established, it could be registered with the content manager of the middleware agents, and the ontology Java classes used as the content of the agent's message within the JADE framework.

5.1.3 Ontology Content Language

According to the FIPA specification the value of the content slot of the agent's message can either be a string or a raw sequence of bytes. When representing complex information such as abstract concepts, objects or structured data, it is necessary to adopt a well-defined syntax so that the content of the message can be parsed by the receiver to extract each specific piece of information correctly. According to FIPA terminology this syntax is known as a content language. To enable a meaningful and consistent communication between software agents a common content language is required to be used by all the agents involved in the communication. JADE provides a support for three types of content languages: the SL language, the LEAP language and an XML–based content language. A codec for a content language is a Java code able to manage content expressions written in that language. SL codec and LEAP codec are directly included in the JADE while XML codec can be found as a JADE add-on.

The **SL content language** is a human readable string encoded content language based on the S-Expression syntax [94]. It is one of the mostly used content languages in the

scientific community dealing with intelligent agents. It can be used in the open agent-based applications where agents produced by different developers and running on different platforms must communicate.

The **LEAP content language** is a non-human-readable byte-encoded content language defined specifically for JADE agents for use in the devices with limited memory capacity. The LEAP codec is lighter than SL codec, but only JADE agents can use this language.

The **XML content language** uses XML syntax. It is a cross-platform, language-independent representation which is also human-readable. This codec is particularly useful when a set of ontological entities has to be exported or imported to/from an external system. However, the developer is required to have additional skills with respect to pure Java programming.

In the RDDM middleware, XML content language was chosen, because it is platform and language independent; it is human-readable - which is very useful when debugging and testing an application; and also, it provides a mechanism for middleware agents to communicate with other external applications.

5.2 Persistence Storage

Persistence storage is an essential part of any information system. Persistence storage is required in the middleware for storing the ontological information, including RFID static and dynamic data in a proper data model in order to provide an efficient querying and analysis of both recent and historical RFID events data. Persistence storage is any technology that can be used to permanently store objects for later update, retrieval, and/or deletion. Although there are different types of persistence mechanisms available such as file, hierarchical database, object oriented database, object-relational database, Network database, relational database (RDB) and XML database, RDB and in particular MySQL database was chosen to persistently store this system prototype objects and data. Among the available data storage technologies, RDBs are the most generally used type of data persistence mechanism. Since the developed middleware is focused on integrating the RFID data with other legacy enterprise applications, an RDB - which is more likely to be used by many enterprises - was chosen. MySQL is used because it is open-source, free

software and it is good enough for our prototype design, even though any type of RDB can be used.

The RDDM middleware system is implemented using the Java-based JADE platform, which is an object oriented technology. So, while the software agent application is based on object-oriented technology, the data storage mechanism is based on relational technology. Relational databases only store data and not objects; therefore, in order to store one's objects in a RDB, one need to *flatten* one's objects in order to create a data representation of one's object. To retrieve an object, one would need to read the data from the database and then create the object, a process called *restoring* the object based on the data. This is not as straightforward as it sounds and can be a challenging task. Another vital difference is that in a RDB, relationships between entities are implemented via foreign keys, while in the object-oriented paradigm, relationships between objects are implemented through class-based *references* and *inheritance* hierarchies.

5.2.1 RDDM Middleware System Persistence Class Model

The ontology classes were reviewed and the concepts and the data that need to be stored were identified. Not all classes or class attributes are persistent because some are processing classes or attributes that are required for temporary usage only. For example, agent action and predicate classes are not persistent. Figure 5-20 shows a portion of our persistence class model diagram. This diagram also shows the relationships between the persistence classes.

The persistent classes in the RDDM ontology were then revised to include the extra scaffolding attributes which are required for relationship implementation and for object to data model mapping. Figure 5-21 shows the portion of the RDDM class Model which includes the persistence attributes assigned with <<Persistence>> stereotype and extra scaffolding attributes to implement relationship between classes. Persistence attributes are extra attributes which have no business meaning, but they are required for the proper functioning of the database, and hence they are included in the object model to facilitate the object-database mapping process. This extra information typically includes primary key information, particularly when the primary key is a surrogate key that has no business meaning, and concurrent control marking such as timestamp or incremental counters. An example of extra scaffolding attribute is the inclusion of the following attributes in the

Event Spec class (see Figure 5-21): *eventSpecID* attribute which is an object surrogate ID; *last Update* attribute which is the time stamp concurrent control mark showing the last time the object was updated in the database; and *subscriptions* collection attribute to represent a bidirectional relationship with *Subscription* class, which was not initially captured in the ontology class.

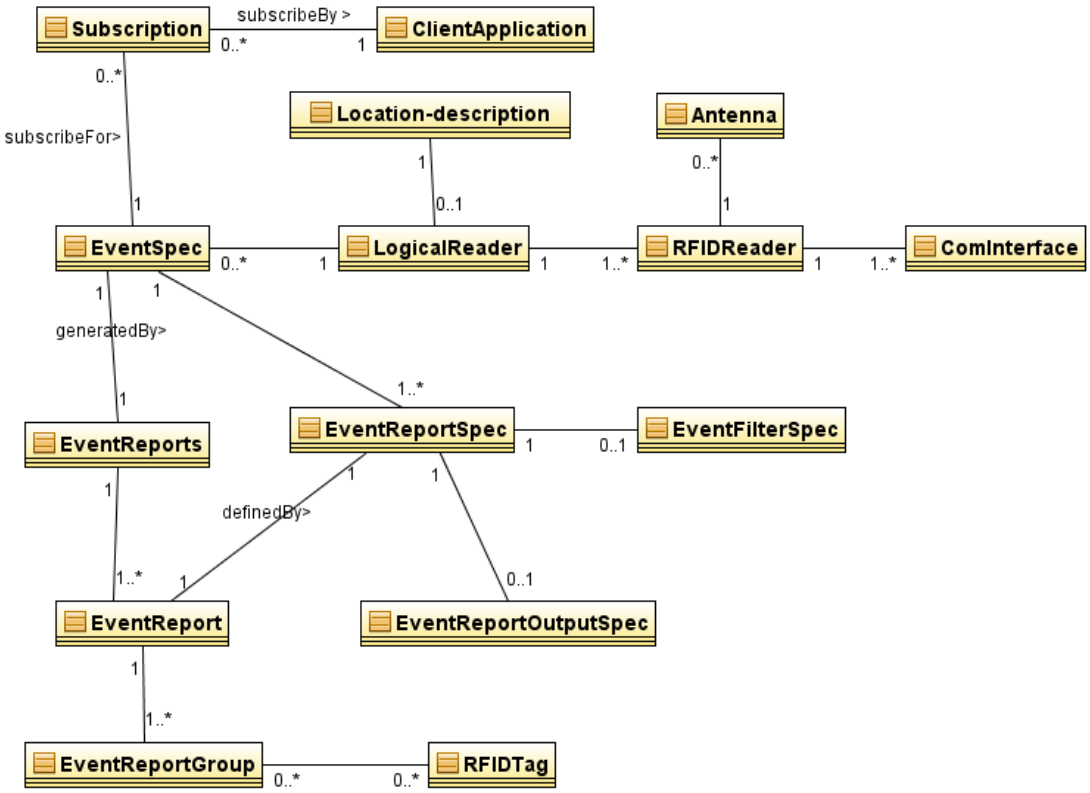


Figure 5-20: Portion of the RDDM Middleware System persistence class model diagram showing relationships between classes

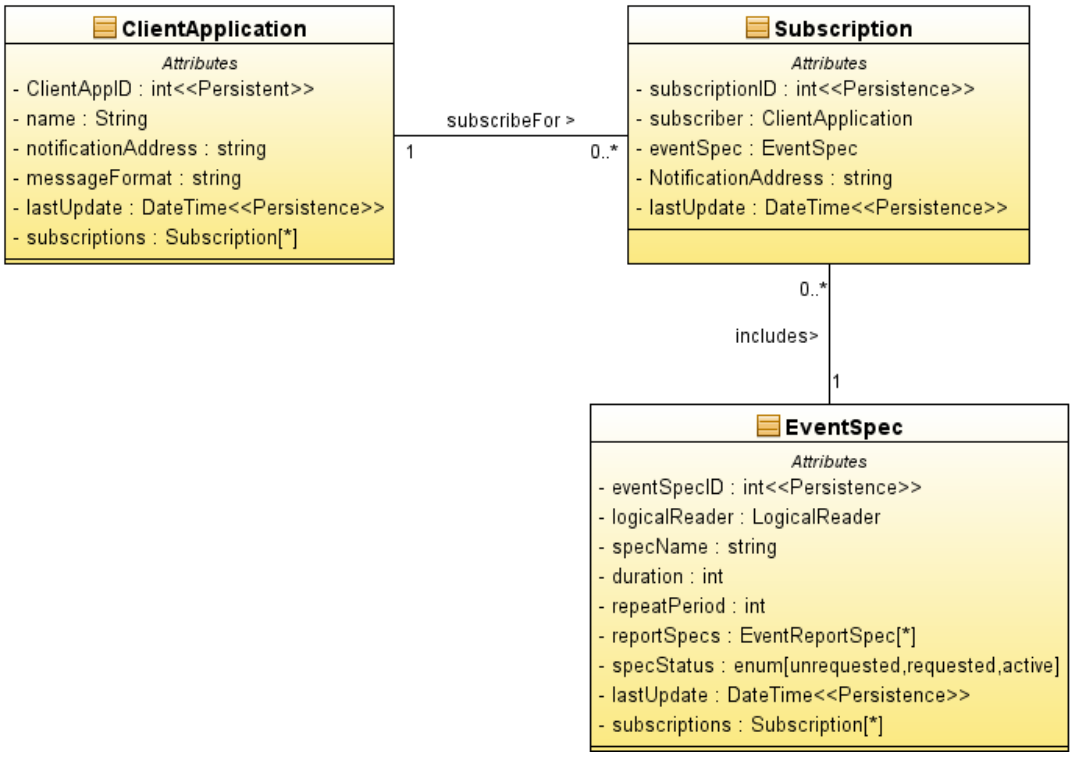


Figure 5-21: A portion of the persistence class model diagram which includes the class persistence attributes and scaffolding attributes required for implementing relationships

5.2.2 Relational Database Model

To develop the database schema based on object schema, the metadata between the two schemas were mapped. Classes are vertically mapped to database tables and class attributes are mapped into table columns while ensuring that the table columns have the same data types as their corresponding class attributes. Class names and table names can be different; column names and attribute names can also be different, but it is important that they have the same data types. The two schemas are then adjusted to ensure that they conform to temporal RFID data model which provides efficient querying of RFID events discussed in chapter 3.

Relationships in relational databases are maintained through the use of foreign keys. In a one-to-one relationship, the unique foreign key is implemented in either one of the tables. For example, $EventSpec$ and $EventReports$ classes have a one-to-one unidirectional relationship with each other. Since one is interested in knowing from which specifications $EventReports$ was generated, the $EventSpec$ primary key $eventSpecID$ was included as the foreign key in the $EventReports$ table and its index type set as **unique** to conform to a one-to-one relationship.

To implement a one-to-many relationship, a foreign key is implemented from the "one table" to the "many table", i.e. by putting the foreign key into the "many table". For example, inclusion of *ClientAppID* which is the primary key of the *ClientApplication* table as a foreign key in the *Subscription* table (see Figure 5-22).

To implement many-to-many relationship, the relationship is first converted into two one-to-many relationships, both of which involves an associative table. Then the foreign key is implemented in the "many table" in each of the two relationships.

The final relational database model schema was then implemented in the MySQL database using MySQL GUI editor.

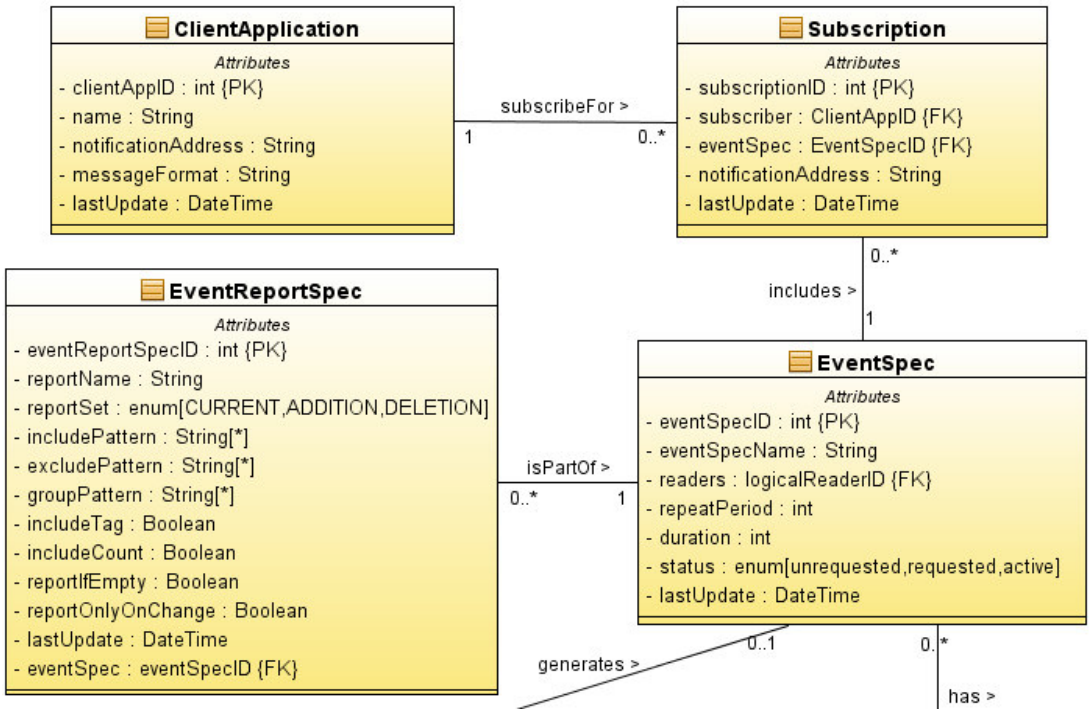


Figure 5-22: Portion of the RDDM Middleware System database model diagram using UML notation

5.2.3 Data Binding Between Persistent Class Model and Database Model

There are several strategies for implementing the object/relational mapping (also known as data binding) between object schema and the relational database schema. These strategies include embedding the Structured Query Language (SQL) statements into one's

objects, using data access objects and persistence frameworks. More information about these strategies and their comparison can be found in [95].

In the prototype implementation discussed here, persistence framework strategy using a Java Persistent API also referred to as JPA [96] was adopted. JPA is a Java programming language framework for managing *object relational mapping* in applications using Java platforms. A persistence framework, often refereed as a *persistence layer*, fully encapsulates database access from the application objects. It reduces coupling between the object schema and the data schema in such a manner that simple changes in data schema do not affect the application code. It provides the application objects with persistent services; the ability to read data from, write data to, and delete data from data sources without the application objects having to know anything about the data sources. The application objects only interact with the framework. Instead of writing code to implement the logic required to access the database, one rather defines the metadata that represent the mappings of one's application objects together with associations between them. Based on this metadata, the persistence framework generates the database access code required to make the application objects persistent. The whole process of flattening an object to create data representation and restoring an object based on data is taken care of by the persistent layer.

JPA uses a Java Persistence Query Language (JPQL) to make both static and dynamic queries against business objects stored in relational database. JPQL queries resemble SQL queries in syntax, but operate against business objects rather than directly with database tables. Instances of a business objects correspond to individual rows in the database table. Persistence business classes and their relationships are expressed through object/relational metadata and are specified in the business class files using *annotations*. *Annotations* are type-safe metadata about a field/method/class. Annotations in the code begin with @ symbol. Figure 5-23 shows the JPA annotations added to Logical Reader ontology class to make it persistent.

After adding JPA annotations into all persistent classes we then defined a *persistent unit* (PU). A persistent unit defines a group of entities that are associated with single application and that are stored in a single database. A *persistent unit* is defined in the configuration file called *persistence.xml*. This file contains a list of the entity classes in the application; defines the name of the entity unit; and defines the database connection

properties. Database connection properties include the username and password for database connection, the database connection string, and the driver class name. A simplified portion of our prototype middleware PU, showing only few classes added into PU is shown in Figure 5-24. The provider element declares the class file that provides the initial factory for creating an *Entity Manager* instance. An *Entity Manager* provides methods to begin and end transactions, to persist and find entities, to merge and remove entities as well as to create and execute queries. The persistence provider knows which application classes to map to the relational database by reading entity names from the persistence.xml file.

```
package RFIDMiddlewareOntology;
import ...
/**...*/
@Entity
@Table(name = "logicalreader")
public class LogicalReader extends EventProcessingConcept
        implements java.io.Serializable {
    /**...*/
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "logicalReaderID")
    private int logicalReaderID;
    public void setLogicalReaderID(int value) {...}
    public int getLogicalReaderID() {...}
    /**...*/
    @JoinColumn(name = "locationID", referencedColumnName = "locationID")
    @OneToOne
    private Location_description locationDescription;
    public void setLocationDescription(Location_description value) {...}
    public Location_description getLocationDescription() {...}
    /**...*/
    @OneToMany(mappedBy = "logicalReader")
    private List<RFIDReader> readers = new ArrayList();
    public void addReaders(RFIDReader elem) {...}
    public boolean removeReaders(RFIDReader elem) {...}
    public void clearAllReaders() {...}
    public Iterator getAllReaders() {...}
    public List getReaders() {...}
    public void setReaders(List l) {...}
```

Figure 5-23: JPA annotations added to Logical Reader ontology class to make it persistent

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.o
  <persistence-unit name="RFIDMASPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>RFIDMiddlewareOntology.CommunicationInterface</class>
    <class>RFIDMiddlewareOntology.LogicalReader</class>
    <class>RFIDMiddlewareOntology.RFIDAntenna</class>
    <class>RFIDMiddlewareOntology.RFIDReader</class>
    <properties>
      <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
      <property name="eclipselink.jdbc.password" value="lmassawe"/>
      <property name="eclipselink.jdbc.user" value="root"/>
      <property name="eclipselink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="eclipselink.jdbc.url" value="jdbc:mysql://localhost:3306/rddmdb3"/>
    </properties>
  </persistence-unit>
</persistence>
```

Figure 5-24: Simplified portion of the middleware PU file with a few entity classes

5.3 Middleware System Implementation

The designed middleware agent classes are then coded by extending JADE library classes and then deployed in the JADE platform.

JADE [41] is a framework that facilitates the development of the multi-agent systems. It includes: a *runtime environment* where JADE agents can "live" and that must be active on a host before one or more agents can be executed on that host; a *suite of graphical tools* that allows administrating and monitoring the activity of the running agents; and a *library of classes* that programmers use directly or by extending them to develop their agents. Jade also has a number of add-ons including 3rd party. The only software requirement to execute the JADE platform is Java Runtime Environment (JRE).

Each running instance of the JADE runtime environment is called a *container* as it contains several agents. A set of active containers is called a *platform*. Platform containers can be distributed over the network. Normally, one container is run on a host; however several containers can also run on the same host. A single special container called *main container* must always be active in the platform. Main container represents the bootstrap point of a platform, it is the first container to be launched and all other containers must join to a main container by registering with it. If another main container is started somewhere in the network it constitutes a different platform to which new normal containers can possibly register. As a bootstrap point, the main container has the

following responsibilities: it manages the registry of all container nodes composing the platform; manages the registry of all agents present in the platform including their current status and location; and hosting the Agent Management System (AMS) and the Directory Facilitator (DF) agents. When JADE platform is launched, the AMS and the DF are immediately created and the Agent Communication Channel (ACC) is set to allow agent communication. AMS exerts supervisory control over access to and use of the Agent platform. It provides a naming service ensuring that each agent in the platform has a unique name. DF provides a yellow page service by means of which agents can register and modify their services and also agents can find other agents providing services they require. ACC controls all the exchange of messages within the platform, including messages to/from remote platforms.

Figure 5-25 shows some of the middleware agents deployed in the JADE agent platform distributed over several containers. At start-up, each agent registers to the common, shared middleware ontology and also registers its services with the DF.

The middleware includes: low level RFID data cleaning filter called WSTD to reduce erroneous read data; persistence data model which facilitate efficient application level event generation and query capabilities; and graphical user interface used for specifying, managing and monitoring the deployed devices and for configuring the rules for generating application level events. The agents within the middleware use XML as their content language and hence messages with application clients can be handled using XML and TCP or HTTP message-transfer-binding.

The WSTD is an adaptive, sliding-window cleaning method which was developed for cleaning captured erroneous RFID data streams in this study. The WSTD data filter is discussed in detail in Chapter 6. The RFID related object history containing all current and past object data and their relationships are implemented using relational database tables using the model in Figure 5-26 and stored in a MySQL database. The RFID data model and its support for event generation are discussed in detail in Chapter 3.

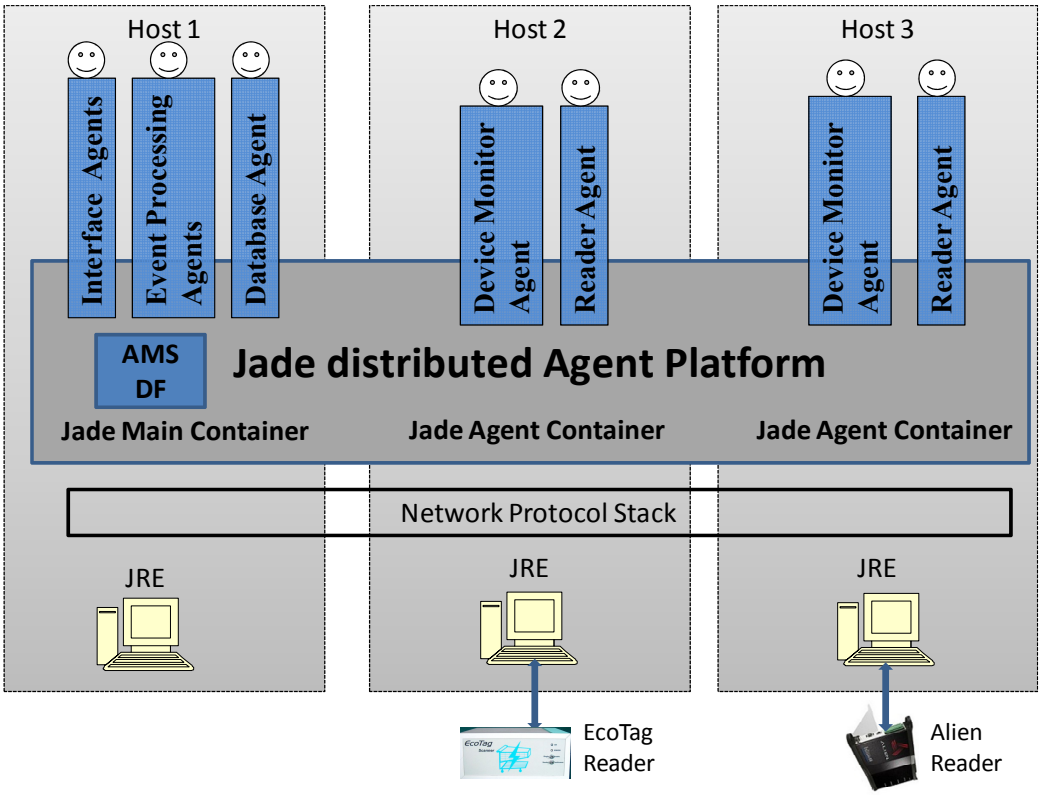


Figure 5-25: Middleware agents deployed in the JADE Agent Platform distributed over several containers

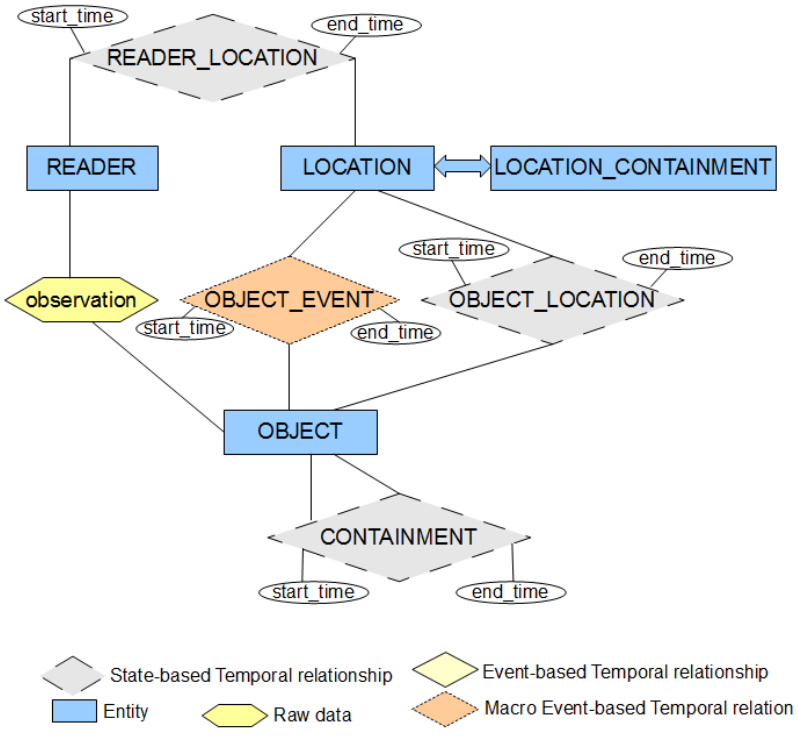


Figure 5-26: Middleware data model

Figure 5-27 shows the reader manager GUI which is used for specifying, managing and monitoring the deployed RFID readers. The middle panel of the window shows the list of deployed readers grouped by their reader manufacturer. The bottom panel shows different navigation tabs to perform different tasks. In addition to registering the reader the user can start and stop a particular reader agent, check the operation status and connection status of the reader. Figure 5-28 shows the window for configuring more than one reader to a one logical reader. Logical reader represents the symbolic location model in the data model.

The middleware supports the following kinds of queries:

- Get all objects situated in a given location over a certain time
- Get all locations of a given object over a certain time
- Get the number of objects in a given location over a certain time grouped by the object type
- Get the number of a particular object in a given location over a certain time.
- Get all objects which are removed or added in a given location. This query compares the present list of the current scanned objects in a location with the previous last stored location objects
- Get the number of objects which are removed or added in a given location grouped by the object type. This query compares the present list of the current scanned objects in a location with the previous last stored location objects

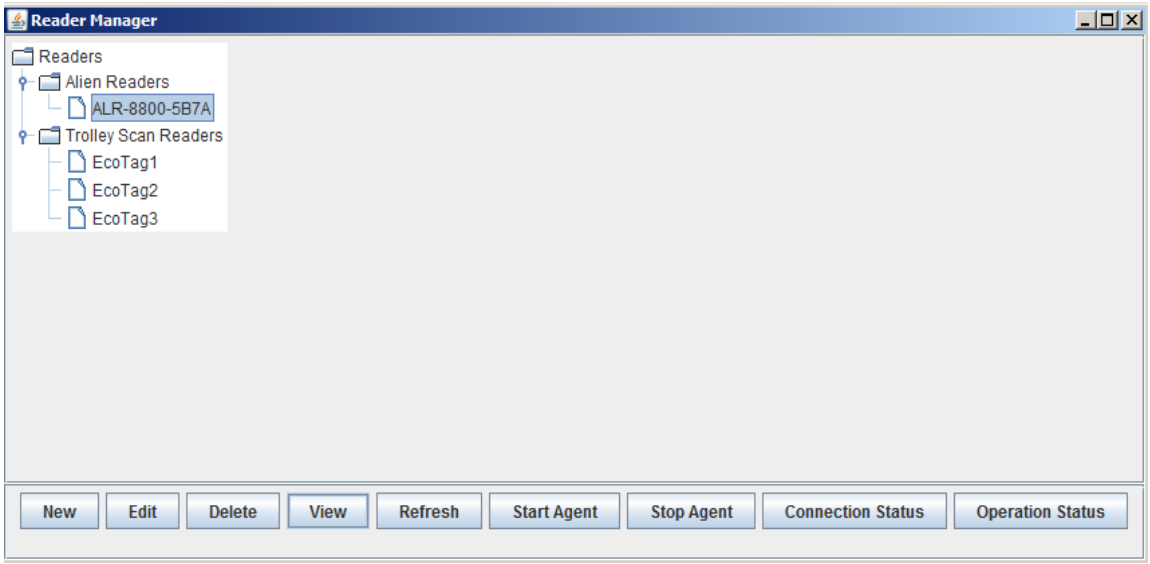


Figure 5-27: Reader Manager GUI for registering, managing and monitoring registered devices

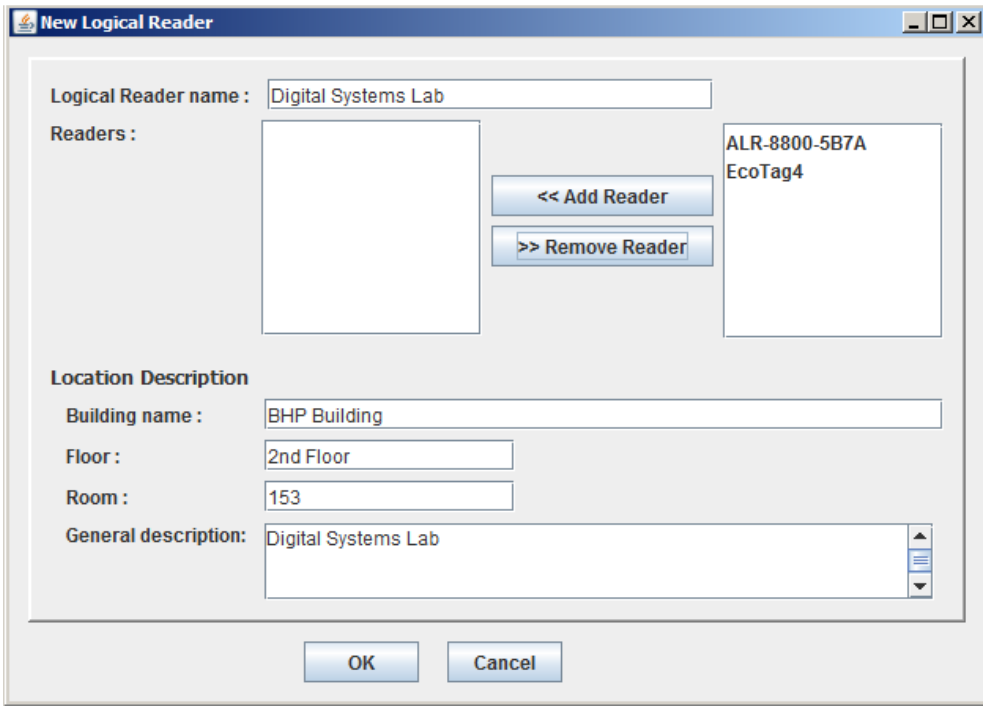


Figure 5-28: GUI for configuring new Logical reader

Figure 5-29 shows a GUI for configuring an application level event generation rules. Event configuration (also known as event specification) involves specifying three main parameters: event boundary; the location to be scanned; and what type of data should be reported (also known as event report specification). Event boundary includes the two parameters event duration and event repeat period. Duration boundary is the interval of real time it takes to accumulate the data; for example, duration of 10 seconds will accumulate the tag reads for 10 seconds and combine them together as one event. Repeat period specify the time interval for the event report to be reported e.g. repeat period of 10 minutes will generate and report the event reports every 10 minutes. In the implementation discussed here, logical reader is synonymous with location in the data model. Logical reader is logical name given to the combination of one or more readers that are monitoring the same location. Event report specification defines the type of data to be reported from the collected event tag reads. Event specification can include one or more event report specifications.

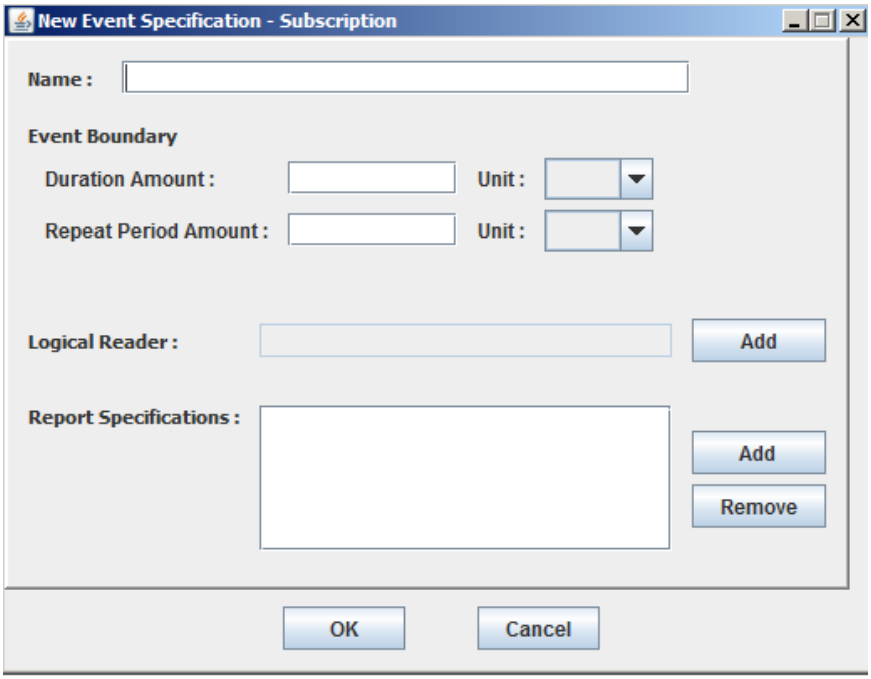


Figure 5-29: GUI for configuring the event generation rules

Figure 5-30 shows the GUI for configuring the event report specification. Event specification includes report name, set of reports to be included, output specification, filter patterns and group pattern. Set of reports specifies whether to report all the current identified objects in the location, added objects in the location, or removed objects from the location compared to the previous last known objects which were located in that location. The output specification includes the option of whether to report the objects Id's, the number of objects, to report only on change or to send the report even if the generated report is empty. Filter patterns specify the type of object to include or exclude from the generated report and group pattern specify how the event data after filtering should be grouped. Figure 5-31 shows the low level event report generated by the middleware. The report includes the total number of objects and the list of objects Id's and the name of the location where they are situated.

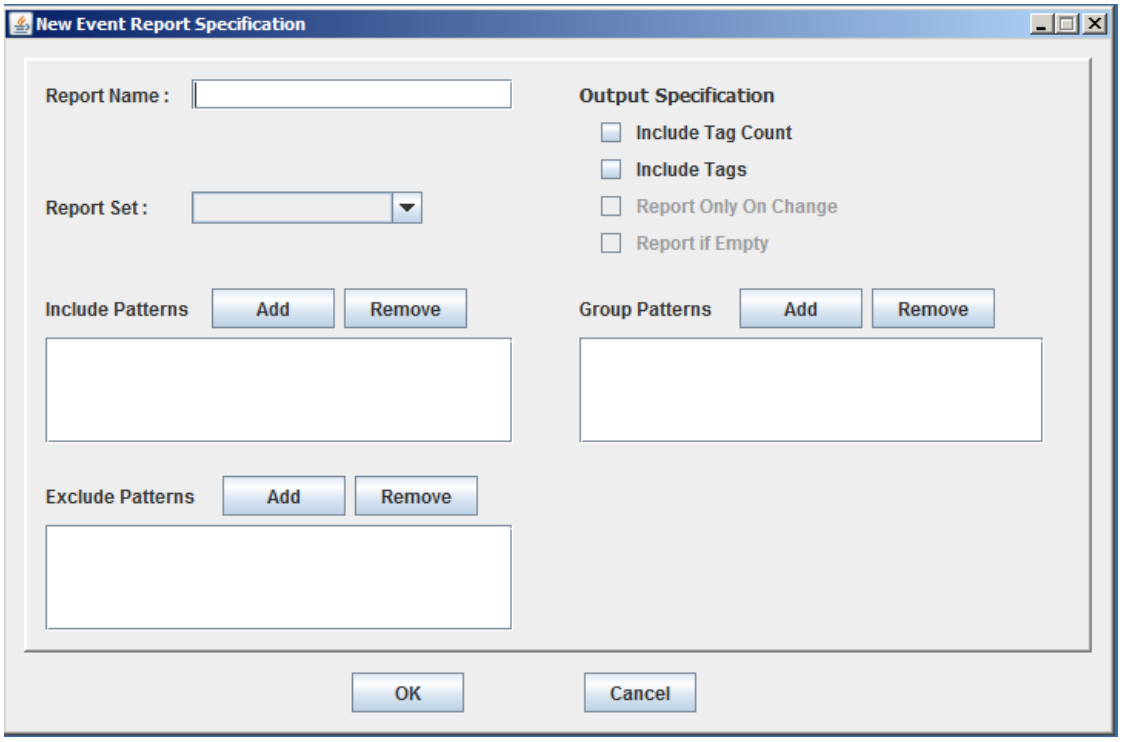


Figure 5-30: GUI for configuring the event report specification

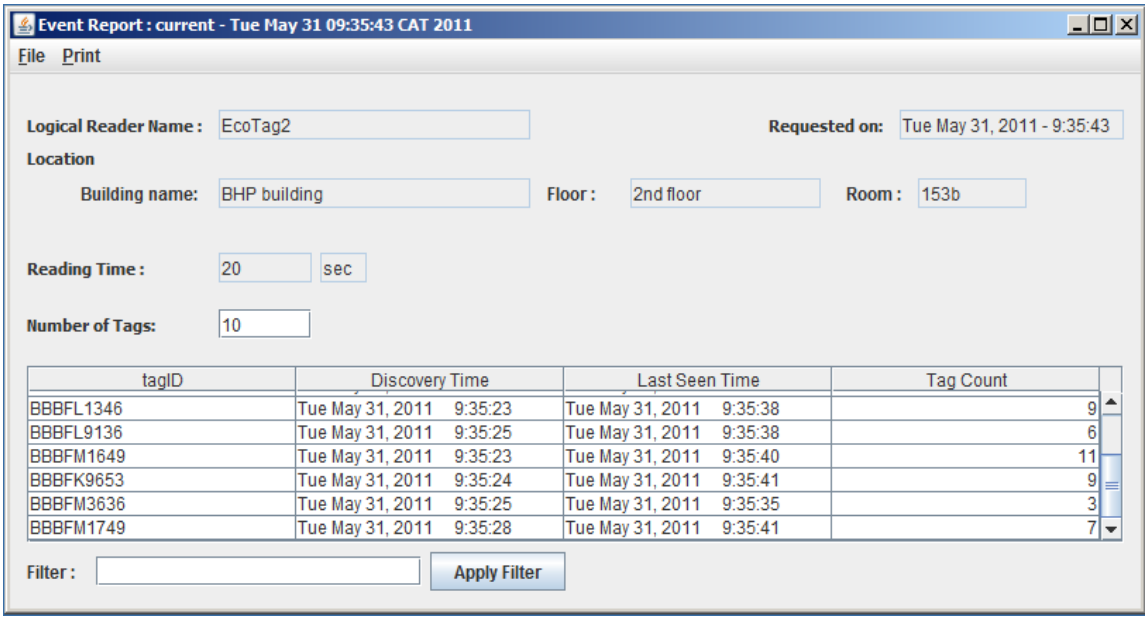


Figure 5-31: GUI showing the low level event report generated by the middleware

Chapter 6: RFID Data Stream Cleaning

6.1 Factors Affecting the Performance of RFID Systems

The main factors that affect the performance of RFID systems are the environment in which they are operating; the materials onto which the tags are attached; and the operating frequency.

The medium through which the tags and the reader communicate is called the channel. The environment in which the system is deployed highly affects the communication channel between the tags and the reader due to effects such as attenuation, multi-path, and interference from other readers and RF devices.

The physical objects to which the tags are attached also affect the tag's performance. Many common materials that the tags are attached to such as metal, metalized/foil-lined packaging, carbon and graphite-impregnated plastics and water containing objects have considerable effects on the performance of the tags. Figure 6-1 shows how the radio frequency waves transmit through different types of materials. This Figure does not consider limitations of speed and distance. Materials that are lucent react very well in all frequency ranges. For non-lucent materials, the radio waves travel through the object with significant distortion in high frequency ranges. Changes in impedance bandwidth, detuning of the antennae, and the reduction in the efficiency of the antennae are some of the factors that change the amount of power being delivered from the antenna to the chip.

The performance of RFID system is also frequency dependent. The ISM band in UHF frequencies varies among countries, for example the ISM band frequencies are 860 - 868 MHz in Europe and Africa, 902 – 928 MHz in USA and Canada, and 950 – 956 MHz in some Asian countries. Thus if the tag has to be read globally, it should operate well across the spectrum. Although tags can operate across the entire ISM band spectrum, their performance across the spectrum is not the same.

To provide a reliable, unbiased and independent source of information that end users can employ to make decision about which RFID tags are likely to perform best on particular product type, the RFID Alliance Lab [98], a not-for-profit facility, conducted more than

5,000 tests on the 10 commercially available UHF EPC tags. Their findings are published into two commercial reports [99], [100].

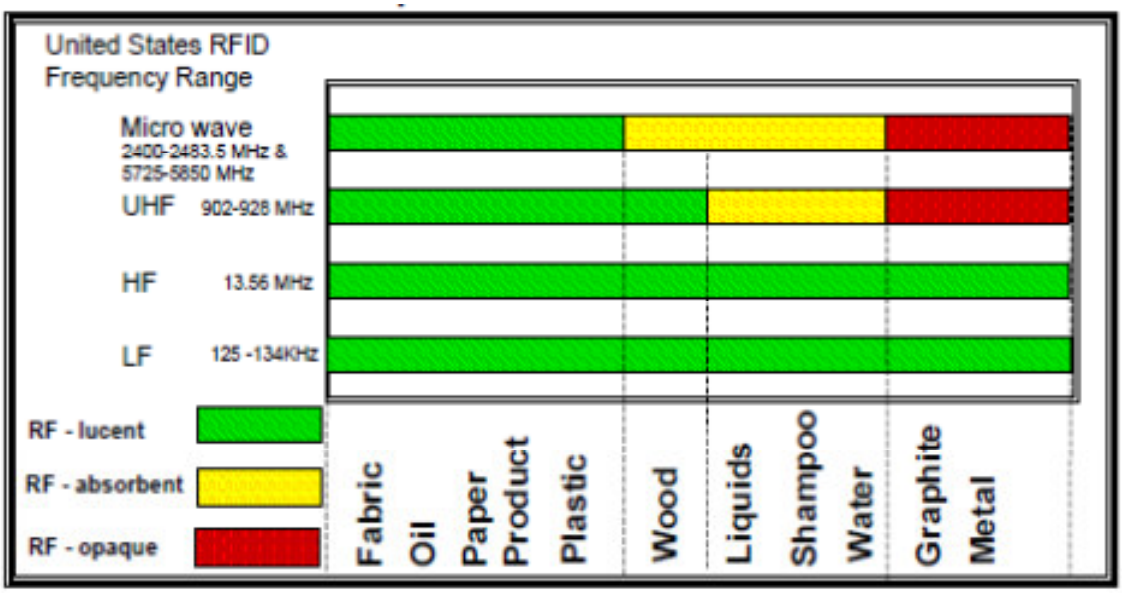


Figure 6-1: Example of material characteristics [97]

These reports were aimed at taking much of the hype out of advertisements and giving users better information for making tag purchase decisions. Although their performance analysis is for UHF RFID tags based on EPCglobal's Class 0 and Class 1 specifications only, they still give good insights for understanding the performance characteristics of RFID tags in general. What follows is a summary of some of their noteworthy findings, which tie in with the subsequent discussion on the importance of cleaning RFID data streams and the best method to use for cleaning these data streams.

a) Maximum distance

Every tag-reader system has a maximum read range; however, tag performance deteriorates with distance. The tags within the maximum read range of the reader, do not respond to all the attempts from the reader to read the tag. There are three distinct regions of operations of passive RFID tags: strong-in-field, weak-in-field and out-of-field.

In the strong-in-field region, the tag responds to most of the attempts from the reader. Thus, the response rate in the strong-in-field region varies between 100% and 77%. The tag performance then degrades gradually with increasing distance in the weak-in-field region. In the out-of-field region, the response rate goes down to 0%.

Figure 6-2 shows a typical response rate vs. distance for commercial UHF RFID tag, and Figure 6-3 shows the read rate vs. distance for different types of commercial UHF RFID tags. The figures provide two scales along the X axis. The first, along the bottom of the graphs, is attenuation in dB. The second, along the top, is an approximate translation of attenuation in dB into distance in feet, based on a standard formula for how a signal attenuates over distance using the *Friis transmission Equation* [101]. To better control the condition of the tests, the tag was placed at fixed distance (3 feet above the reader antenna in free air) and the change in distance simulated by varying the signal attenuation. Increasing signal attenuation simulates the increasing read distance.

Response rate/Read Rate is the ratio of the number of times a tag is read successfully to the number of times the reader attempts to read the tag. Response rates are measured for increasing attenuation levels, and attenuations are increased until the response rate drops to 0%.

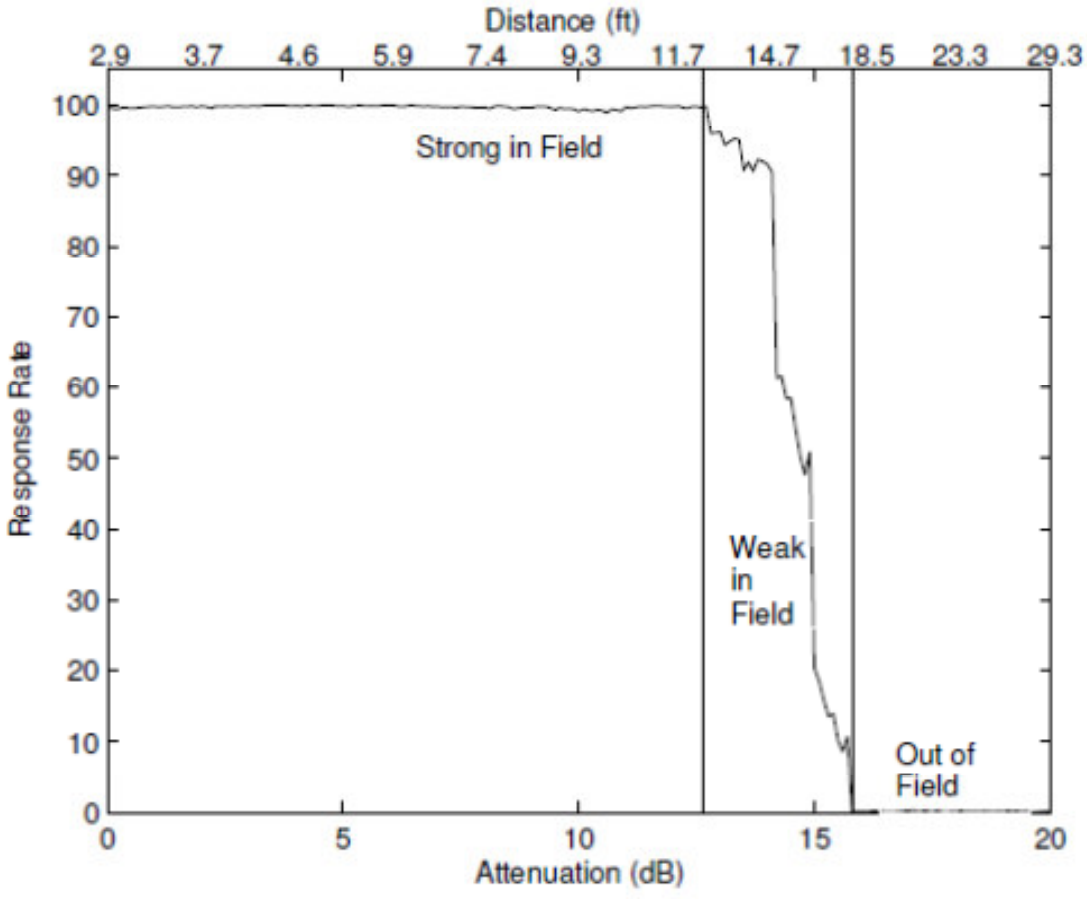


Figure 6-2: Typical response rate versus distance for a passive RFID tag [99]

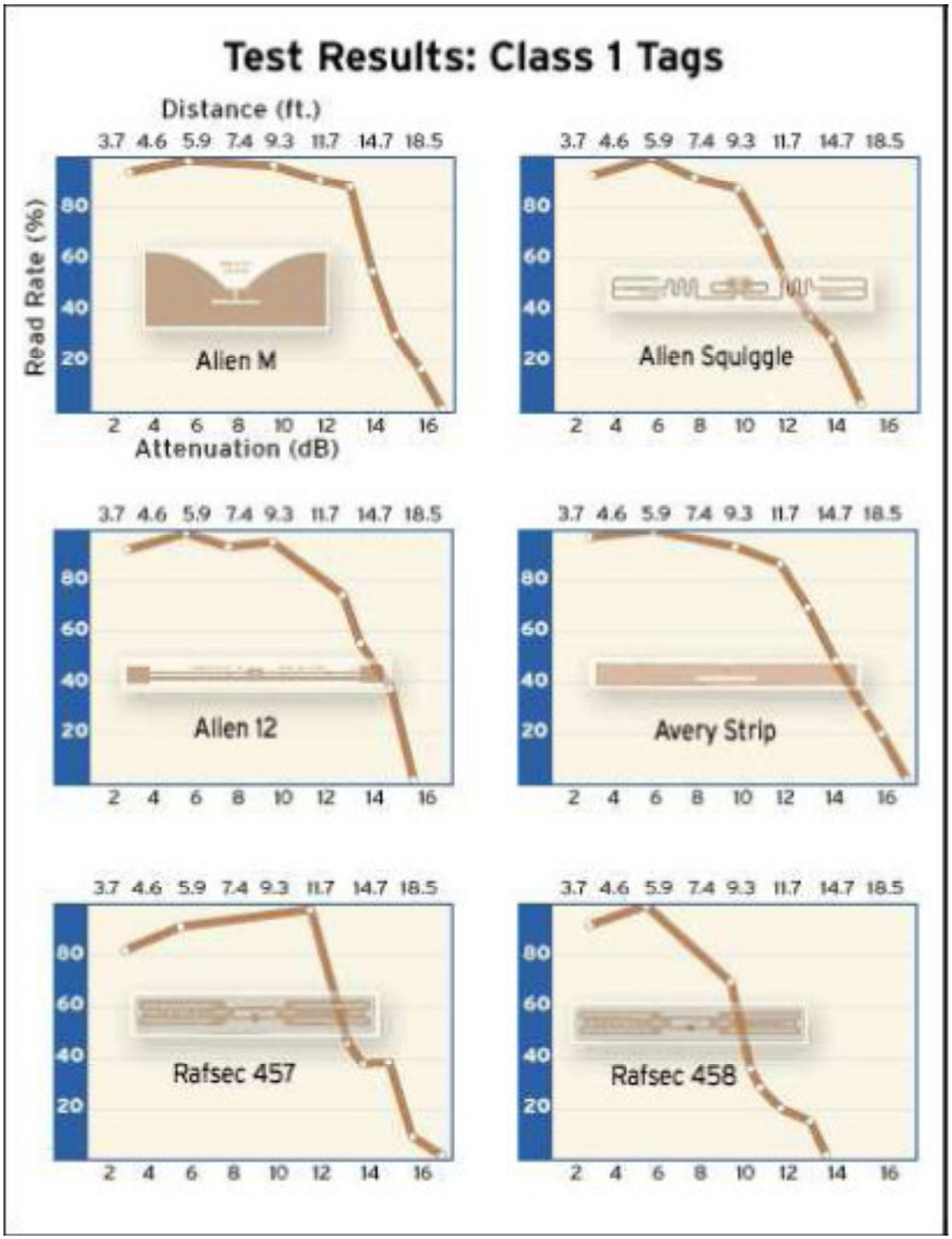


Figure 6-3: Read rate vs. distance for different types of commercial UHF RFID tags [99]

b) Tag Orientation sensitivity

The radiation pattern of The RFID tag antenna determines the ability to read the tag in any orientation. UHF passive tags can be classified into two categories: the tags

which are based on a single dipole or slot antennae, and tags which are based on dual dipole antennae. The single dipole tags are typically long and thin tags and the dual dipole tags are squarer in shape.

Dipole antennae receive and emit radiation at best when perpendicular to its axis and not along its axis. Typical dipole designs have a null reading zone on their antenna tips which means there is a read angle in which one cannot read the tag (dead spots). Figure 6-4 shows the radiation pattern of Alien ALN-9640 Squiggle inlay tag with a dipole antenna.

A dual dipole tag has two dipoles oriented in perpendicular directions so that if we are looking at the null of one antenna, the second antenna is at the best receiving orientation eliminating the null zones [102]. Figure 6-5 shows the radiation pattern of Avery Dennison AD-833 inlay tag with dual dipole antenna.

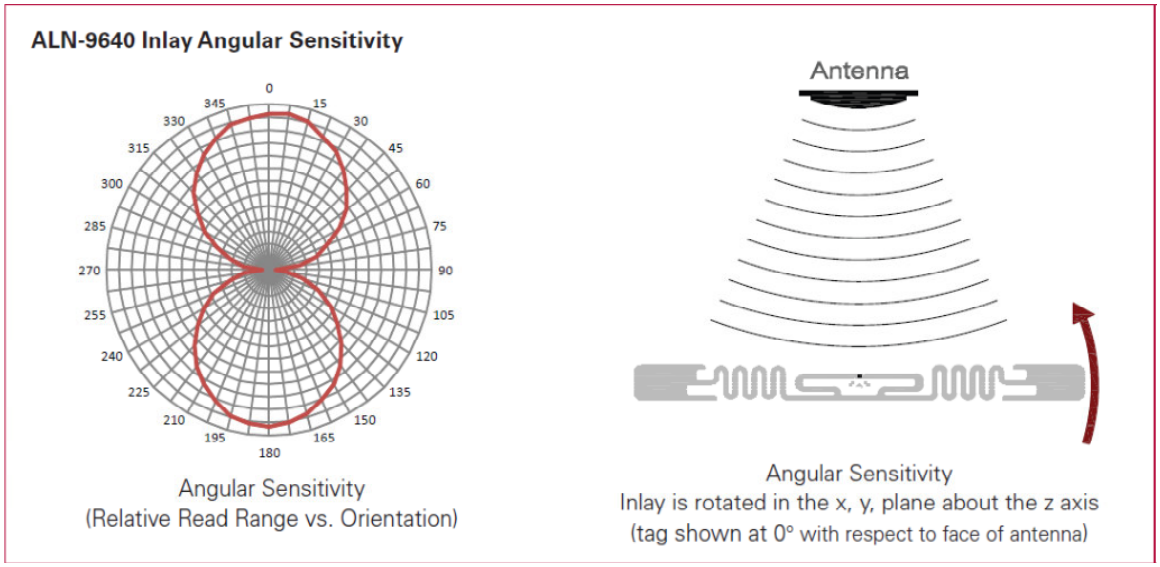


Figure 6-4: Angular sensitivity of Alien ALN-9640 UHF tag using dipole antenna [103]

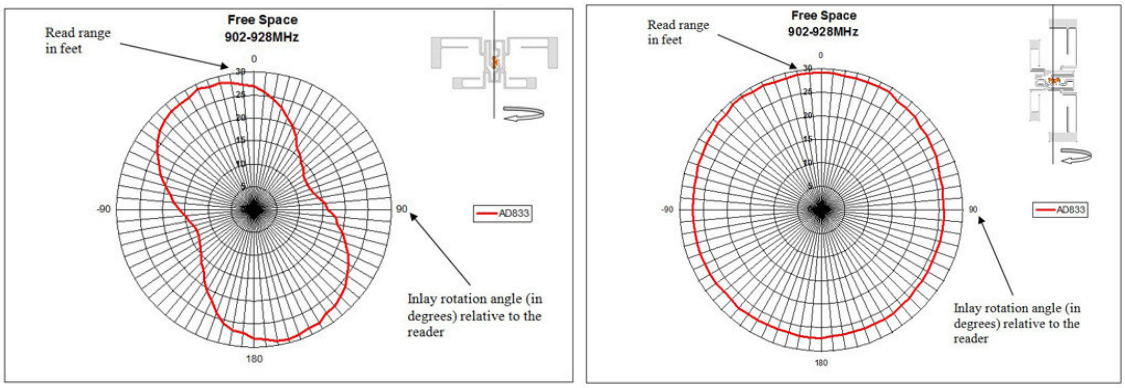


Figure 6-5: Angular sensitivity of Avery Dennison AD-833 UHF RFID tag using dual dipole antenna [104]

c) Variance in performance

Tags within the same model are expected to give similar performance. However, it is an unfortunate reality that there are considerable variations in performance from one tag to another, even among tags of the same manufacturer and model. For example, Alien Squiggle tag model ALL-9238 was found to have a difference of 3.5 dB between the worst and the best tags, which means that the worst-performing tag requires more than twice the amount of RF power from the reader to be read, or can be read at only 60 percent of the distance of the best-performing tag. Inconsistencies in tag performance increases complexity in designing RFID-enabled applications.

d) Distance of tags in front of metal /water

Aqueous products absorb RF energy and reduced the RF signal that reaches a tag, while dielectric and metal materials detune and reflect RF signals, which lead to a reduced read range, phantom reads or no read signal at all. All tested commercial tag models were unreadable within 2.5 cm of separation from metal and water (see Figure 6-6).

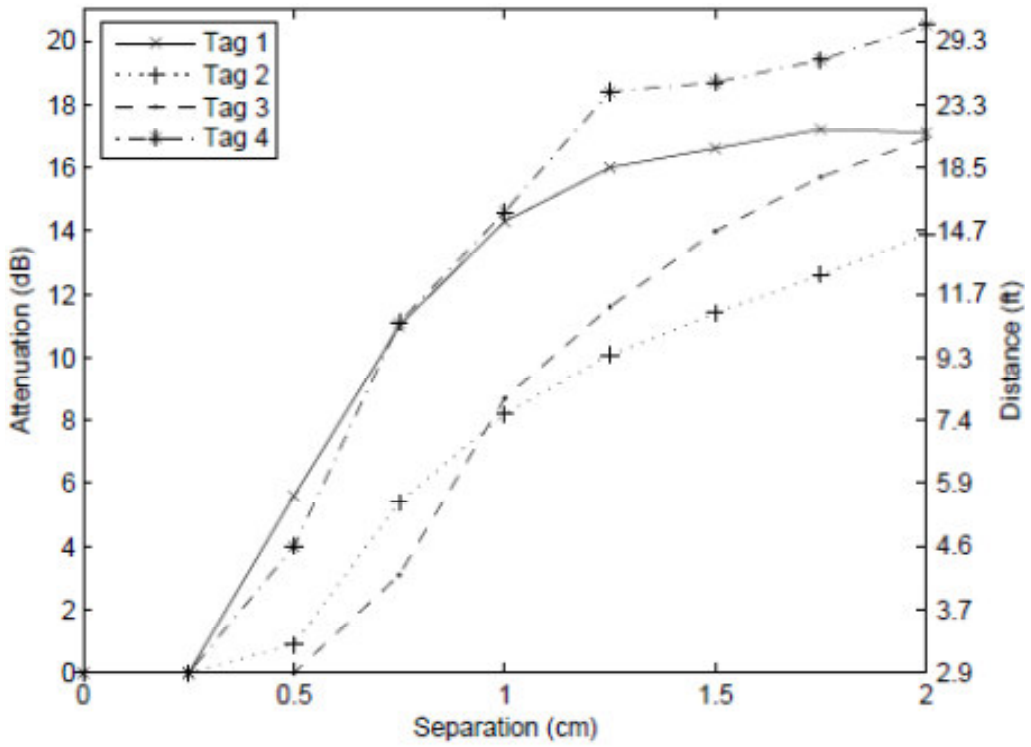


Figure 6-6: Tag performance in front of metal [100]

e) Frequency response of tags in front of the metal/dielectric materials

The presence of metal and dielectric materials near a tag affects its frequency response. Some of the tags are unreadable at certain ISM bands when the tags are near metal, even though they are readable at all ISM bands in free air. Figure 6-7 shows a frequency dependent response of two tags. Tag1 shows a normal typical frequency variation response in which it performs better at lower frequencies and moderately degrades with increasing frequency. Tag2 shows an extreme behaviour in which it performs better than Tag1 in lower frequencies less than 928 MHz but it is unreadable at the higher frequency of 955 MHz.

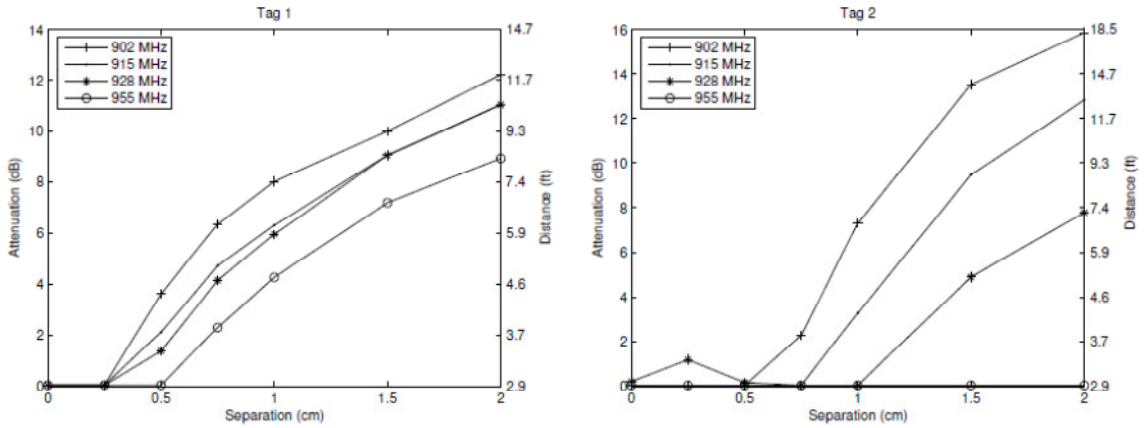


Figure 6-7: Frequency-dependent performance of the tag in front of a metal material [100]

Other factors which also affect tag-reader system performance is the number of tags in the reader's range and the speed in which the tagged object moves as it passes through the detection zone.

6.2 Errors in RFID Data Streams

As can be observed from section 6.1, the tag-reader system performance is highly dependent on the environment in which the system is deployed, materials in which the tag is attached to, the orientation sensitivity of the tag, and the distance between the tag and the reader. In addition, inconsistencies in tag performance even for the same tags' model from the same manufacturer make the tag performance unpredictable.

These performance limitations cause the data streams produced by these systems to be extremely unreliable and essentially useless in many application areas. There are three typical undesired scenarios: false negative readings, false positive readings and duplicate readings.

- False negative readings

  RFID tags while present in the reader's detection range might not be read by the reader at all. This can be caused by a number of reasons.

  i) RF collision in the presence of multiple tags placed too close to each other. When tags are placed too close together, tag antennas can detune each other, thereby reducing the chance of reading by minimizing their chances of being activated.

ii) The material and the environment in which the tag is deployed such as presence of water, metal or RF interference from other RF devices.

iii) The distance between the tag and the reader, as has been discussed in the previous section. The detection rate deteriorates with distance and the tags in the weak-in-field region (far end of detection range) have a smaller chance of being read compared to the strong-in-field region (near the reader antenna).

iv) The tag orientation in the dipole antenna tags. When the tag is aligned with its null reading zone, the tag will not be read, and when it is not properly vertically aligned, its chances of being read decreases.

- False positive readings

In addition to the normal RFID tag reads, unexpected readings are generated. This can be caused by the following reasons:

i) RFID tags outside the normal reading scope of the reader are captured by the reader. For example, while reading items inside the office, a reader may read the tag carried by the person walking by the door.

ii) For unknown reasons from the reader or environment, a reader may send a wrong or phantom identification code.

- Duplicate readings

RFID naturally generate a large number of duplicate readings. This can be caused by the following reasons:

i) To enhance reading accuracy, some readers perform several interrogation cycles during one read cycle and as a result the tag with the same id can be read more than once.

ii) Also, to enhance reading accuracy multiple tags with the same id can be attached to the same object.

iii) Multiple readers installed to cover large area may lead to multiple reading of the same tag by different readers in the overlapping areas.

iv) Tags in the scope of the reader for a long time are read by the reader multiple times.

Although the performance of UHF passive RFID based systems improved significantly by introduction of EPC Class-1 Generation-2 protocol (C1G2) [105], several studies on the performance of the C1G2 RFID systems indicate that the overall performance of the system is still implementation dependent [106]-[108], [109]. The empirical study of UHF RFID performance by Buettner *et al*. [107] shows that physical effects such as errors and multipath are significant factors that degrade the overall performance of commercial readers. These effects increase both the duration of each reader cycle and the number of cycles to read all tags in a tag set. They argue that the error rates are highly location dependent and the level of degradation is implementation specific. The work by Kawakita *et al*. [109] shows that the bit errors, due to erroneous communication links, significantly degrade C1G2 performance. In actual UHF passive RFID deployment, the RFID's usually shares the frequency band with other UHF wireless devices as well as neighbour RFIDs. While some interference is predictable and controllable some are unpredictable and uncontrollable - such as mobile wireless devices. Therefore, despite the improvements on tag detection rates by using C1G2 protocol, factors such as tag-reader configurations, multipath and unpredictable interferences in the deployment environment still contribute to degradation of the performance and reliability of the RFID system leading to noisy and incomplete data. RFID data cleaning is, therefore, essential in order to correct the reading errors, and to allow these data streams to be used to make correct interpretations and analysis of the physical world they are representing.

6.3 RFID Data Cleaning Approaches

Methodologies for improving reliability of RFID data proposed in the literature can be divided into three main categories: physical solutions, middleware solutions and deferred solutions [110]. Physical solutions include improvement of hardware performance to improve the reliability of the data such as [111] and use of redundant techniques by using multiple tags and readers to identify the same object [112], [113]. Middleware solutions include algorithms to correct the incoming sensor data streams before the data is passed into the database [10], [114], [115]. The deferred solutions incorporate intelligent techniques, which correct the data in the later stages within the data storage [110], [116]. The research presented here falls in the category of middleware-based solutions;

specifically window-based smoothing methods. Temporal based smoothing-window filters are the most commonly used RFID data cleaning mechanisms, [4], [10], [16] [117]. It was decided to use the window-based method because of its simplicity and this piece of work extends the work proposed by Jeffrey *et al*. [10].

In temporal based smoothing filters a time based sliding-window is used to interpolate for the lost readings from each tag within the time window. The goal is to reduce or eliminate false negative (dropped) readings by giving each tag more opportunities to be read within the smoothing window. Due to the factors which limit the readability of the tags discussed in section 6.1, setting a proper window size is a challenging task, especially when tags are mobile. While bigger windows are necessary to ensure that even the poor performing tag is read, the bigger window can introduce false positive errors caused by inability to capture tag movements. In the large window, the tag can be mistaken as being present while they have already exited the detection range.

The experimental results presented here show that, firstly the window size is application dependent. Bigger windows can be efficient in static tag applications but they are not ideal in dynamic tag applications. This is due to the fact that the lack of reading from the tag may be due to the tag having moved out of the detection field rather than the missed reading. Applications with higher degrees of mobility require small smoothing windows in order to capture rapid changes in measurement data; but at the same time smaller windows are unable to compensate for the missed reading. In such situations, increasing the window size to combat missed reading will not be efficient since the tag dynamics will be lost through aggregation. Secondly, a small change in the operating environment may render the initial window setting unable to clean the data. This raises an important requirement for an adaptive variable window size which changes as the environment changes. Thirdly, due to variation in tag performance as mentioned earlier, we need to have a per tag cleaning mechanism. A per tag cleaning mechanism considers each tag individually and sets its cleaning window accordingly. Fourthly, even in the static tag applications knowing when a window is big enough to clean the data while optimizing performance and efficiency is a non-trivial task for an application to handle. Therefore, taking into consideration all these requirements for setting the optimal window size, this task should be included and automatically be carried out by the middleware instead of being left to the application programs to set the cleaning window size.

6.3.1 Fixed Window Filters

There are many different types of windows, but we may basically categorize them all according to two axes of references:

a) The *window policy,* which configures the way the window expires. There are basically two main policies: sliding and tumbling, as well as a small variation of the tumbling window, which are called landmark windows.

b) The *window size,* which defines the size of a filter window according to some metric. There are two types of sizes: time-based and tuple-based.

These two axes are orthogonal to each other; for example, one can create a time-based sliding window, or a tuple-based tumbling window.

Sliding window

A sliding window contains the events received within a fixed-size interval. This interval can be temporal or tuple-based. For example, a time-based sliding window with an interval equal to three minutes will contain only the events received during the past three minutes. As soon as the event becomes older than this threshold, it is expired from the window. The window continuously moves forward as the time advances and its size never changes. Figure 6-8 illustrates the sliding window with the size of three minutes. When the new tuple arrives in the tuple-based sliding window, the oldest of the three expires and the new tuple is appended to the window. Every insertion corresponds to a deletion of the oldest element. In a time-based sliding window there is no restriction on the number of insertions and deletions, typically each element is associated with a timestamp, and the window contains all elements with active timestamps.

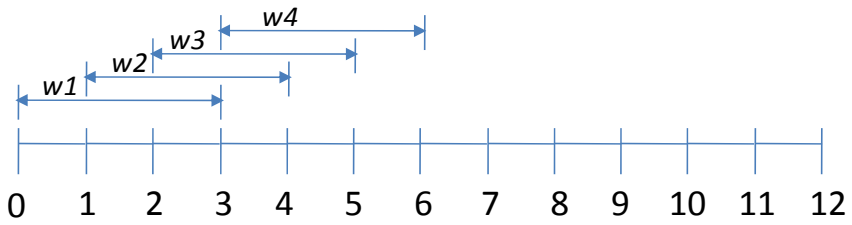


Figure 6-8: Time-based sliding window

Tumbling windows

Unlike sliding windows, tumbling windows expire all events at once, once they reach their predetermined size. Afterwards, they refill again and the cycle repeats itself. For

example, in a time-based tumbling window of three minutes, the window will expire every three minutes and the count will reset to zero no matter how many elements are actually inside it. Figure 6-9 shows an illustration of a time-based tumbling window. Tumbling windows are particularly useful if the value at the time the window expires is the priority (that is, if one wants to sample the value using periodic interval) and not all the values produced between those intervals.

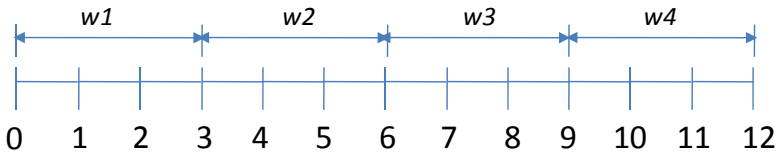


Figure 6-9: Time-based tumbling window

Landmark windows

Landmark windows are similar to time-based tumbling windows, except that one can control the exact moment the window will expire. The motivation behind this is that sometimes you want the window to expire at well-defined moments, not just ten minutes after the application starts. For example, you may want the window to expire at the end of the day, to collect daily statistics, or at the 30th minute every hour. To set-up a landmark window, one needs to schedule its expiration.

In fixed window filters, the reader data stream is collected for the duration of the window time and then the data is analysed. In the fixed tumbling window, if there is at least one occurrence of the tag during the window period, the tag is considered present during the whole window period. In the fixed sliding window, the output of the window corresponds to the time the window expires. If there is more than one reading of the same tag, the readings are aggregated into a single tag reading such as the tuple $(tagID, discoveryTime, lastSeenTime, tagCount)$.

6.3.2 Variable Window Filter

Variable window filters are filters which continuously adopt the size of the smoothing window accordingly in order to improve data accuracy. Adaptive variable window data

cleaning for RFID data streams was initially proposed by Jeffery *et al*. [10]. Their proposed method is known as SMURF *(Statistical sMoothing for Unreliable RFid data*). SMURF models the unreliability of RFID readings by viewing RFID streams as a statistical sample of tags in the physical world, and exploits techniques grounded in sampling theory to drive its cleaning processes. SMURF contains two primary cleaning mechanisms aimed at firstly, to produce accurate data streams for individual tag-ID readings (per tag cleaning), and secondly to provide accurate aggregate (e.g. count) estimates over large populations (multi-tag cleaning). It uses the binomial sampling and π estimators to continuously adapt the smoothing window size. This variable sampling-based data cleaning method shows some similarities with the fixed sliding-window method. Like the fixed sliding window method, its window slides by a single epoch. The tag is said to be present if there exist at least one reading for that tag within that window and the output corresponds to the midpoint epoch of the window. The mid-window concept is the intuitive concept of smoothing; it makes it possible to predict the presence of the tag based on the closest previous and future readings of that particular tag. For example, if the tag was present at time $t-1$ and $t+1$, it is mostly likely to be present at time t. In addition to these features, it includes two processing modules which make it different from other window-based methods. The first module is a sliding window processor, which dynamically adjusts the window size based on statistical properties of the data. The second module is an optimization mechanism for improving cleaning effectiveness by detecting movement of tags.

Based on the evaluation of the performance of RFID described in section 6.1, and the results of our experiments in which we compare the fixed and variable window smoothing algorithms, we concluded that adaptive window based smoothing algorithm is the best way to clean the RFID data streams. Adopting and extending the statistical approaches proposed in SMURF, we developed our own adaptive cleaning scheme for RFID data streams, called WSTD, with a more efficient transition detection mechanism. WSTD is able to automatically and continuously adapt its window size to cope with fluctuations of the tag-reader performance due to changes in the environment while relatively accurately detecting the transition points. The following subsection describes in detail our adaptive sliding-window cleaning mechanism, WSTD.

6.4 RFID Data Stream as a Statistical Sample

Readers interrogate nearby tags by sending out RF signals and the tags which are within the reader's interrogation zone (reader read range) responds to these signals with their unique identification code. An interrogation cycle is a single iteration through the reader's protocol that attempts to determine all tags in the reader's vicinity. The result of multiple interrogation cycles are grouped together into one read cycle. A read cycle is the smallest interaction between RFID reader and the middleware. Read cycle is typically specified in time units or in terms of number of interrogation cycles. The result of each read cycle is the list of detected tags, together with additional information such as the number of times that the tag responded, the time at which the tag was first read, the last time the tag was read, or the antenna that read the tag. Additional information differs from one reader to another. Table 6.1 shows an example of a typical read cycle output using Alien Reader ALR-8800 scanned for 10 interrogation cycles. In the discussion here an atomic unit of time used by one read cycle will be referred to as an epoch and in the implementation an epoch corresponds to 1 second. A smoothing window is therefore made up of a sequence of consecutive epochs.

Table 6.1: Example of reader tag list

ID:E200 3411 B802 0111 6519 5246, Discovered: Tue Sep 13 15:18:07, Last Seen: Tue Sep 13 15:18:07, Reads:4
ID:E200 3411 B802 0111 6519 5256, Discovered: Tue Sep 13 15:18:07, Last Seen: Tue Sep 13 15:18:07, Reads:1
ID:E200 3411 B802 0111 6519 5255, Discovered: Tue Sep 13 15:18:07, Last Seen: Tue Sep 13 15:18:08, Reads:10

According to the RFID reader-tag performance analysis presented in section 6.1, it is an undeniable fact that, the raw RFID data streams do not provide a correct representation of the physical world that they are representing. A significant number of tags which are within the reader's read range are not read by the reader due to either tag orientation, tag distance from the reader antenna, presence of metal, dielectric or water material close to the tag and other factors discussed in section 6.1. These missing tags imply that typically only a *subset* of the tag population is actually observed. Therefore, the observed RFID readings can be viewed as a random sample of the population of tags in the physical world. The key insight is viewing each epoch output as a sampling trial and the smoothing window readings as repeated random sampling trials.

Let N_t denote the unknown size of the underlying tag population at epoch t and let $S_t \subseteq \{1, \dots, N_t\}$ denote the subset of the tags observed ("sampled") during that epoch. S_t can be viewed as, unequal probability of a random sample of the tag population. Probability $p_{i,t}$ of selecting tag i at epoch t can be calculated from the epoch t output information (Table 6.1) using the number of reads (tag responses) for tag i in combination with the known number of interrogation cycles (number of requests) Equation (1).

$$p_{i,t} = \frac{number\ of\ responses}{number\ of\ requests} \tag{1}$$

For example, for the epoch output in Table 6.1, in which each epoch accumulates data for 10 interrogation cycles, the sampling probabilities for the first, second and third tags are 0.4, 0.1 and 1 respectively. These probabilities vary across the tags and can also vary over time as the observed tags move within the reader's detection range as well as due to the changes in the operating environment.

6.5 Adaptive Individual Tag Cleaning

6.5.1 Completeness Requirement

Each epoch is viewed as an independent Bernoulli trial (i.e. a sample draw for $tag\ i$) with success probability $p_{i,t}$ using Equation (1). This implies that the number of successful observations of $tag\ i$ in the window W_i with w_i epochs (i.e.$W_i = (t - w_i, t]$) is a random variable with a binomial distribution $B(w_i, p_{i,t})$. In the general case, assume that $tag\ i$ is seen only in subset $S_i \subseteq W_i$ of all epochs in the window W_i. Assuming that, the tag probabilities within an approximately sized window calculated using Equation (1), are relatively homogeneous, taking their average will give a valid estimate of the actual $p_{i,t}$ of $tag\ i$ during window W_i [10]. Therefore, the average empirical read rate p_i^{avg} over the observation epochs is given is given by Equation (2).

$$p_i^{avg} = (1/|S_i|) \cdot \sum_{t \in S_i} p_{i,t} \tag{2}$$

Also S_i can be seen as a binomial sample of epochs in W_i i.e. a Bernoulli trial with probability p_i^{avg} for success and $|S_i|$ as a binomial random variable with binomial distribution $B(w_i, p_i^{avg})$. Hence, from standard probability theory the expected value and variance of $|S_i|$ is given as Equation (3) and Equation (4) respectively.

$$E[|S_i|] = w_i \cdot p_i^{avg} \tag{3}$$

$$Var[|S_i|] = w_i \cdot p_i^{avg} \cdot (1 - p_i^{avg}) \tag{4}$$

The derived binomial sampling model is then used to set the window size to ensure that there is enough epochs in the window W_i such that $tag\ i$ is read if it does exist in the reader's range. Setting the number of epochs within the smoothing window according to Equation (5) ensures that $tag\ i$ is observed within the window W_i with probability $>$ $1 - \delta$ [10].

$$w_i \geq \lceil (1/p_i^{avg}) ln(1/\delta) \rceil \tag{5}$$

6.5.2 Individual Tag Adaptive Window Size Adjustment

WSTD uses binomial sampling concepts to calculate the appropriate window size and π-estimator to estimate the number of tags as proposed by SMURF. WSTD then uses the comparison of the two window sub-range observations or estimated tag counts and some rules to detect when transition occurs within the window, and then adjust the window size appropriately. We first present how WSTD cleans individual tag data and then present how it cleans tag aggregates in the applications which only need to know the number of tags available.

In order to balance between guaranteeing completeness and capturing tag dynamics the WSTD algorithm uses simple rules together with statistical analysis of the underlying data stream to adaptively adjust the cleaning window size.

Assume $W_i = (t - w_i, t]$ is $tag\ i$ current window, and let $W_{1i}^{'} = (t - w_i, t - w_i/2]$ denote the first half of window W_i and $W_{2i}^{'} = [t - w_i/2, t]$ denote the second half of the window W_i. Let $|S_{1i}|$ and $|S_{2i}|$ denote the binomial sample size during $W_{1i}^{'}$ and $W_{2i}^{'}$

respectively. Note that the mid-epoch (i.e. epoch at $t - w_i/2$) is inclusive on both ranges as shown in Figure 6-10.

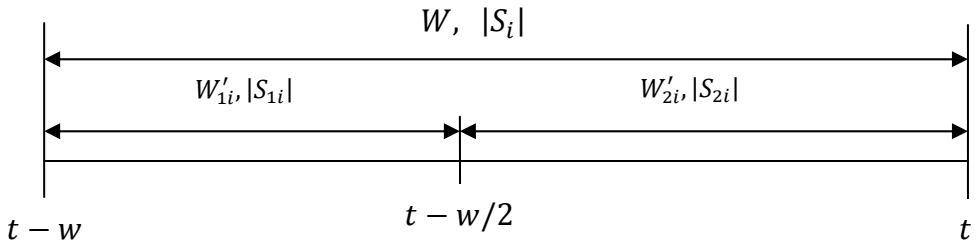


Figure 6-10: Illustration of the sub-ranges in the smoothing window

Rule 1:

Similar to SMURF, variation within the window is detected if the number of observed readings is less than the expected number of readings (Equation (6)) and there is statistically significant variation in the tag observations using the Central Limit Theorem (CLT) according to Equation (7).

$$|s_i| < w_i \cdot p_i^{avg} \tag{6}$$

$$\left||S_i| - w_i p_i^{avg}\right| > 2 \cdot \sqrt{w_i p_i^{avg}\left(1 - p_i^{avg}\right)}\,. \tag{7}$$

However, we noted that this variation within the window can also be caused by missing tags and not necessarily only due to transition. Hence, to reduce the number of false positives due to transition and the number of false negative readings which will be further introduced in case of wrong transition detection, the window size is reduced additively by reducing the window size by two epochs.

To improve the transition detection mechanism for the mobile tags we combine the mobile detection mechanism together with the observations of the second half of the window $|S_{2i}|$ to estimate when the tag is exiting the detection range. The slope of the best-fit line using the least squares fitting with the observed probabilities in the window $(\frac{\Delta p_{i,t}}{epochs})$ is used to determine if the tag is moving out. If the tag is detected with consistently falling $p_{i,t}$, within the window it is inferred as that the tag is moving out. Hence, the negative slope of the best-fit line indicates that the tag is moving out.

Rule 2:

If the tag is moving out and it was not detected in the second half of the window (i.e. $|S_{2i}| = 0$) the tag is assumed to have exited or is exiting the detection range. In this case the window size is halved to reduce the false positive readings.

One weakness of this rule is that premature exit transition detection will also lead to false negative reading due to a small window size.

Rule 3:

The window size is increased if the computed window size using Equation (5) is greater than the current window size and the expected number of observation samples is less than the actual number of observed samples (i.e. $|S_i| > w_i p_i^{avg}$).

Low expected observation samples indicates that the probability of detection p_i^{avg} is low, in this case we need to grow the window size to give more opportunity for the poor performing tag to be detected. Otherwise, if the expected observation sample is equal or greater than the actual sample size it means that, the p_i^{avg} is good enough and we do not have to increase the window size. This rule ensures that the window size is increased only when the read rate is poor.

Figure 6-11 shows a pseudo-code description of the WSTD adaptive per tag cleaning algorithm. Each individual tag is cleaned in its own window. The rules described above are used to adjust the tag's cleaning window size adaptively based on the statistical analysis of the underlying tag observations. Initially, all newly detected tags' windows are set to 1 epoch, the window sizes are then adjusted according to their detection rates with minimum window size set to 3 epochs. Setting the minimum window size to 3 epochs strikes a balance between maintaining the smoothing effect of the algorithm and reducing the false positive errors. In a way similar to SMURF, WSTD also slides its window per single epoch (read cycle) and produces output readings corresponding to the midpoint of the window after the entire window has been read.

Input: T = set of all observed tag IDs
δ = required completeness confidence
Output: t = set of all present tag IDs
Initialize: $\forall i \in T, w_i \leftarrow 1$
while($getNextEpoch$) **do**
  for (i in T)
    $processWindow(W_i) \rightarrow p_{i,t}, p_i^{avg}, |S_i|$
    if ($tagExist(|S_i|)$
      output i
    end if
    $w_i^* \leftarrow requiredWindowSize(p_i^{avg}, \delta)$
    $tagExiting \leftarrow mobileDetection(p_{i,t}s,)$
    if $(tagExiting \wedge |S_{2i}| = 0)$
      $w_i \leftarrow max\ (min\{w_i/2\ , w_i^*\}, 3)$
    else if $\left(detectTransition(|S_i|, w_i, p_i^{avg})\right)$
      $w_i \leftarrow max\ (min\{w_i - 2,\ w_i^*\}, 3)$
    else if $(w_i^* > w_i \wedge |S_i| < w_i p_i^{avg})$
      $w_i \leftarrow min\{w_i + 2,\ w_i^*\}$
    else
      $w_i \leftarrow min\{w_i,\ w_i^*\}$
    end if
  end for
end while

Figure 6-11: WSTD individual tag cleaning algorithm

During each epoch and for each observed tag, the algorithm starts by processing the readings inside the window W_i ($processWindow(W_i)$). This processing includes estimating the required model parameters for $tag\ i$ (i.e. $p_{i,t}$'s, $p_i^{avg}, |S_i|$) using tag list information. By using the estimated tag parameters, the algorithm checks if there exits at least one reading within the window (i.e. $|S_i| \neq 0$) and outputs the detected $tag\ i$. The algorithm then uses the mobile detection module to check if the tag is moving out. If the slope is negative (i.e. the tag is exiting) and there is no tag detected in the second half of the window ($|S_{2i}| = 0$) this means that there is higher probability that the tag has already exited the window. In this case the window size is aggressively reduced multiplicatively to half its size to alleviate the false positive readings which might arise due to large window size.

If the tag has not exited the detection region, the algorithm then tries to check if transition occurred during the window $\left(detectTransition(|S_i|, w_i, p_i^{avg})\right)$ based on Equation (6) and Equation (7). If transition is detected the window size is reduced by two epochs.

If transition is not detected the algorithm uses conditions in *rule 3* to decide whether to increase the window size and set the window size appropriately. Since a midpoint sliding

window is used here, in order to advance the slide point by one epoch, the window must be increased by two epochs (i.e.$w_i \leftarrow w_i + 2$). Because of this condition, the window size should be an odd number.

If the tag is not exiting, and if no transition is detected, and also the current window is large enough to ensure detection with high probability, the tag continues with the same smoothing window size.

6.6 Adaptive Multi-tag Aggregate Cleaning

Some applications do not require information for each individual tags, but only need to track the number of tags in the detection region. These types of applications typically track large populations of tags. For instance, a retail store monitoring application may only need to know when the count of items on the shelf or store drop below a certain threshold level.

The per tag cleaning method could be used to clean tags in such scenarios, whereby each tag in the population is individually cleaned and their result is aggregated across individual smoothing filters for each epoch. However, this solution can be highly affected by poor performing tags especially in the static environment. The per tag cleaning algorithm adapts the window size for each individual tag and because window sizes for individual tags might be different, based on their detection rates, the decision on whether the tag is present or not is taken at different epochs. Therefore, due to different window sizes, the tags that are not ready for processing (i.e. the readings for all epochs in its window have not be accumulated) will delay the output. To avoid this limitation caused by low performing tags, the multi-tag cleaning algorithm uses the same smoothing window for all the tags together with a statistical estimation technique to accurately estimate the tags population count without cleaning on a per-tag basis.

6.6.1 Completeness Requirement for tag aggregates cleaning

As with individual tag observation, the smoothing window size plays a critical role in capturing the underlying tag's population aggregate. A large window ensures that the tags are observed and aggregated with high probability, but a small window is also desired to ensure that variability in the population count is adequately captured.

The multi-tag cleaning mechanism uses some of the concepts proposed in SMURF whereby the Horvitz-Thompson estimator [119] also known as the π-estimator together with unequal-probability random sampling model is used to approximate the population aggregates. As with the per-tag cleaning method, the multi-tag cleaning mechanism also views each epoch as an independent Bernoulli trial with probability p_i^{avg} for success. Where p_i^{avg} denotes the average empirical sampling probability for $tag\ i$ during window W derived from the readers tag list information using Equation (2).

Let S_W denote the sample of distinct tags read over the current smoothing window and let p^{avg} in Equation (8) denote the average per-epoch sampling probability over all observed tags.

$$p^{avg} = \frac{\sum_{i \in S_W} p_i^{avg}}{|S_W|} \tag{8}$$

Following the similar rationale used in the per-tag cleaning, to ensure that the underlying tag population is read with high probability ($\geq 1 - \delta$) we set the upper bound of the smoothing window size for multi-tag aggregate at:

$$w = \left\lceil \frac{ln\left(\frac{1}{\delta}\right)}{p^{avg}} \right\rceil \tag{9}$$

According to the binomial distribution, the probability of reading $tag\ i$ at least once during window $w = |W|$ is estimated as one minus probability of not detecting $tag\ i$ in all the trials:

$$\pi_i = 1 - \left(1 - p_i^{avg}\right)^w \tag{10}$$

Let $S_W \subseteq \{1, \ldots, N_W\}$ denote the subset of distinct observed (i.e. sampled) RFID tags over the window W and N_W denote the true tags count. The π-estimator for the population count based on the sample S_W is defined as:

$$\widehat{N_W} = \sum_{i \in S_W} \frac{1}{\pi_i}. \tag{11}$$

The π-estimator uses the sampling probability π_i to weigh the responses in estimating the population total. The poor performing tags with lower response probability are given higher weights while higher probability responses are given lower weights. The

π-estimator gives unbiased estimation of tag population $\widehat{N}_W$ with its estimated mean and variance given by Equation (12) and Equation (13) respectively.

$$E\left(\widehat{N}_W\right) = N_W \tag{12}$$

$$\hat{V}ar\left(\widehat{N}_W\right) = \sum_{i \in S_W} \frac{1-\pi_i}{\pi_i^2} \tag{13}$$

6.6.2 Adaptive window size adjustment for tag aggregates cleaning

The WSTD cleaning algorithm employs the random-sampling model and π-estimator concepts proposed in SMURF together with comparison of the two window sub-range estimated tag counts to dynamically adapt its smoothing window size. Transitions are detected as statistically significant changes in aggregate estimates over sub-ranges of its current smoothing window. The transition detection model used is the main difference between our multi-tag cleaning algorithm and the SMURF multi-tag cleaning algorithm.

Assume $W = (t-w, t]$ is current window, and let $W_1^{'} = (t-w, t-w/2]$ denote the first half of window W and $W_2^{'} = [t-w/2, t]$ denote the second half of the window W. Let $\widehat{N}_{W_1^{'}}$ and $\widehat{N}_{W_2^{'}}$ denote the π-estimators for tag population counts during $W_1^{'}$ and $W_2^{'}$ respectively. Note that the mid epoch (i.e. epoch at $t-w/2$) is inclusive in both ranges. The mid-point divides the window such that the numbers of epochs are equally spaced on either side of the window and this requires the use of an odd number window size. The transition is detected if there is significant change in tag counts between these two ranges.

In the SMURF multi-tag cleaning algorithm the transition is detected as a statistically significant transition in population count that has occurred in the second half of the window compared to the whole window population count by using CLT condition in Equation (14).

$$\left|\widehat{N}_W - \widehat{N}_{W_2^{'}}\right| > 2\left(\sqrt{Var\left(\widehat{N}_W\right)} + \sqrt{Var\left(\widehat{N}_{W_2^{'}}\right)}\right) \tag{14}$$

However, in our model the transition is detected as a significant change in the population count by comparing the count estimates in the first half and the second half of the window by using Equation (15).

$$\left|\widehat{N}_{W_1'} - \widehat{N}_{W_2'}\right| > 2\left(\sqrt{Var\left(\widehat{N}_{W_1'}\right)} + \sqrt{Var\left(\widehat{N}_{W_2'}\right)}\right) \quad (15)$$

Our experimental results verified that using the comparison of the sub-range population count estimates to detect population count variation within the window, gives a more accurate transition detection technique than comparison between full window count and the sub-range count estimates used by SMURF. The SMURF detection condition detects any significant variation within the window. However, for a transition detection mechanism we are more interested in detecting significant changes on the edge of the window, which signals that the tag is either entering or leaving the detection range and respond accordingly.

By comparing the population count of the two window sub ranges, it is possible to determine when the tag is exiting and entering the detection range, eliminating the need to use a mobile detection algorithm as proposed by SMURF. In the environment where tags are mobile, there are two scenarios; one is tags exiting the detection range, and the second is tags entering the detection range.

Simple rules are used to detect when these transitions occur by comparing the estimated tag counts in the two window sub-ranges.

Rule 1:

The tags are said to be exiting the detection range if the transition is detected according to Equation (15) and there is more estimated tag counts in the first half of the window than in the second half of the window (i.e. $\widehat{N}_{W_1'} > \widehat{N}_{W_2'}$).

$$exitTransition \leftarrow transitionDetected \wedge \widehat{N}_{W_1'} > \widehat{N}_{W_2'}$$

In this case, the window size is reduced multiplicatively (i.e. divided in half) to circumvent false positive readings.

Rule2:

The tag is said to be entering the detection region if transition is detected according to Equation (15) and there is more estimated tag count in the second half of the window than in the first half of the window (i.e. $\widehat{N}_{W_2'} > \widehat{N}_{W_1'}$)

$$enterTransition \leftarrow transitionDetected \wedge \widehat{N}_{W_2'} > \widehat{N}_{W_1'}$$

In this case, if the required window size is greater than twice the current window size, the window size is increased multiplicatively (i.e. doubled) if not, the window size is additively increased by two epochs. This is because as the tag enters the detection range it is assumed to be on the far end of a reader's detection ranges, i.e. long distance from the reader's antenna. Increasing the window size gives more opportunity even for the weak performing tags to de detected.

When the tags are leaving and entering the detection range, false positive readings will be produced regardless of the window size because the readings are interpolated throughout the window. Bigger windows are prone to this problem. To reduce false positive readings under these scenarios, we made two estimation assumptions. These approximation assumptions are used to detect when the tag(s) completely exit(s) the detection range and when the tag(s) just entered the detection region as illustrated in Figure 6-12.

Rule 3:

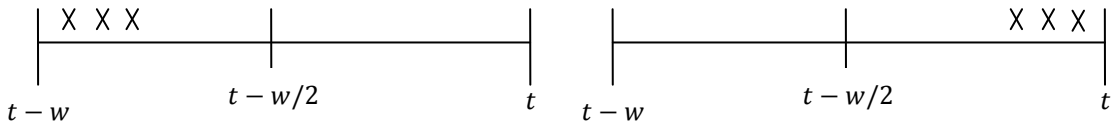
The tag(s) are said to have exited the reader's detection range if the overall window tag count is not zero, but the second half of the tag population count is zero (i.e. $\widehat{N}_W > 0 \ \wedge \ \widehat{N}_{W_2'} = 0$). This means that there was no tag observed in the second half of the window $(t - w/2, t)$.

$$exit \leftarrow \widehat{N}_W > 0 \ \wedge \widehat{N}_{W_2'} == 0$$

Similarly, the tag(s) are said to have just entered the reader's detection range if the overall window tag count is not zero, but the first half of tag population count is zero (i.e. $\widehat{N}_W > 0 \ \wedge \ \widehat{N}_{W_1'} = 0$). This means that there was no tag observed in the first half of the window $(t - w, t - w/2)$.

$$enter \leftarrow \widehat{N}_W > 0 \ \wedge \widehat{N}_{W_1'} == 0$$

Considering that the cleaning window size slides by the midpoint, we assume that the observed tags under these scenarios are more likely to be a false positive readings caused by a bigger window size. Therefore, tag(s) observed in these scenarios are dropped and the window size is reduced for an exiting scenario and increased appropriately for an entering scenario.



(a) Tags exiting (b) Tags entering

Figure 6-12: Illustration of mobile tag window sub-range as tags enters and exits the detection range

By taking advantage of the π-estimator, which scales-up the reading in the window to estimate the underlying tag population we can reduce the window sizes to enhance transition detection, hence the minimum window size can be reduced to 1 epoch. We introduce another estimation condition, which we call *strong region detection*. The aim of *strong region detection* is to detect when the tags within the window are observed with high probability of detection and when there is no significant variation in tag population within the two window sub-ranges.

Let S_i be a binomial sample of epochs in the current window W in which a single tag is observed and p_i^{avg} be the average read rate as defined in the per-tag cleaning approach. Let S_W denote the sample of distinct tags read over the current smoothing window and p^{avg} denote the average sampling probability over all observed tags given be Equation (8), and S^{avg} given by Equation (16) denote the average sample of epochs in the window in which the tags where observed.

$$S^{avg} = \frac{\sum_{i \in S_W} S_i}{|S_W|} \tag{16}$$

Rule 4:

The tags are then said to be observed in the strong detection region if the following condition holds:

$$\left(p^{avg} > \frac{S^{avg}}{W}\right) \wedge \left(\left|\widehat{N}_{W_1'} - \widehat{N}_{W_2'}\right| < \left\lceil 0.05 \cdot min\left(\widehat{N}_{W_1'}, \widehat{N}_{W_2'}\right)\right\rceil\right) \tag{17}$$

The second portion of the logical condition tests if the two window sub-range estimates have a relatively small difference of less than 5% of the lowest estimated tag counts. If the condition holds, the window size is reduced by two epochs.

Figure 6-13 shows a pseudo-code description of the adaptive multi-tag cleaning algorithm. All the tags are cleaned using the same window. Similar to per-tag cleaning,

the smoothing-window size is systematically adjusted based on the analysis of the observed tags binomial-sampling data and the transition is detected by comparing the window sub-range estimated population counts.

Input:: T = set of all observed tag IDs
Input: T = set of all observed tag IDs
δ = required completeness confidence
Output: t = tags count
Initialize: $w \leftarrow 1$
while($getNextEpoch$) **do**
for (i in T)
$processWindow(W) \rightarrow p_{i,t},\ |S_i|,\ p_i^{avg},\ p^{avg},\ S^{avg},\ \widehat{N_W}, \widehat{N_{W_1'}}, \widehat{N_{W_2'}}$
end for
$W^* \leftarrow requiredWindowSize(P^{avg}, \delta)$
$transition \leftarrow \left|\widehat{N_{W_1'}} - \widehat{N_{W_2'}}\right| > 2\left(\sqrt{Var\left(\widehat{N_{W_1'}}\right)} + \sqrt{Var\left(\widehat{N_{W_2'}}\right)}\right)$
$exitTransition \leftarrow transitionTest \wedge \widehat{N_{W_1'}} > \widehat{N_{W_2'}}$
$enterTransition \leftarrow transitionTest \wedge \widehat{N_{W_2'}} > \widehat{N_{W_1'}}$
$exit \leftarrow \widehat{N_W} > 0 \wedge \widehat{N_{W_2'}} == 0$
$enter \leftarrow \widehat{N_W} > 0 \wedge \widehat{N_{W_1'}} == 0$
$strongDetection \leftarrow \left(\ p^{avg} > (S^{avg}/W)\right) \wedge \ \left(\left|\widehat{N_{W_1'}} - \widehat{N_{W_2'}}\right| < \left\lceil 0.05 \cdot min\left(\widehat{N_{W_1'}}, \widehat{N_{W_2'}}\right)\right\rceil\right)$
if ($exit \vee exitTransition \vee strongDetection$)
if ($exit$)
t = 0
else
t = $\widehat{N_W}$
end if
if ($strongDetection$)
$W \leftarrow$ max (min$(W-2, W^*)$, 1)
Else
$W \leftarrow$ max (min $\left(\frac{W}{2}, W^*\right)$, 1)
end if
else if $((w_i^* > w_i) \wedge (|S^{avg}| < w \cdot p^{avg}))$
if ($enter$)
t = 0
else
t = $\widehat{N_W}$
end if
if $(W^* > 2 * W \wedge (enter \vee enterTransition))$
$W \leftarrow$ min $(W * 2,\ W^*)$
else
$W \leftarrow$ min $(W + 2,\ W^*)$
end if
else
t = $\widehat{N_W}$
$W \leftarrow$ min (W, W^*)
end if
output (t)
end while

Figure 6-13: WSTD-π Multi-tag cleaning algorithm

6.7 Summary

The unreliability of the data streams generated by RFID readers is among the primary factors limiting the potential widespread adoption of RFID technology. RFID data cleaning is therefore an essential task in the RFID middleware systems in order to reduce reading errors, and to allow these data streams to be used to make correct interpretations and analysis of the physical world they are representing.

This chapter opened with an investigation of the factors that affect RFID system performance and the need for low level RFID data to be cleaned before being used in any applications. Specifically, the focus was on RFID missing readings problem and the sliding-window based approaches which are used to address this problem. Two categories of sliding-window based cleaning methods were considered, namely fixed, static window cleaning methods and variable window cleaning methods.

The RFID data cleaning scheme called SMURF proposed by Jeffery *et al*. [10] has been extended, and a new scheme with a more efficient transition detection mechanism was developed. The transition detection mechanism referred to as the Window Sub-range Transition Detection (WSTD) uses the comparison of the two window sub-range observations or estimated tag counts to detect when transitions occur within the window. A distinction was also drawn between detecting an individual tag and detecting the number of tags which are available for application which are only interested knowing the number of available tags. Performance evaluation of the proposed WSTD cleaning scheme is next presented in chapter 7.

Chapter 7: Experimental Evaluation of WSTD cleaning Scheme

In this chapter the experimental evaluation of the proposed WSTD cleaning algorithms is presented. The data sets for the experiments were generated by a synthetic data generator that simulates the operation of RFID readers under a wide variety of conditions using MATLAB. The generator is composed of two components. The first component simulates the movement of tags and the second component simulates tag detection by an RFID reader.

7.1 Reader Detection Model

The reader detection model is based on the RFID reader detections regions. As was discussed in section 6.1, there are three distinct regions of operations of a passive RFID reader tag system: strong-in-field, weak-in-field and out-of-field as shown in Figure 6-2 and Figure 6-3. In the strong-in-field region, the tag responds to most of the read attempts from the reader. Thus, the response rate in the strong-in-field region varies between 77% and 100%. The tag performance then degrades gradually with increasing distance in the weak-in-field region. In the out-of-field region, the response rate goes down to 0%. Based on these observations, a simplified reader detection model shown in Figure 7-1 was derived.

The model uses the following parameters to capture a wide range of reader behaviour in different conditions:

- $MaxDetectionRange$ – is the distance in meters from the reader's antenna to the edge of the reader's detection range.
- $StrongPercentage$ – is the percentage of the reader's overall detection range that is in the strong detection region. According to the reader performance analysis in section 6.1, this percentage decreases with the increase in the environment noise or the presence or close proximity of an RF unfriendly material such metal, metalized/foil-lined packaging, carbon and graphite-impregnated plastics and water. Therefore, varying this percentage simulates varying the environment conditions in which the system operates.

- $MaxReadRate$ – is the read rate (i.e. the probability of detection) of a tag within the strong detection region. The read rate in the weak detection region drops linearly to the end of the reader's detection range.

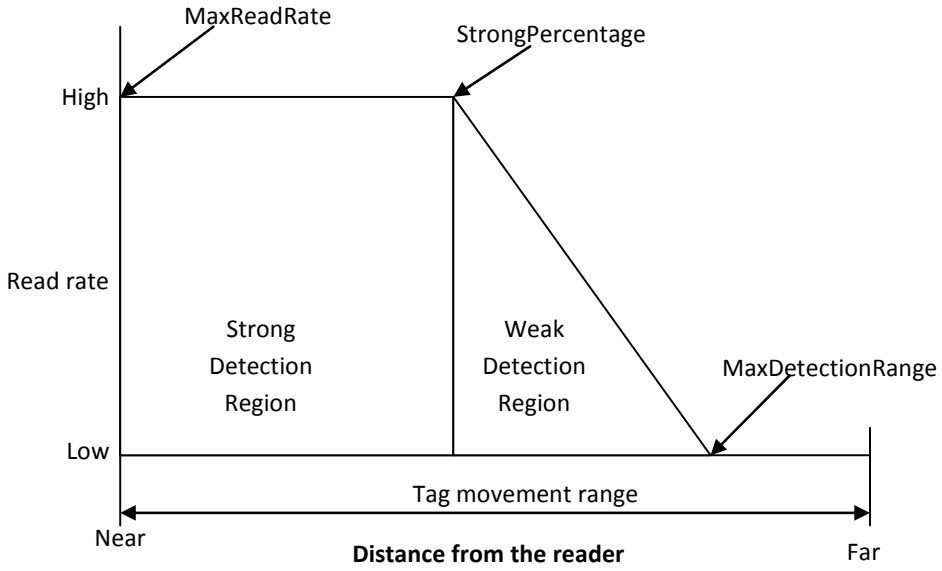


Figure 7-1: Reader detection model for RFID data generator

7.1.1 Tag Movement Behaviours

The tag movement component simulates the tag's movements. We looked at three tag movement behaviours. The first behaviour is that of static tag(s). This is simulated by randomly placing tags uniformly within the reader's detection region. This simulates static tagged items which are constantly monitored by the RFID system (i.e. both tags and readers are static). The second behaviour is that of tags moving with the same velocity. This simulates grouped tags, such as tagged items on a trolley or conveyor belt. The third behaviour is that of tags moving with different velocity. This behaviour simulates tracking environments, such as a digital work place where each tag displays independent random behaviour.

Tag movements are simulated by moving the tags in and out of the reader detection range between 0 and 6 m. We set the maximum detection range to be 4.6 m (~15 feet) between 4.6 and 6 m the tag is out of the detection range. To simulate movement with different velocities, each tag selects a random initial velocity between 0 and 0.9 m/epoch. After

every 100 epochs on average (i.e. 80 – 120 epochs), each tag switches from moving state to resting state and vice versa. Figure 7-2 illustrate the constant velocity movement trace for 400 epochs at a speed of 0.6 m/epoch, and Figure 7-3 illustrates a variable velocity movement trace for 400 epochs for a single tag.

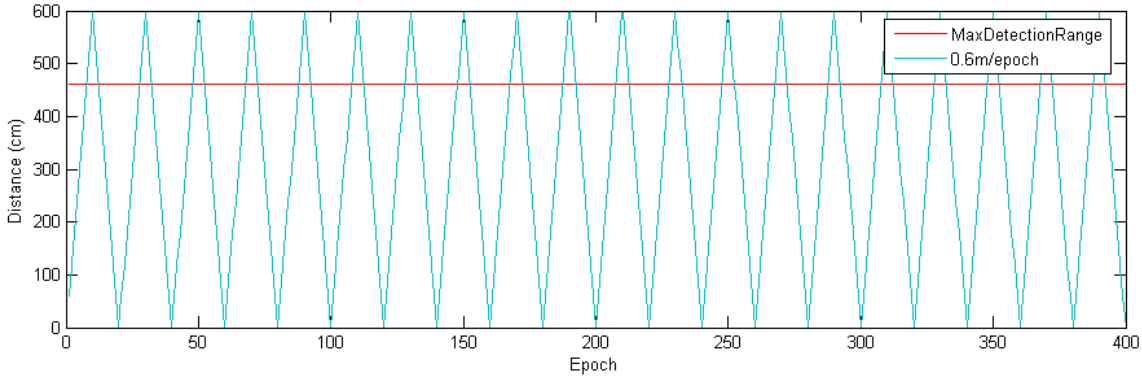


Figure 7-2: Tag Movement behaviour with constant velocity of 0.6 m/epoch

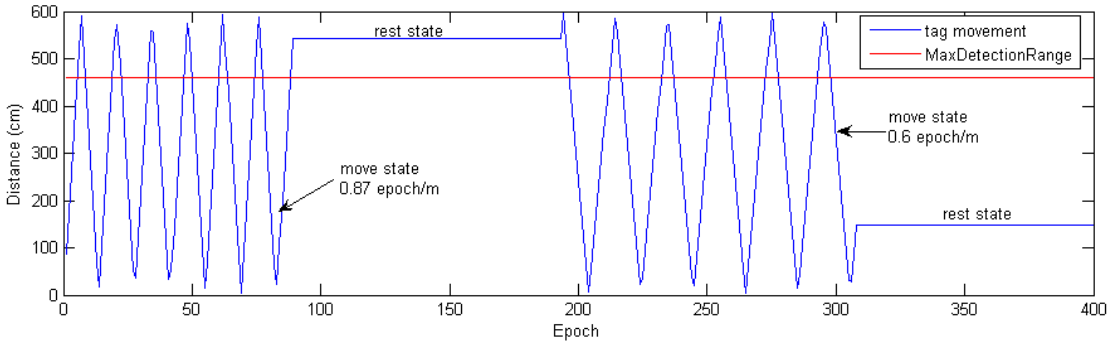


Figure 7-3: Tag movement behaviour with variable speed

7.1.2 Tag Detection

The tag detection component uses the reader detection model (Figure 7-1) to determine which tags are detected based on the read rate (probability of detection $p_{i,t}$) of each tag's location relative to the reader. The corresponding mathematical model of the reader detection is given in Equation (18).

$$p_{i,t}(x) = \begin{cases} MaxReadRate, & x < StrongPercentage \\ \frac{MaxReadRate(x - MaxDetectionRange)}{StrongPercentage - MaxDetectionRange}, & StrongPercentage \leq x \leq MaxDetectionRange \\ 0, & x > MaxDetectionRange \end{cases} \quad (18)$$

The output of the tag movement generator is then sent to the tag detection component for detection. At each epoch the detection component produces a set of readings containing an epoch number, tag ID, and the tag $p_{i,t}$ (read rate at which the reader read the tag). The set of all tags within the reader's detection region from the data movement generator is also produced and saved as reference values for comparing with the output of each cleaning mechanism.

The data produced by the tag detection component was then cleaned using the developed cleaning method discussed in Chapter 6, which is denoted as WSTD as well as the implementation of SMURF, and various sized static smoothing-window methods. We denote each fixed window method as Fxd$x$, where x is the size of static cleaning windows in epochs. Table 7.1 summarizes the experimental parameters used to produce our synthetic RFID data traces.

Table 7.1: Experimental Parameters

Parameter	**Value**
$MaxDetectionRange$	4.6 m
$MaxReadRate$	0.8
$StrongPercentage$	Varied
$NumTags$ (number of tags)	25(per-tag), 100(multi-tag)
$velocity$	varied
$NumEpochs$ (number of epochs)	2000 epochs

7.2 Individual Tag Cleaning

The cleaning techniques which report individual tag ID readings are examined first. The performances of different cleaning schemes are compared as the tag movement and reliability of the environment are changed. Tag movement is changed from either static motionless, constant velocity or variable velocity. The reliability of the environment is changed by changing the $StrongPercentage$ parameter.

The evaluation metric for individual tag cleaning is the average number of errors per epoch. An error is a reading that indicates a tag exists when it does not exist (a false

positive), or a lack of reading where the tag exists (false negative). The average number of errors per epoch is calculated using Equation (19).

$$ErrorsPerEpoch = \frac{\sum_{i=1}^{NumEpochs}(FalsePositive_i + FalseNegative_i)}{NumEpochs} \quad (19)$$

7.2.1 Experiment 1: Environment Reliability with randomly moving tags

In this experiment we determine how each technique reacts to different levels of environment unreliability with the randomly moving tags. Each tag moves at its own random velocity of between 0 to 90 cm/epoch and after every 100 epochs on average the tag changes its state from moving to rest state and vice versa as shown in Figure 7-3. The strong-in-field region percentage is varied between 0 and 100%. The lower $StrongPercentage$ corresponds to an unreliable environment and higher values of $StrongPercentage$ corresponds to a more controlled environment. At each $StrongPercentage$ we measure the average number of errors produced by each scheme.

Figure 7-4 shows the result of this experiment, the poor performing traces 'raw' and '*Fxd2*' are truncated to a enable clear view of other traces.

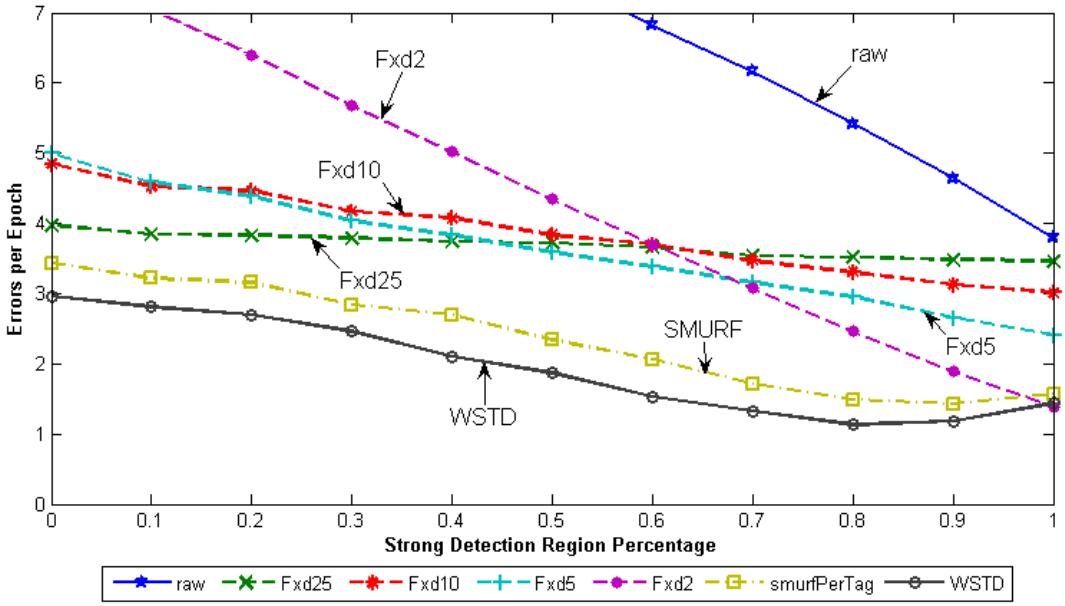


Figure 7-4: Average errors per epoch as strong-in-field region percentage is varied

In general, all schemes have the same pattern; that is, their performance improves as the environment noise decreases and the reader produces more reliable data. Looking at the fixed window schemes, there is no single fixed window scheme which performs consistently better than other schemes in all the environments. In the noisy environment large windows ('*Fxd25*' and '*Fxd10*') perform well than small windows ('*Fxd5*' and '*Fxd2*') while in controlled environment ($StrongPercentage > 60\%$) small windows perform better than large windows. This can be explained by looking at the positive and negative error contributions in these schemes as shown in Figure 7-5. This Figure verifies that large window schemes '*Fxd25*' and '*Fxd10*' errors are highly contributed by the false positive errors due to interpolations of readings within the large window. In the small windows the errors are highly contributed by the false negative readings due to their inability to compensate for missed readings. As the strong-in-field percentage increases and the reader produce more reliable data, the false negative errors decreases while the false positive errors increase.

On the other hand, variable window schemes SMURF and WSTD perform consistently well across the entire range of environments. Its performance efficiency is attributed to its per tag cleaning concept whereby each tag's smoothing window is adjusted independently based on its individual random behaviour. The WSTD scheme performs better than SMURF producing an improvement of approximately 17% less overall error in comparison to that produced by SMURF. This performance is attributed to its improved transition detection mechanism as shown in Figure 7-6. Comparing the two variable cleaning-window sizes, WSTD uses a smaller window size in comparison to that used by SMURF, as shown in Figure 7-7. Because of its small window size, WSTD is more efficient in detecting transition than SMURF; however, it also produces slightly more negative errors than SMURF as shown in Figure 7-8. The increase in false negative errors in the noisy environment by WSTD can be associated with the premature transition detection by *rule2* of the WSTD algorithm. As the noise decreases, their performances in compensating for missed readings become competitive and their difference decrease.

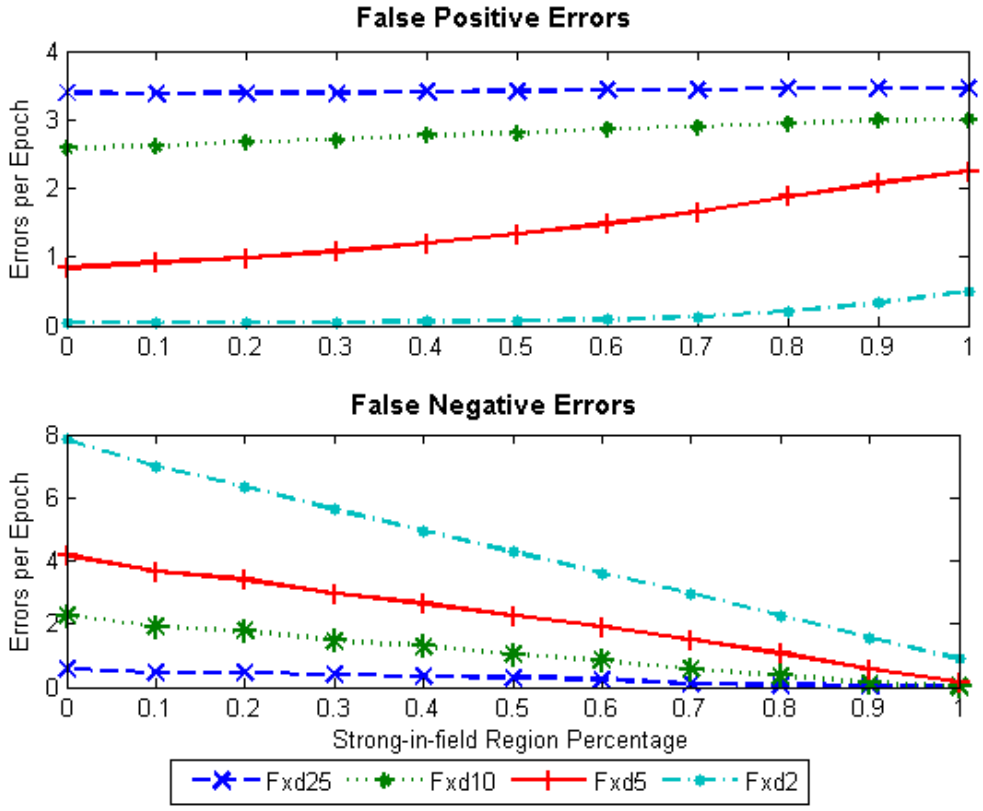


Figure 7-5: Fixed window schemes false positive and false negative error contributions as the environment noise is varied

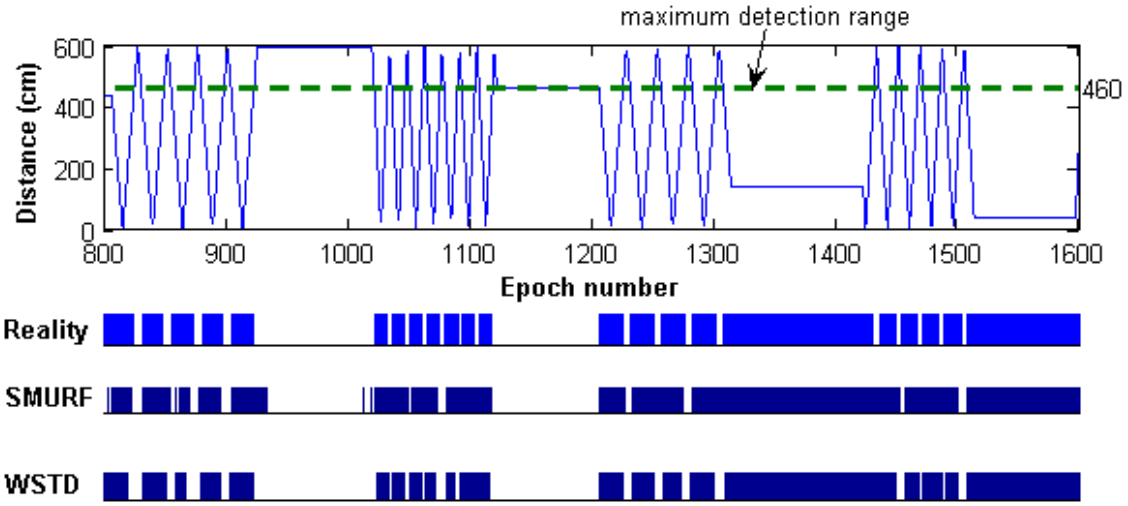


Figure 7-6: Comparison of WSTD and SMURF schemes transition detection mechanisms as a tag moves at random velocity

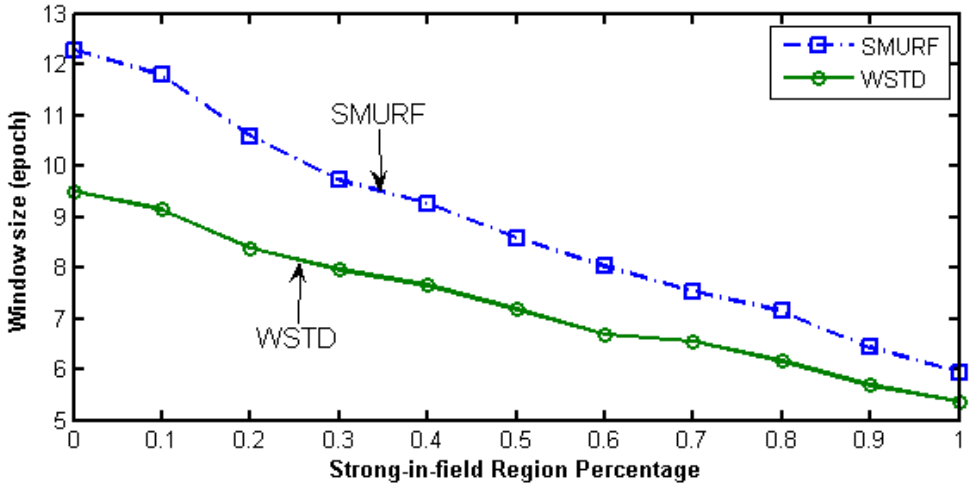


Figure 7-7: Comparison of WSTD and SMURF schemes cleaning window sizes as the environmental noise is varied

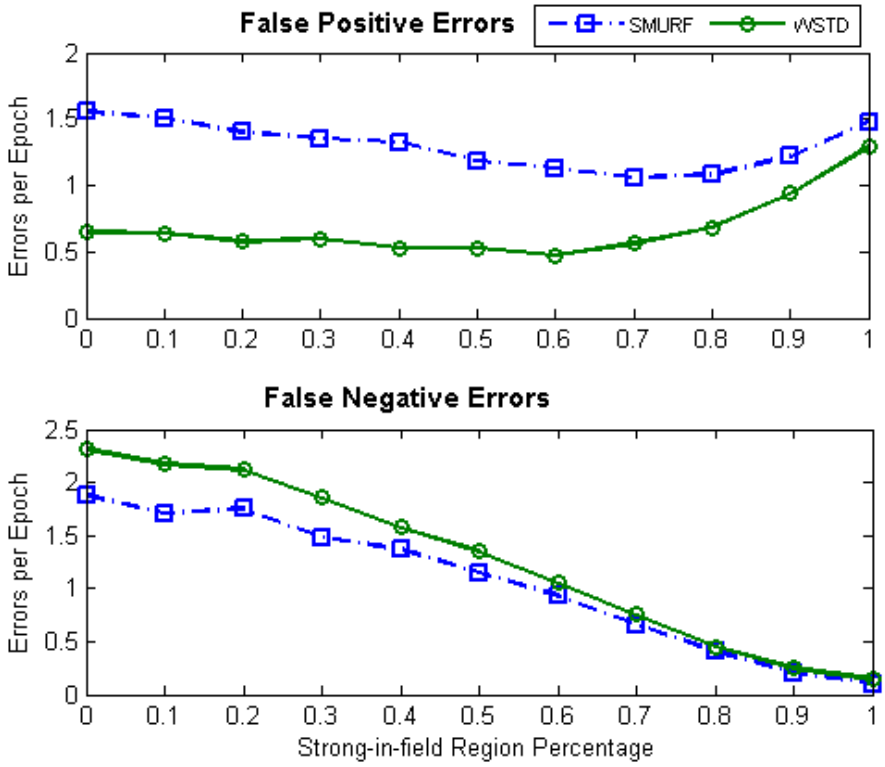


Figure 7-8: WSTD and SMURF schemes false positive and false negative error contributions as the environmental noise is varied

Figure 7-9 shows a comparison of all cleaning schemes transition detection capabilities as the tags move in and out of the detection range at variable speeds. The focus is on the readings produced from a single tag in a noisy environment (i.e. $StrongPercentage = 0$) over 350 epochs. The readings produced by the tag in this scenario are particularly

challenging to clean as data are highly unreliable and the tag intermittently moves at high speed. The cleaning scheme must be able to differentiate between periods of dropped readings and periods when tags are transiently absent, and act accordingly.

The top subsection of Figure 7-9 shows the tag movement trace. The tag moves in and out of detection range with a speed of 0.47 m/epoch; it stops at point A 143 cm from the reader for a period of time (108 epochs) and resumes movement at point B (1422 epoch). From point B the tag moves with the speed of 0.67 m/epoch until point C (epoch 1512) and stops again 43cm from the reader for 82 epochs before resuming the movement. The following subsection of the Figure shows the output produced by the reader (i.e. raw), followed the reality based on the tag movement. The following subsections are the output generated by different static window cleaning schemes followed by a variable SMURF and WSTD per tag cleaning schemes. The last subsection shows the WSTD per tag window sizes over the course of the trace.

The main observation from this trace is that large windows are completely unable to detect transition. While small windows are better at detecting transition, they are unable to compensate for false negative readings. This trend can also be observed on the last subsection of the Figure which shows the WSTD window sizes over the course of the trace. When there is a transition with a bigger WSTD window size, WSTD is unable to detect transition and when the WSTD window is small and the tag is missing, WSTD is also unable to compensate for it. '*Fxd5'* fixed window scheme perform competitively well with variable window schemes. Comparing the variable window schemes, WSTD performs better than SMURF.

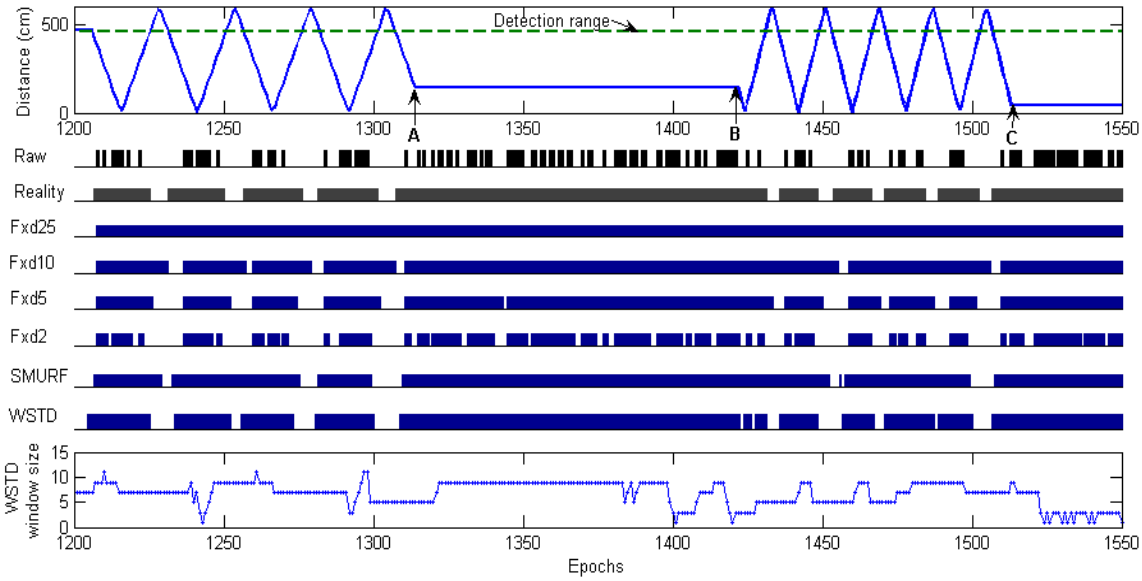


Figure 7-9: Cleaning schemes cleaning the readings from a single tag moving with different velocities over 350 epochs

7.2.2 Experiment 2: Effect of tag speed

The effectiveness of the individual tag cleaning schemes is then compared as the tag velocity is varied. The $StrongPercentage$ parameter is fixed at 70% to represent the controlled environment and the tags are moved in and out of the detection range at the same constant velocity. The velocity is varied from 0 to 90 cm/epoch and the average numbers of errors produced by each scheme were measured. Figure 7-10 shows the result of this experiment and Figure 7-11 and Figure 7-12 shows the positive and negative errors of fixed and variable window schemes respectively as the tag velocities are varied.

In the mobile environment, the larger the window size, the higher the number of false positive errors. At one particular fixed window size, the false positive errors increase with the increase in tag speed until it reaches a saturation speed beyond which no transition is detected. Beyond the saturation speed in the worst case scenario, the scheme continuously reports all tags as being present with no false negatives. The saturation speed increases with the decrease in window size; that is, the higher the window size the lower the saturation speed e.g. in Figure 7-10, schemes: '*Fxd25*', '*Fxd10*' and '*Fxd5*' have saturation speeds of 0.2, 0.4 and 0.8 m/epoch respectively. When the number of positive errors is less than the number of negative errors, the fixed window schemes perform

competitively well compared with the variable window schemes (compare Figure 7-10 and Figure 7-11).

The small window scheme ('*Fxd2*') has a relatively consistent performance, irrespective of the change in velocity. This is because small windows are able to detect transitions caused by varying velocities, although they are unable to compensate for the missing tags. On the other hand, variable window schemes SMURF and WSTD perform consistently well and also outperform the '*Fxd2*' scheme. This is because in addition to being able to detect transitions, they are also able to compensate for missed readings. The WSTD scheme performs better than SMURF, producing an improvement of approximately 30% less overall errors in comparison to that produced by SMURF. This performance improvement is attributed to its improved transition detection, as shown in Figure 7-13. WSTD uses smaller window sizes than SMURF (see Figure 7-14); as a result it produces fewer positive errors but slightly higher false negative errors in comparison to that produced by SMURF, as shown in Figure 7-12. In the controlled environment, their performance in compensating for missed readings become competitive and their differences decrease (see Figure 7-12).

The reduction in environmental noise has the effect of reducing false negative readings and, hence, has a more positive effect on smaller cleaning window size schemes compared to big window size schemes.

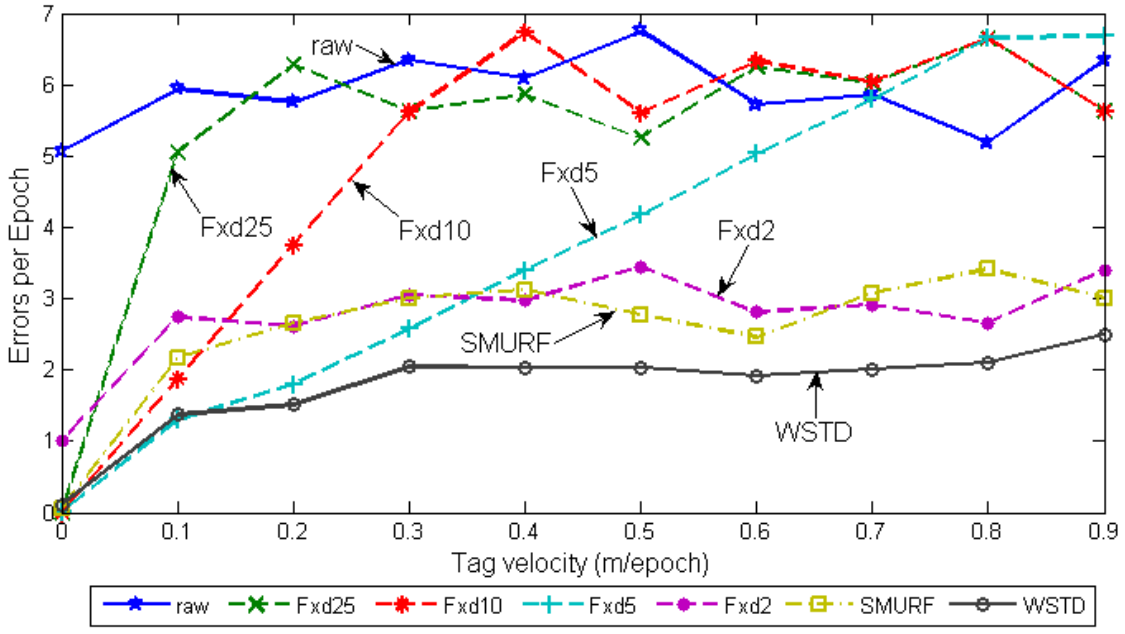


Figure 7-10: Average errors per epoch as tag velocity varies

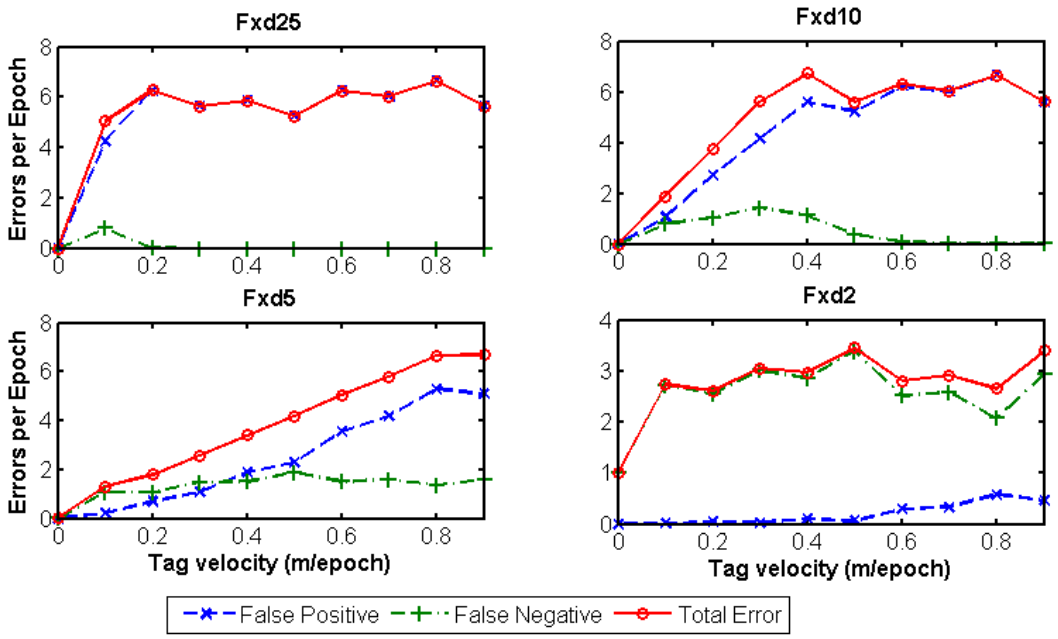


Figure 7-11: Fixed window schemes' false positive and false negative error contributions as the tag velocity is varied

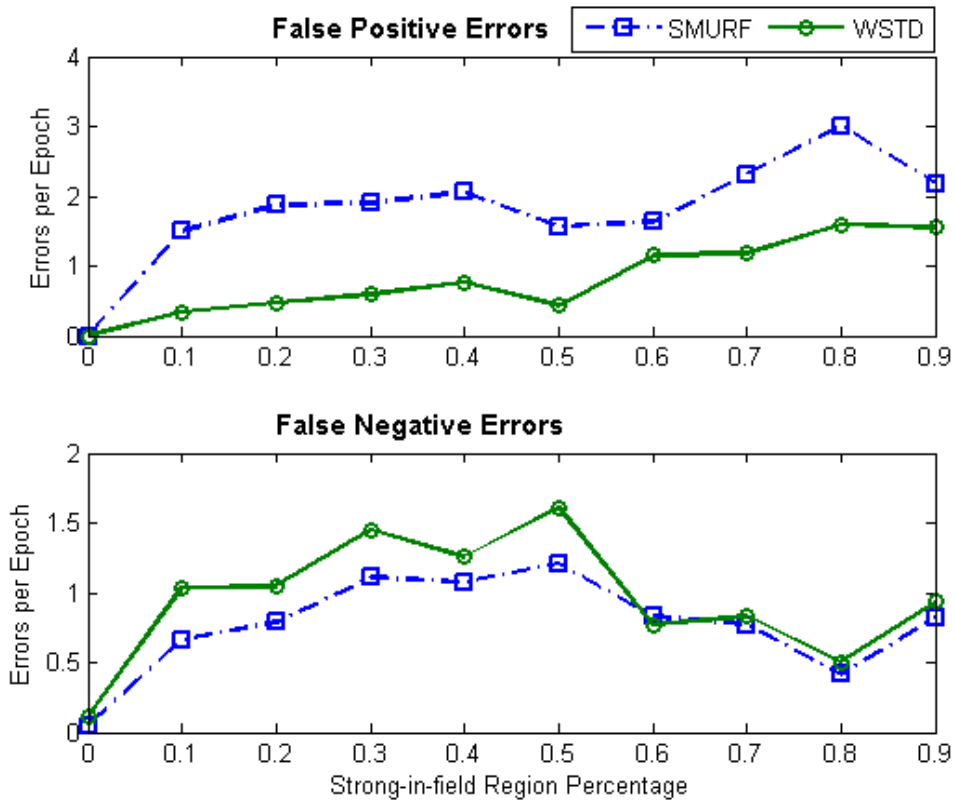


Figure 7-12: WSTD and SMURF schemes false positive and false negative error contributions as the tag velocity is varied

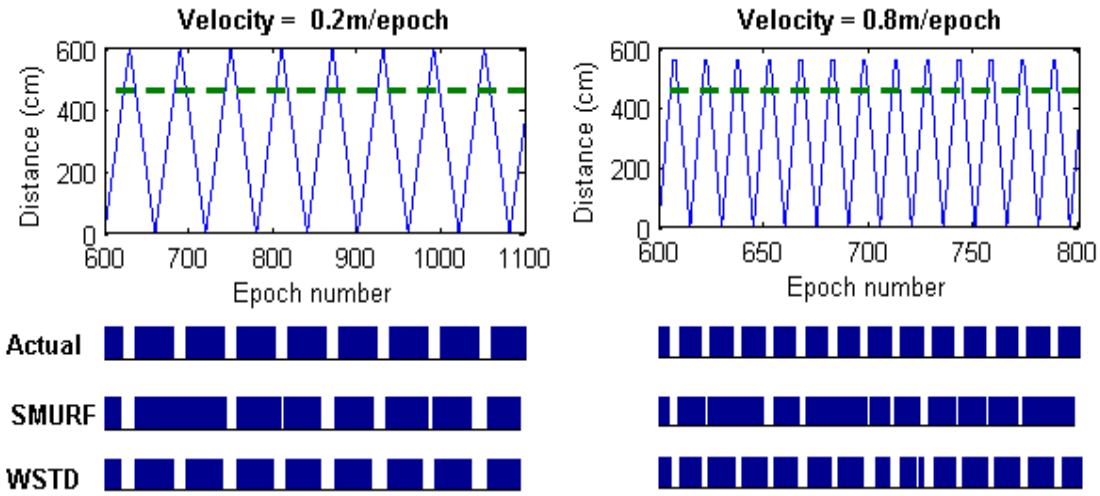


Figure 7-13: Comparison of WSTD and SMURF schemes' transition detection mechanisms as a tag moves at constant velocity

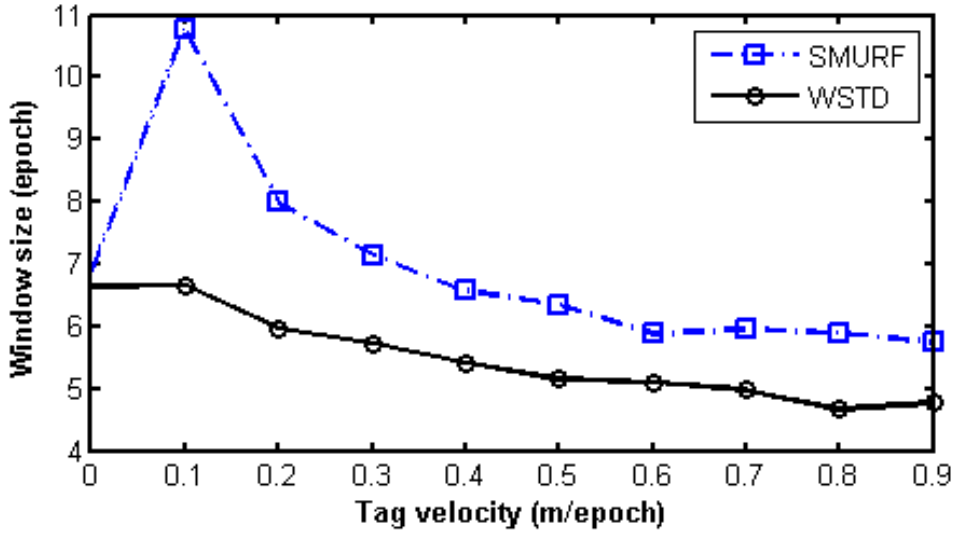


Figure 7-14: Comparison of WSTD and SMURF schemes' cleaning window sizes as the velocity is varied

What was also evaluated is how cleaning schemes perform when tag velocities are varied in different environments. Figure 7-15 shows the result of the two extreme environments. The noisy environment $StrongReg_0$ (with $StrongPercentage$ parameter set to 0%) and the controlled environment $StrongReg_1$ (with $StrongPercentage$ parameter set to 100%). Figure 7-16 shows the error contribution of the fixed window schemes and Figure 7-17 shows the error contributions of the variable window schemes in these two extreme environments.

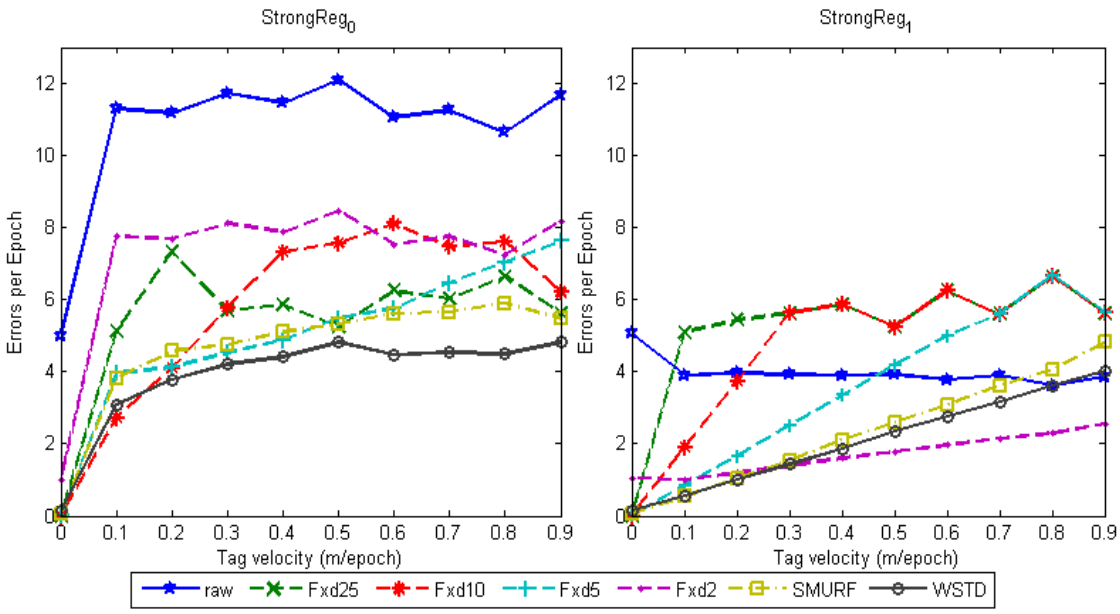


Figure 7-15: Average errors per epoch as tag velocities are varied from 10 to 90 cm/epoch in different environments

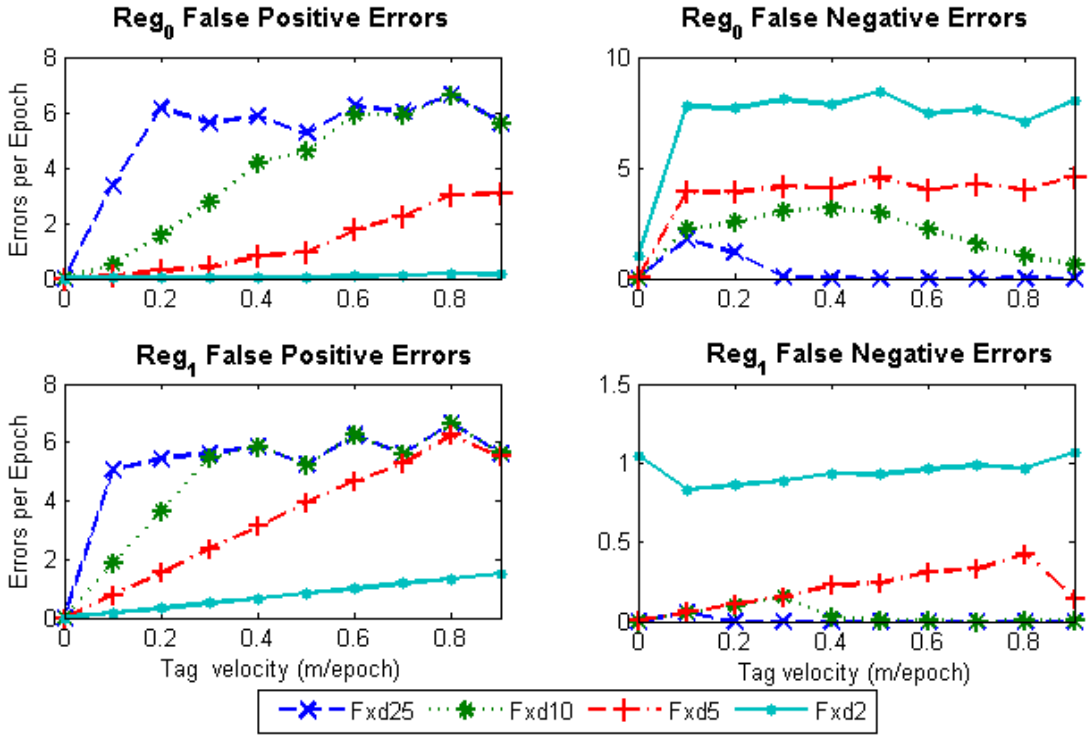


Figure 7-16: Fixed window schemes error contributions as tag velocities are varied in different environmental conditions

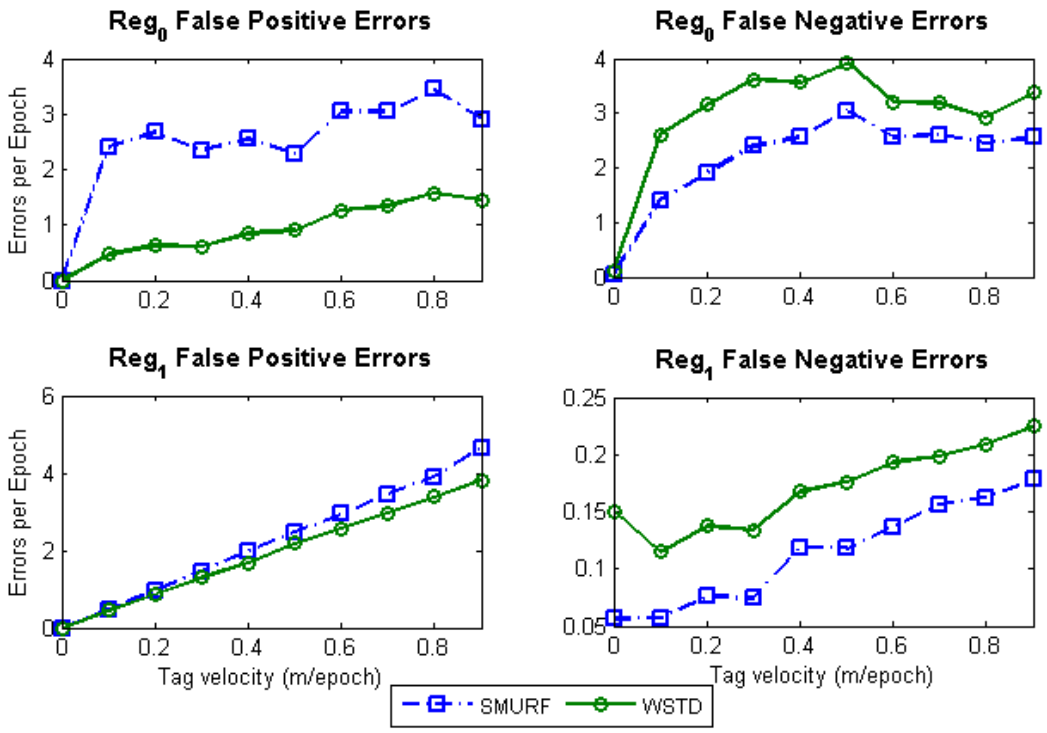


Figure 7-17: Variable window schemes error contributions as tag velocities are varied in different environmental conditions

These results show that reducing the environmental noise has the effect of reducing the false negative errors as the reader produces more reliable results. Likewise, as the data becomes more reliable, the false positive errors introduced by the cleaning schemes also increase. The results also show that the rate of increase of the positive errors depends on the window size and the speed of the tag. The false positive errors introduced increase with the increase in window size as well as with the increase in tag velocity. At one particular fixed window size, the false positive errors increase linearly with the increase in tag speed until it reaches a saturation speed beyond which no transition is detected. The more reliable the data, the higher the rate of increase of positive errors as the tag speed increases and the lower the saturation speed of the window.

7.2.3 Experiment 3: Environment Reliability with Static Tags

Also evaluated was the performance of different cleaning schemes in the environment where tags are stationary. To simulate this scenario, 25 tags were randomly distribute uniformly within the detection range and the $StrongPercentage$ parameter varied and the average errors produced by each scheme measured. Figure 7-18 shows 25 tags

randomly distributed within the reader's detection range (0 to 4.6 m). Figure 7-19 shows the result of this experiment and Figure 7-20 shows the average cleaning-window sizes for the variable schemes SMURF and WSTD as the environmental noise is varied.

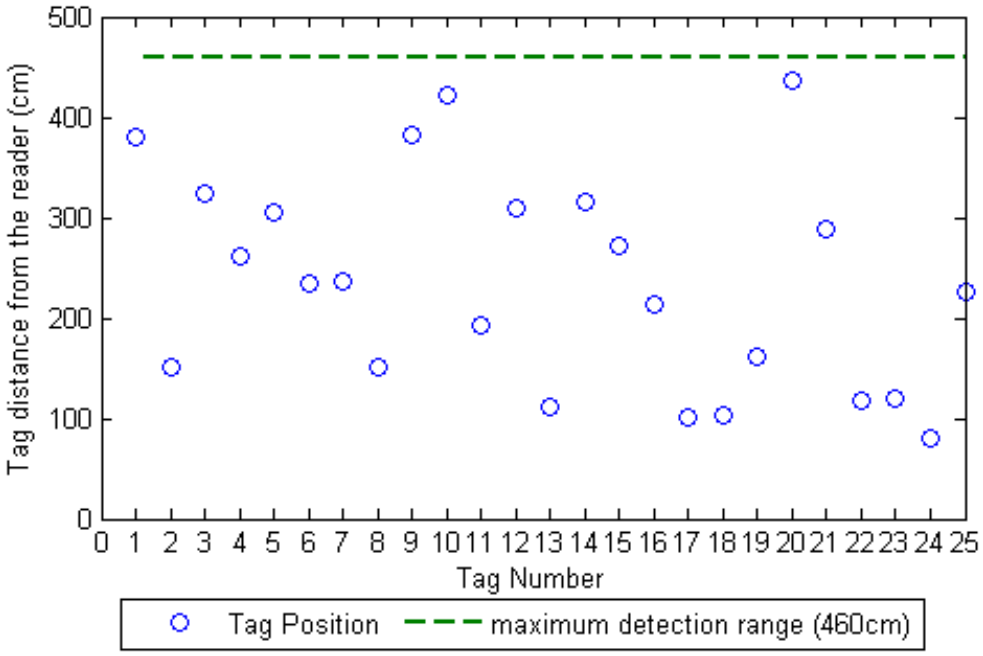


Figure 7-18: 25 Static tags randomly distributed within the readers detection range

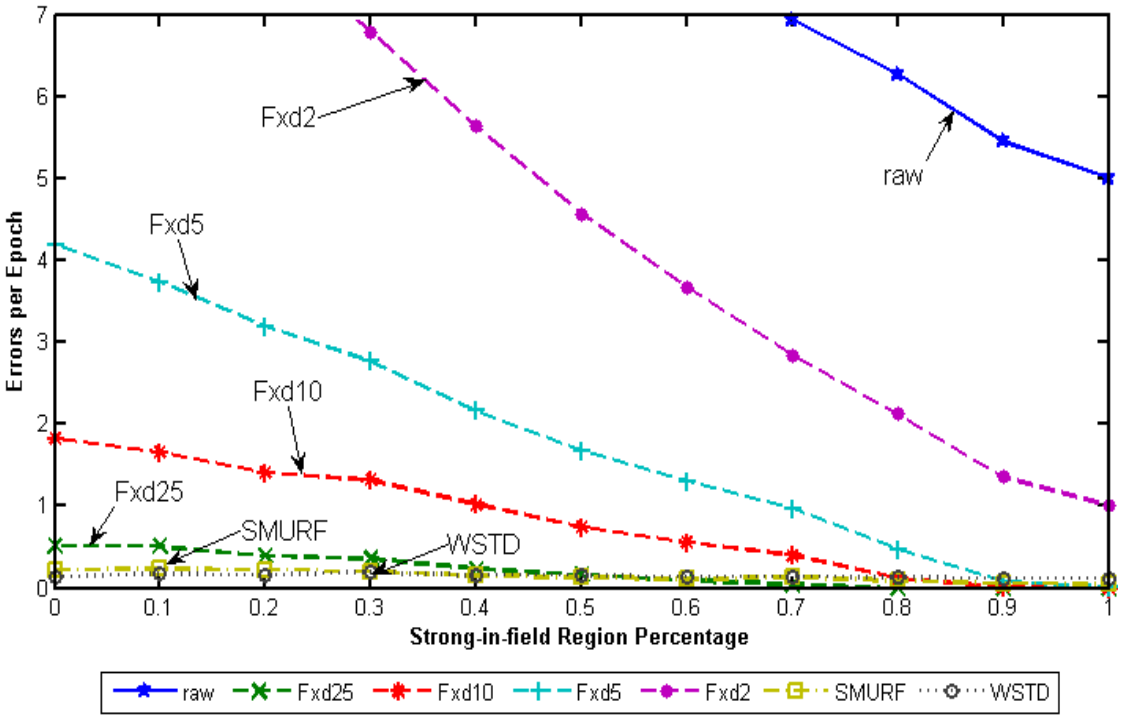


Figure 7-19: Average errors per epoch as strong-in-field region percentage is varied in the static tag environment

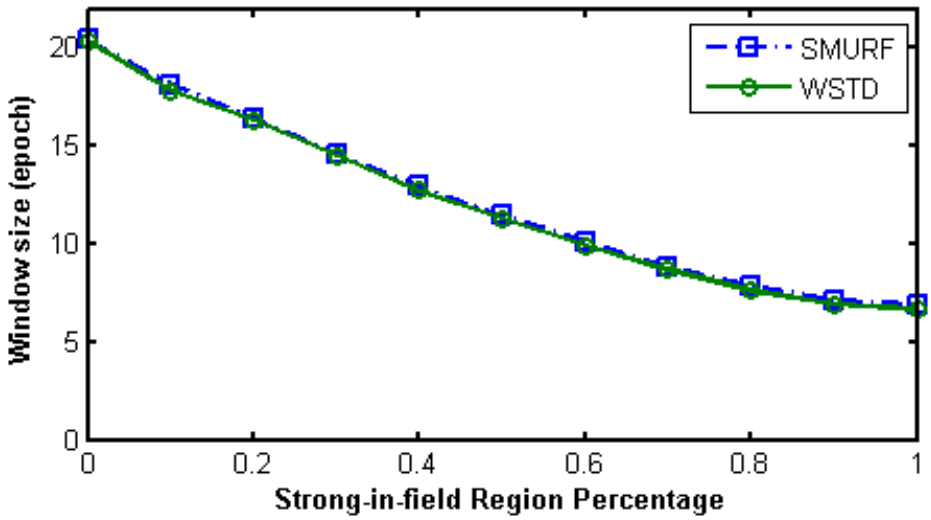


Figure 7-20: Comparison of WSTD and SMURF schemes' cleaning-window sizes as the environmental noise is varied

All static window schemes have the same pattern, which is that the errors decrease as the strong-in-field percentage increases. Since all the tags were within the detection region and at the same position throughout the simulation time, all the errors are false negative readings. Large static windows perform better than small windows throughout the trace. Hence, in general, in the static environment bigger windows perform better than small windows. The variable window schemes SMURF and WSTD have a consistent performance irrespective of the environment changes.

In a noisy environment (with $StrongPercentage < 60\%$) the variable window schemes outperform all the static window schemes producing less than 1 tag out of 25 tags per epoch on average. In a controlled environment, large static windows perform competitively with variable window schemes compensating for all missing tags or producing very few false negative readings. In the static environment, variable window schemes WSTD and SMURF exhibit closely matching performances, which can be attributed to their use of similar cleaning window sizes as shown in Figure 7-20. From this observation we can conclude that the main difference between WSTD and SMURF is in the transition detection mechanism, and WSTD performs better than SMURF in the mobile environment.

In summary, across different speeds and environments, there is no single static window that works uniformly well. Smaller window schemes like '*Fxd2*' work very well at high speeds in the controlled environment, but their performances deteriorate in the static tag

environment, slower speeds and in a noisy environment due to their inability to compensate for missed readings. On other hand, large window schemes like '*Fxd25*' work very well in static tag environments but they perform poorly in mobile environments due to their inability to detect tag transitions. Hence, when dealing with static schemes, it is challenging to know the exact size of the optimum cleaning window. In addition, the RFID tag-reader system performance is very sensitive to the environment, and through the analysis of these few fixed window schemes we have shown that there is no one optimum window size which can work best in all the environments.

This calls for the need to use variable cleaning window schemes for low level cleaning of RFID tag data streams. It relieves the system user from the calibration work in trying to set the optimum fixed window size which might not be efficient in cases where small changes occur in the environment or the tags' speed change. We have proposed a variable sliding window scheme based on the concepts proposed in SMURF scheme with an improved transition detection mechanism. Our experiment result shows that the WSTD scheme provides 30% performance improvement over SMURF in mobile environments.

7.3 Analysis of WSTD per- tag variable window size

This section presents an analysis of how the WSTD scheme cleaning window size varies as it cleans the data in different conditions. Firstly, the window sizes as it cleans the static tags in the scenario depicted in Figure 7-18 are considered. Figure 7-20 shows that the overall average cleaning window size decreases linearly with the decrease in the environmental noise.

Secondly, how the size of the window is affected by the position of the tag from the reader is scrutinized. To achieve this, the window size of one tag as it gets to be cleaned over 2000 epochs is analysed. Three positions of the tag relative to the reader's antenna is analysed: a *tag number 20* on the Figure 7-18 located on the far edge of detection range at 436cm; a *tag number 16* located at middle of the detection range at about 213 cm; and *tag number 24* located closer to the reader at about 80 cm. The results of this analysis are shown in Figure 7-21, Figure 7-22 and Figure 7-23 respectively. In the figures, $StrongReg_0$ refers to the noisy environment and $StrongReg_1$ refers to the controlled environment.

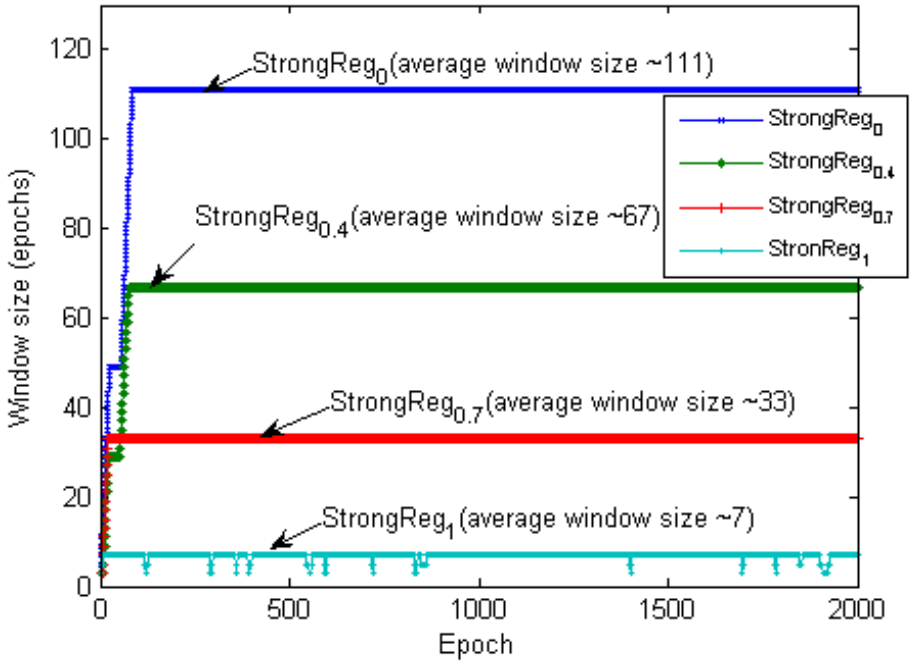


Figure 7-21: WSTD per tag window sizes as a single static tag placed on the far edge of detection range is cleaned in different environmental conditions

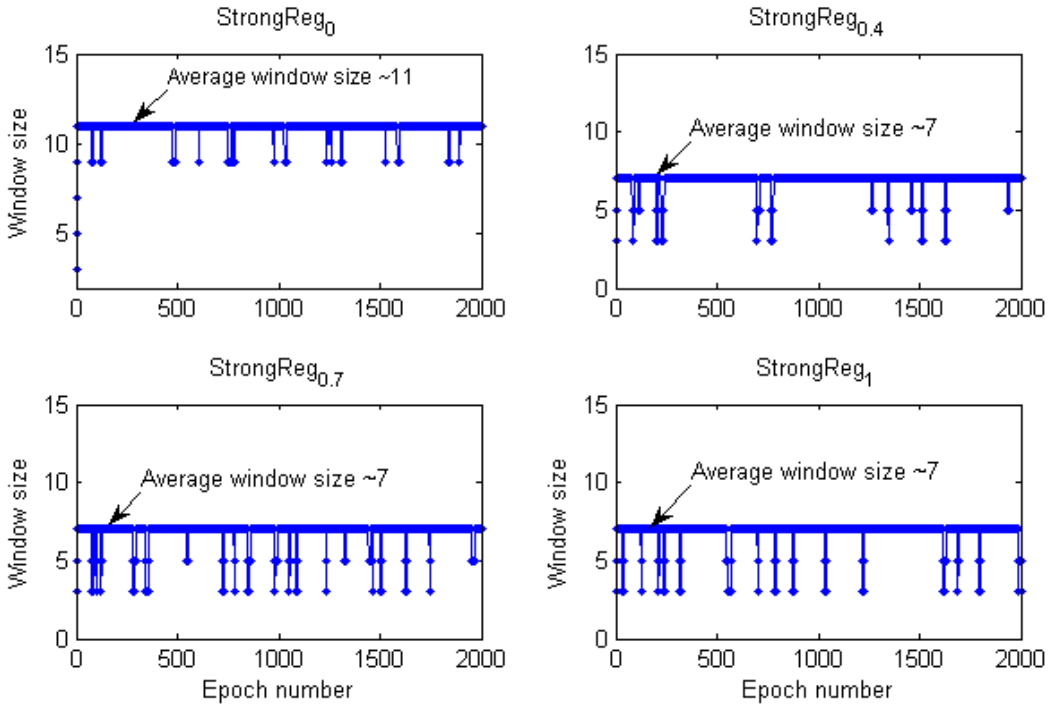


Figure 7-22: WSTD per tag window sizes as a single static tag placed on the middle of detection range is cleaned in different environmental conditions

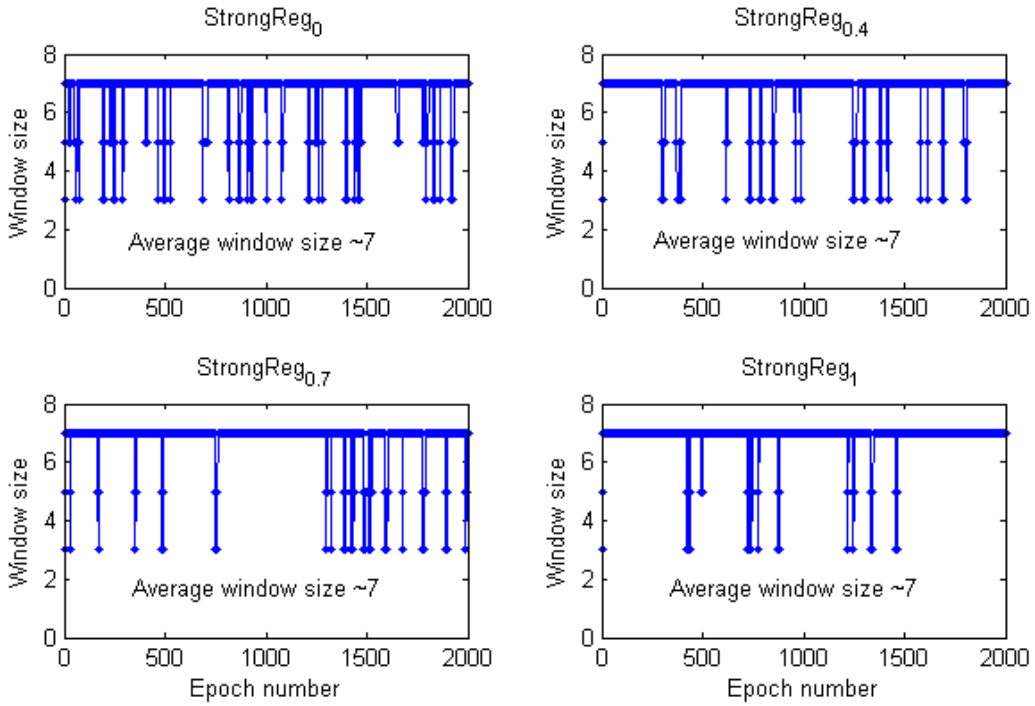


Figure 7-23: WSTD per tag window sizes as a single static tag placed closer to the reader is cleaned in different environmental conditions

As it can be observed from these figures, the cleaning window size depends on the two factors, the distance of the tag relative to the reader, and the environment. When the tag is closer to the reader, as in the case of Figure 7-23, the tag(s) have a high detection range and its window size is not affected by the environment noise. As the tag(s) are poisoned further away from the reader, its detection rate decreases and the cleaning window size becomes sensitive to the environment noise. Figure 7-22 shows the cleaning window size for the tag placed in the middle of the detection range. It can be observed that in a controlled environment ($StrongReg_1$) the average cleaning window is about 7 epochs, and at a high noisy environment ($StrongReg_0$) the average cleaning window size grows to 11 epochs.

Figure 7-21 shows the cleaning window size for a tag placed on the far edge of the detection range. In this case a tag has a very low detection rate causing the cleaning window to be very sensitive to the environment noise. In a controlled environment ($StrongReg_1$) the average cleaning window is about 7 epochs, and it grows linearly as the noise increases. In a high noise environment ($StrongReg_0$), the average cleaning window size grows to an average of 111 epochs.

In a highly controlled environment ($StrongPercentage = 1$) even the static window scheme '*Fxd5'* with window size of 5 epochs outperforms the WSTD variable scheme while using the smaller window size than the average window size used by WSTD (i.e. 7 epochs). This is because the per tag cleaning window increases from 1 epoch linearly in steps of 2 epochs and when the window sizes are still smaller it fails to compensate for the missed readings.

Using the same procedure we also analyzed the WSTD per tag window sizes used to clean mobile tags exhibiting random behaviours under different environmental conditions. The data generated from the previous experiment were used to compare the effectiveness of cleaning schemes whose results are shown in Figure 7-4. In this case, one tag was randomly selected to study its cleaning window sizes as the environment changes. The results of this analysis are shown on Figure 7-24. The top graph of the figures shows the tag movement as it moves in and out of the detection range within the distance of 0 to 6 m.

The maximum reader detection range is 4.6 m and beyond this distance the tag is out of the detection range. The tag moves with a constant speed for an average of 100 epochs, it stops for duration of about 100 epochs and then starts moving with a different constant speed again. This movement simulates the tracking environment, such as digital office where each tag displays independent random behaviour. The other three graphs in the Figure are the cleaning window size traces used to clean the tag readings under different environmental conditions where $StrongReg_0$ refers to a noisy environment and $StrongReg_1$ refers to a controlled environment.

There are two main observations from these figures. The first one is that the cleaning window size is sensitive to the environmental noise level. As the noise in the environment increases, the average cleaning window size also increases. For example in Figure 7-24, the average cleaning window size in the noise environment ($StrongReg_0$) is about 9 epochs and the size decreases as the noise decreases. In a controlled environment ($StrongReg_1$) the average cleaning window size decreased to 5 epochs.

The second observation is related to the type of tag movement. This movement involves a tag stopping at a random location relative to the reader for some duration of time. The distance of the position where the tag stops relative to the reader's antenna has a

significant effect on the cleaning window size as it was observed in the static tag analysis discussed before. It can be deduced from the window size graphs that all the large window spikes in the window size traces correspond to a tag that has stopped during the tag movement trace. It is also interesting to note the correlation between the tag large window spike size and its corresponding tag distance from the reader. When the tag stops on the far edge of the detection range (see epoch 900 on Figure 7-24), it causes the window size to increase dramatically.

In this experiment, the cleaning window size increased to 47 epochs while the average window size at around 1200 epochs, is only 9 epochs. This window size increase is caused by the fact that a tag at this position is continuously detected with low detection rates causing a large window size. The situation is more severe in the noisy environment where a tag at this location is more likely to be missed and hence the average probability of detection becomes even smaller causing a much bigger window size according to Equation (5). As in the case of static tag the large window size spikes caused by tags stopping far away from the reader (but within the detection range) decreases as the environmental noise decreases as it can be observed in Figure 7-24.

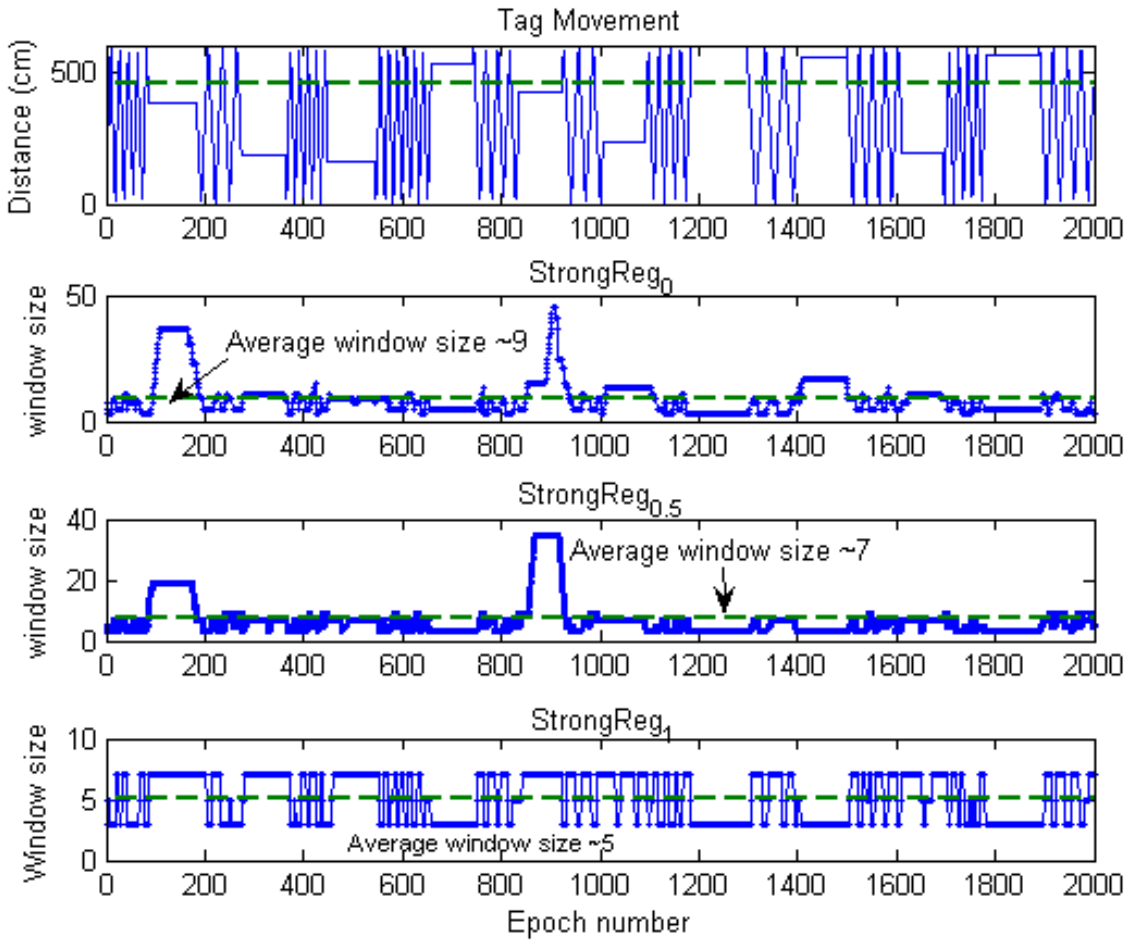


Figure 7-24: WSTD per tag window sizes as a single mobile tag is cleaned under different environment conditions.

The last analysis for the WSTD individual tag variable window size is to check how the cleaning window size varies as the tag speed varies in different environments. Figure 7-25 shows the result of this analysis. The first row graphs shows the tag movement at different velocities, the first column is the results of the tag moving at a velocity of 10 cm/epoch, the second columns shows the results of the tag moving at a velocity of 40 cm/epoch and the third column shows the results of the tag moving with a velocity of 90 cm/epoch. Only few epochs trace i.e. 600 epochs for the slower velocity 10 cm/epoch and 300 epochs for faster velocities are shown for clarity purpose. The second to the fourth row figures shows the variable window size traces at different velocities and different environmental noise levels. The second row graphs ($StrongReg_0$) shows the noisy environment with the $StrongPercentage$ parameter set to 0%, the third row graphs ($StrongReg_{0.7}$) shows the controlled environment with the

$StrongPercentage$ parameter set to 70%, and last row graphs ($StrongReg_1$) shows a controlled environment with its $StrongPercentage$ parameter set to 100%.

There are two main observations in this Figure that can be made by looking at its dimensions. The first one is along the vertical dimension, which is the variation of window size when a tag moves at a constant speed while the environment is varied. In general, at a constant speed, the average cleaning window size decreases as the environmental noise decreases. This is because as the noise decreases the reader produces more reliable data and the window size becomes less sensitive to the environment.

The second observation is along the horizontal dimension of the figure, which is the variation of window size when the tag is operating in the same environment while the tag velocity is varied. In general, it can be observed that the window size decreases as the tag speed increases in all the environments except for the highly controlled environment (the last row graphs) where the average window size seems to be almost constant.

Consider the window size trace graph in the second row and first column (i.e. window size trace for the tag moving at 10 cm/epoch and operating in a noisy environment $StrongReg_0$). As the tag moves away from the reader, its window size increases and when transition is detected the window size is reduced. The window size is specifically increased when a tag approaches the further edge of the detection region, a similar concept to a static tag scenario. Hence, when a tag is moving with a slow velocity it takes a longer time moving out of the detection region and during that time the reader continuously detect tags with decreasing, lower detection rates leading to a continuous increase of the cleaning window size. On the other hand, as the speed increases a tag takes shorter time to move out of the detection region, which means the reader detects a tag with low detection rate fewer times and hence the cleaning window size is increased fewer times.

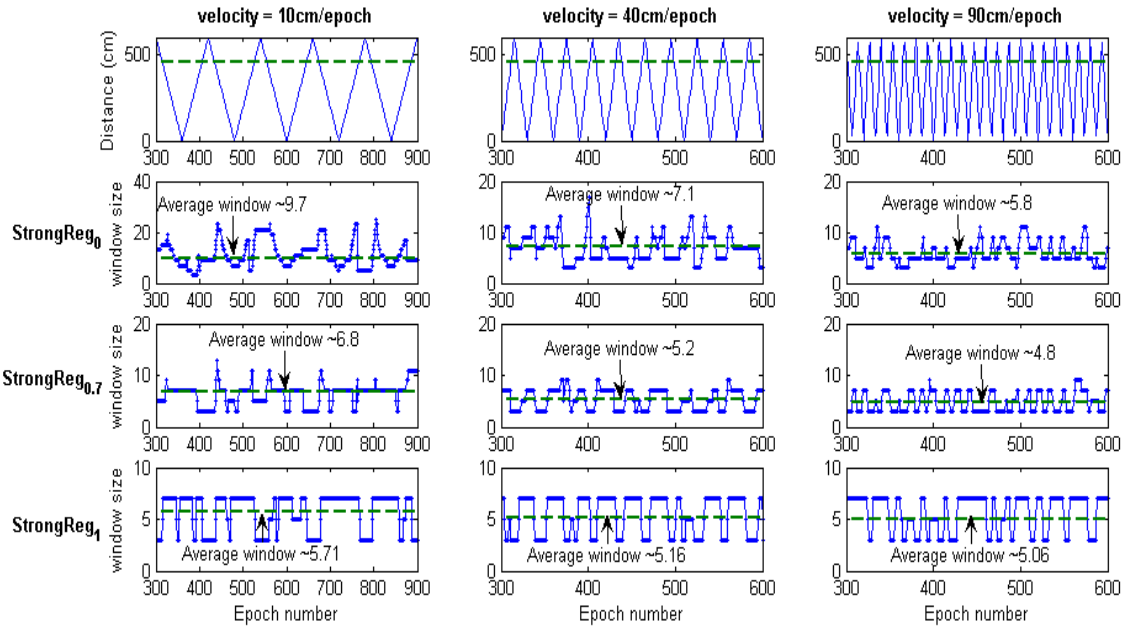


Figure 7-25: WSTD per tag window sizes as a single mobile tag in different velocities is cleaned under different environmental conditions.

In summary, if one defines efficiency in terms of the time it takes to clean the data, the WSTD per tag variable window scheme works more efficiently in high speed environments. Its efficiency deteriorates in low speed and static environments. In static environments - especially when a tag is placed on the far edge of the detection range - the window size grows too big. For example, in Figure 7-21, the average window size for cleaning the tag placed on the far edge of the detection region in the noisy environment is about 111 epochs, while the overall average cleaning window for all tags in the same environment is about 19 epochs. If we assume an epoch to be 1 second, this is approximately to 2 minutes cleaning window. This type of cleaning window can be too long and unacceptable for low level cleaning of data streams and can result in delays in application processing. However, the experiment data shows that this window size can be the optimized window size to clean that data. This fact can be observed in the static experiment in Figure 7-19 at 50% controlled environment in which the WSTD per tag scheme has the same performance as the static window '*Fxd25*' scheme. On this particular point, while the '*Fxd25*' uses the window size with 25 epochs, WSTD uses an overall average cleaning window size of 11.3 epochs (see Figure 7-19), which is half of that of fixed window size. In the same figure, at 80% controlled environment WSTD has the same performance with the '*Fxd10*' scheme, which uses a 10 epoch's window while WSTD uses an overall average of about 7.6 epochs to produce the same result.

7.4 Tags Aggregate Cleaning

The cleaning techniques, which report the number of tags in the detection region instead of individual tag Id, were then examined. The performances of different multi-tag aggregate cleaning schemes are compared as the tag movement and reliability of the environment are varied. Tag movement is changed by varying the tag's velocity and the reliability of the environment is changed by changing the $StrongPercentage$ parameter. The static window schemes (*Fxd25*, *Fxd10*, *Fxd5* and *Fxd2*) count the distinct number of tags within their window, and the variable window method used here detects transitions by comparing the window sub-range population count and uses the π-estimator to estimate the number of distinct tags within the window. This cleaning scheme is denoted as "WSTD-π". The result of SMURF and WSTD population count, which reports the number of distinct tags in each epoch, is also included.

The evaluation metric used for multi-tag cleaning is the root mean square (RMS) error of the count of reported tags compared to the actual tag count. The RMS error is calculated using Equation (20).

$$RMS\ Error = \sqrt{\frac{\sum_{i=1}^{NumEpochs}(ReportedCount_i - ActualCount_i)^2}{NumEpochs}} \tag{20}$$

The mean error of the estimated tag count was also compared to see the contribution of the overestimate and underestimate tag count using Equation (21) and (22) respectively.

$$OverEstimateErrors = \frac{\sum_{i=1}^{NumEpochs} ReportedCount_i - ActualCount_i}{NumEpochs} \tag{21}$$

$$UnderEstimateErrors = \frac{\sum_{i=1}^{NumEpochs} ActualCount_i - ReportedCount_i}{NumEpochs} \tag{22}$$

7.4.1 Experiment 4: Effect of Tag Speed on Tags Aggregate Cleaning

The tag count accuracy of the cleaning schemes is evaluated as the tags' velocity is varied. The $StrongPercentage$ parameter is fixed at 25% to represent the noisy environment and the tags are moved in and out of the detection range at the same

velocity. The velocity is varied from 0 to 90 cm/epoch and we measured the RMS errors produced by each scheme. Figure 7-26 shows the result of this experiment.

This result reveals the same fact; that large static windows are not ideal in cleaning data in the mobile tags environment. The large fixed window schemes beyond saturation speed are unable to detect transition and constantly report all the tags as being present leading to a high number of overestimated tag count. Smaller fixed windows are unable to compensate for missed readings and constantly produce under count results. On the other hand, variable window schemes; SMURF, WSTD and WSTD-π perform better than the fixed window schemes. WSTD-π outperforms all the other schemes, producing relatively stable and lower errors than all the other schemes compared. Its performance is contributed to its ability to detect transition and adjust the window accordingly and the use of π-estimator to estimate the number of tags.

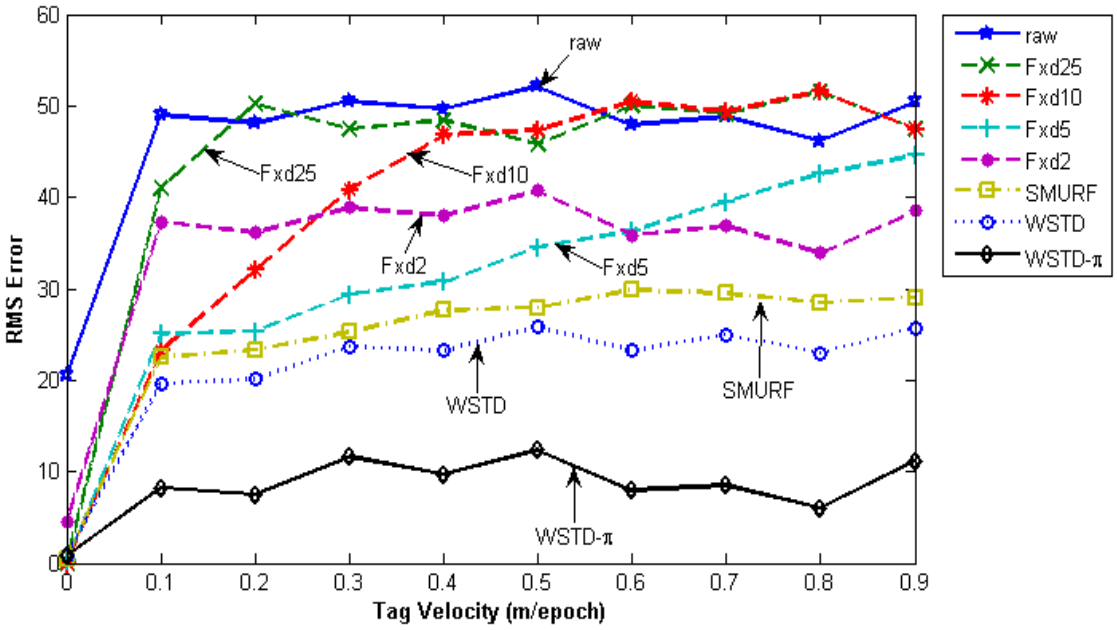


Figure 7-26: The RMS error of different cleaning schemes counting 100 tags as their velocities varies from 0 to 0.9 m/epoch in the noisy environment with the $StrongPercentage$ parameter set to 25%

Figure 7-27 shows the average number of overestimate and underestimate tag counts for the variable window schemes and Figure 7-28 shows their average cleaning-window sizes as the tags' velocity is varied. The WSTD-π scheme has the smallest number of underestimate and overestimate errors and it also uses the smallest average cleaning-window sizes compared to other variable window schemes. The WSTD-π small

overestimate errors are attributed to its smaller window size while its small underestimate errors are attributed to its use of π-estimator to estimate the number of tags.

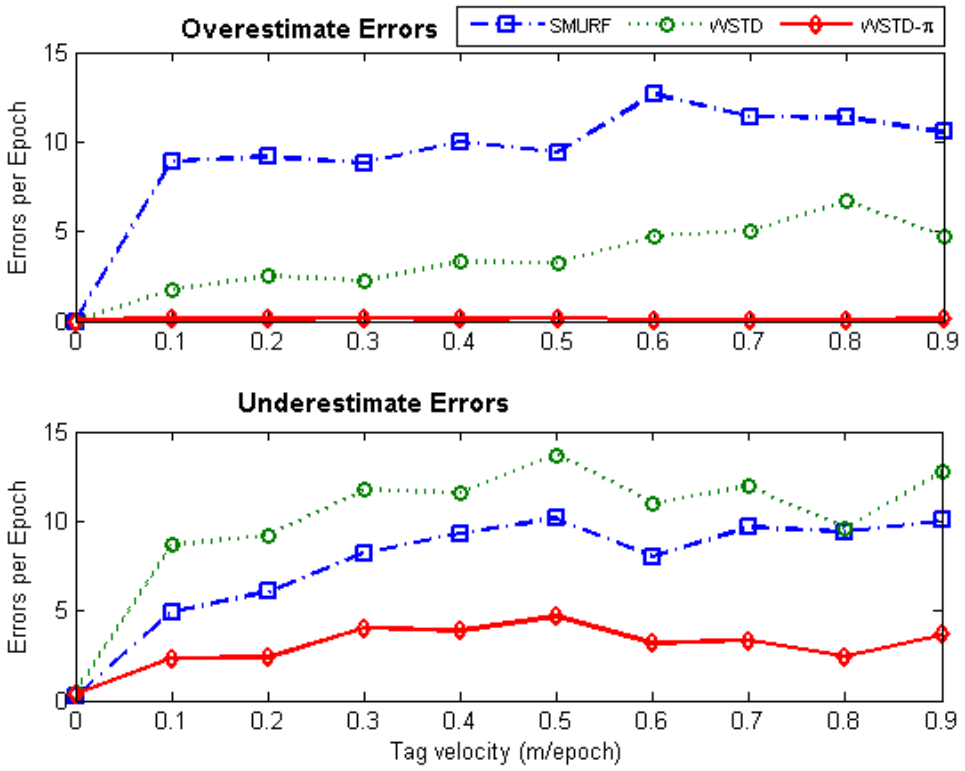


Figure 7-27: Variable window schemes overestimate and underestimate error contributions as the tags' velocity is varied

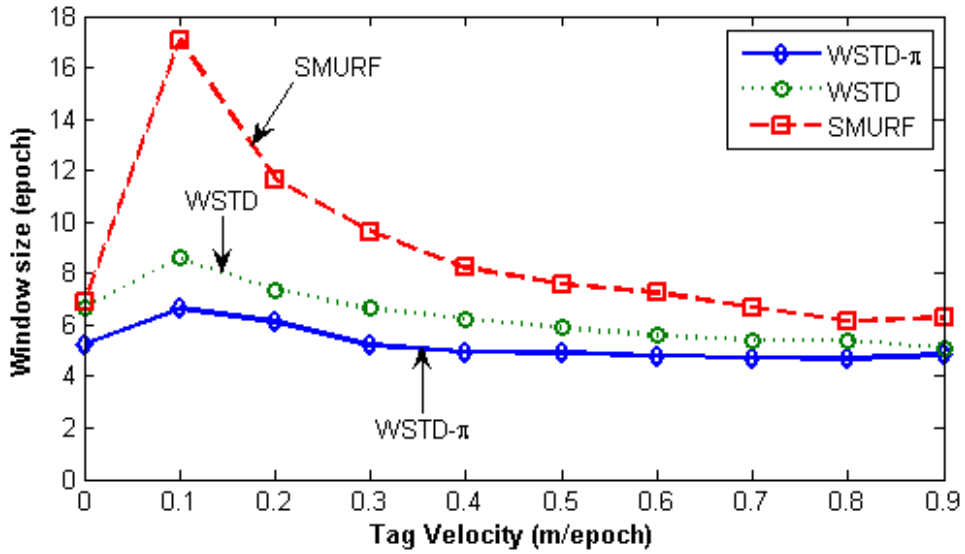


Figure 7-28: Comparison of variable window schemes cleaning windows as the tags' velocity is varied

Although an average number of overestimate and underestimate tag count per epoch as a metric to compare the performance of these schemes was used, it was noticed that most of the undercounting errors occur during the transition periods. This is caused by the nature of algorithms whereby the window size is reduced when an exit transition is detected. While this measure limits the false positive errors, it also leads to false negative errors due to a resulting small window size in case of premature exit transition detection. In addition, when the tags are entering the detection region on the far edge of the detection range, the small window size and a weak read rate leads to a high number of undercounting errors. Figure 7-29 shows the comparison of the reported estimated tags count and that of the actual tag count for the three variable window schemes SMURF, WSTD and WSTD-π with the tags moving at a velocity of 0.4 m/epoch. WSTD-π provides close accurate tag-count estimation compared to other schemes.

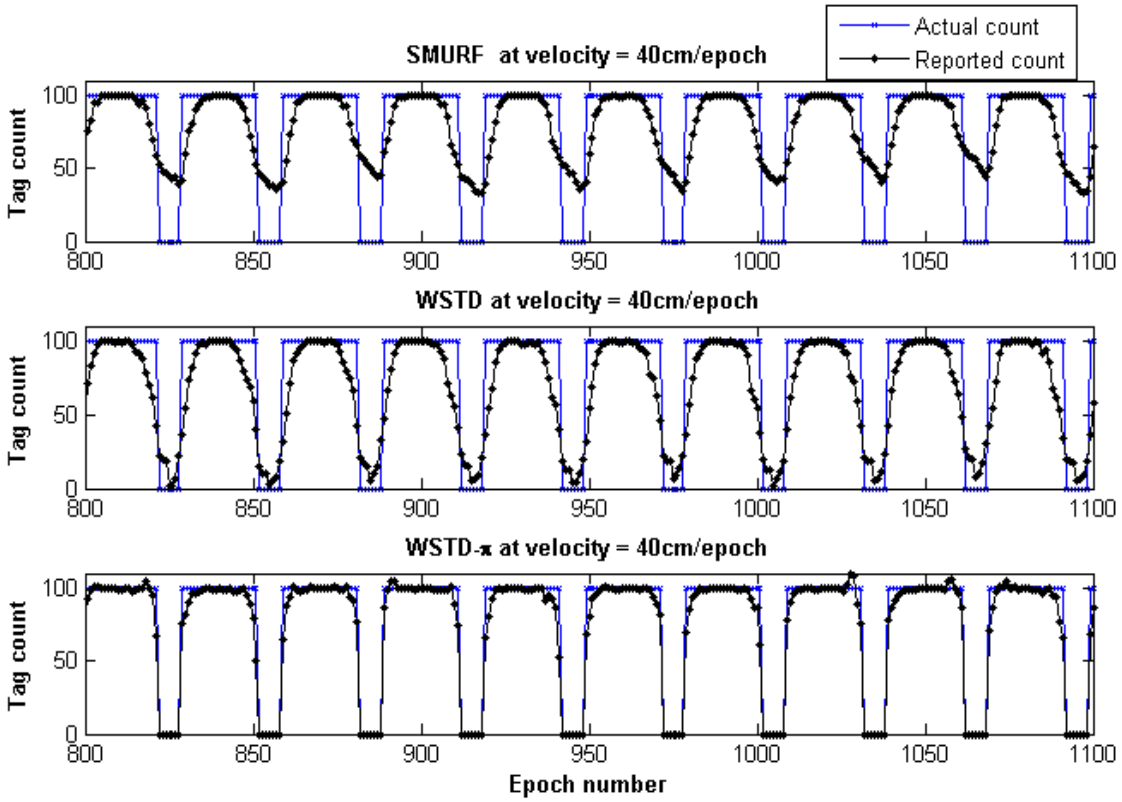


Figure 7-29: Comparison of variable window-cleaning schemes' reported tags count with the actual tag count. All the tags move with the same velocity of 0.4 m/epoch

7.4.2 Experiment 5: Effect of Environment Reliability with randomly moving tags

In this experiment it is determined how each multi-tag cleaning technique reacts to different levels of the environment's unreliability with the randomly moving tags. The experimental parameters are the same as the ones used for the individual tag cleaning experiment, except that in this experiment we used 100 moving tags instead of 25 tags. The strong-in-field region percentage is varied between 0 and 100% and at each $StrongPercentage$ we measure the RMS errors produced by each scheme. Figure 7-30 shows the result of this experiment, poor performing traces 'raw' and '*Fxd2*' are truncated to enable a clear view of other traces. Figure 7-31 shows the average number of overestimate and underestimate tag counts for fixed window schemes as the environment noise is varied.

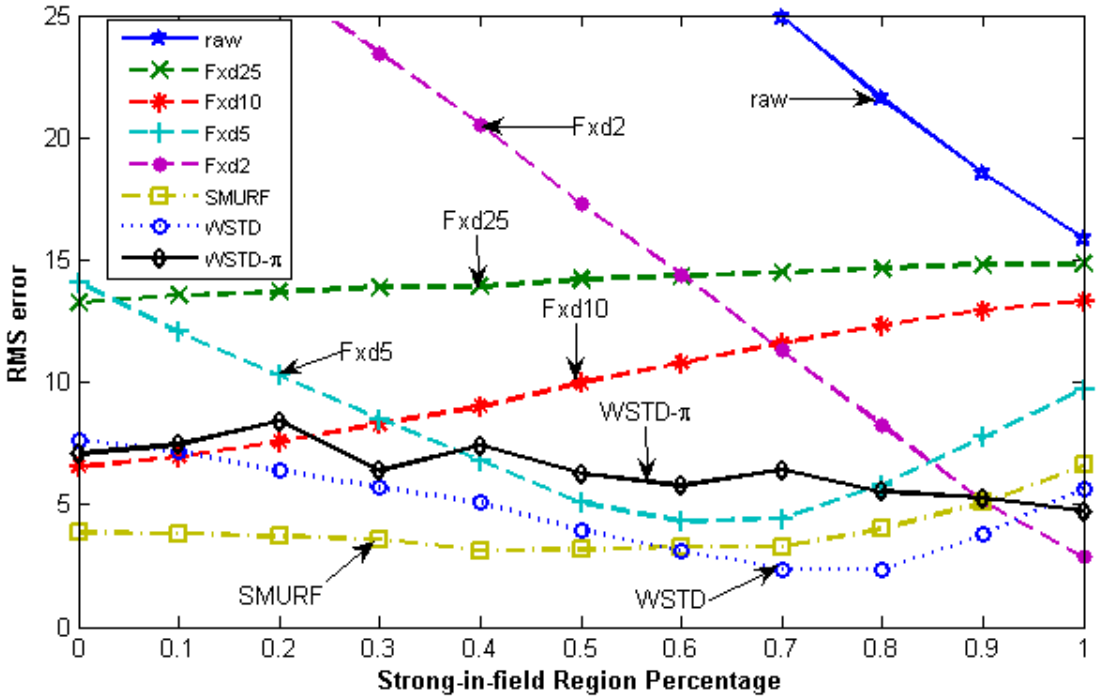


Figure 7-30: RMS errors of the tags count for different cleaning schemes as the $StrongPercentage$ parameter is varied with each tag moving with its own velocity

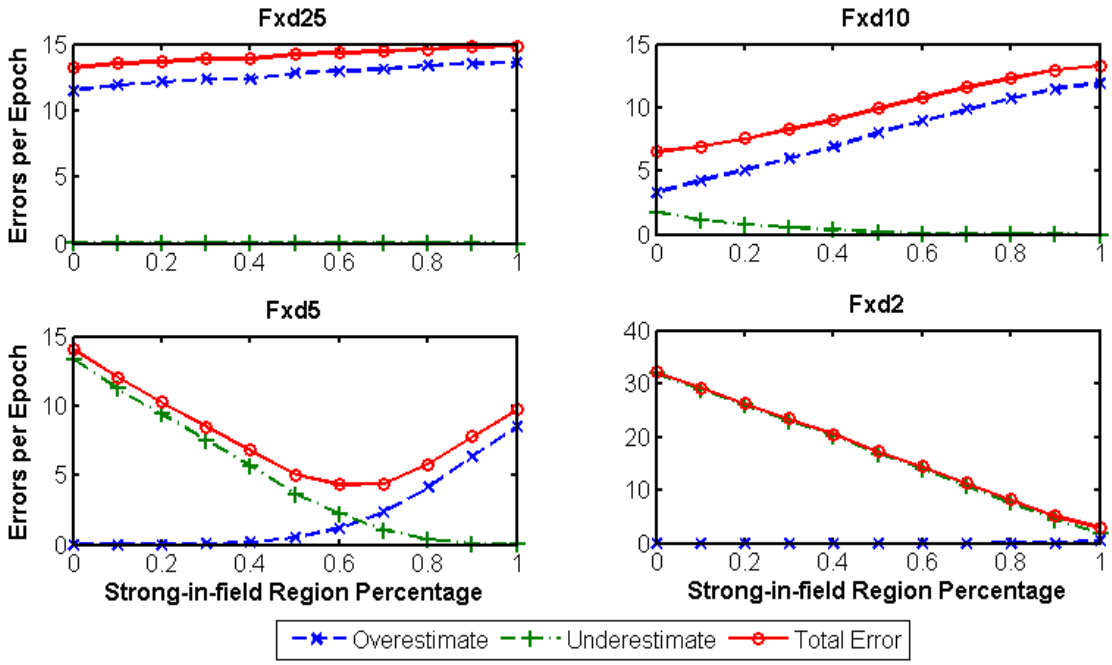


Figure 7-31: Fixed window schemes average overestimate and underestimate error contributions as the $StrongPercentage$ parameter is varied with each tag moving with its own velocity

For the static window schemes (see Figure 7-31), the large window schemes ('*Fxd25*' and '*Fxd10*') errors are highly contributed by overestimate errors due to interpolations of readings within the large window. Small window schemes ('*Fxd5*' and '*Fxd2*') errors are highly contributed by undercount errors due to their inability to compensate for missed readings. As the strong-in-field percentage increases and the reader produce more reliable data, the underestimate errors decreases while the overestimate errors increase. Since the small fixed window scheme '*Fxd2*' has negligible overestimate errors its performance increases consistently as the noise level decreases. The performance of the '*Fxd5*' scheme increases up to the 70% strong percent region and then starts decreasing due to an increase in the overestimate errors as the data becomes more reliable.

Both variable window schemes demonstrate a relatively consistent performance throughout the environment range compared to static window schemes. Figure 7-32 shows the average number of overestimate and underestimate tag counts, and Figure 7-33 shows the average cleaning window size of the variable window schemes as the environmental noise is varied.

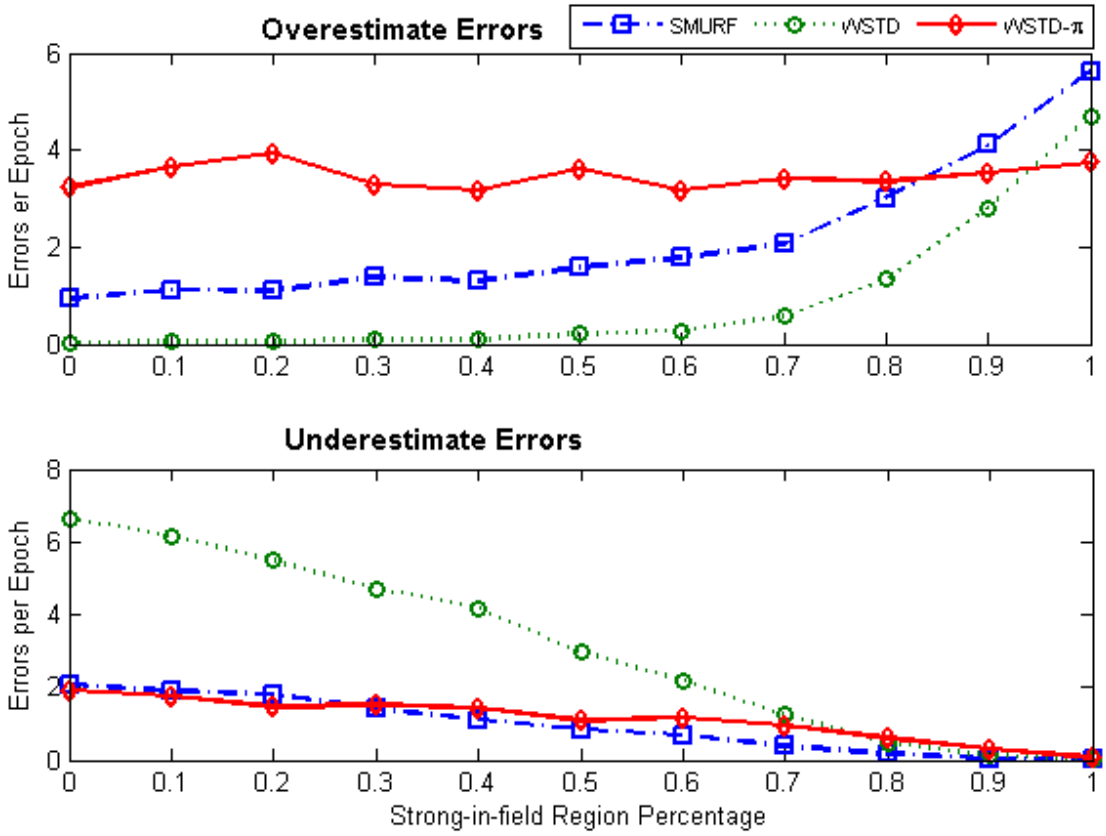


Figure 7-32: Variable window schemes' average overestimate and underestimate error contributions as the $StrongPercentage$ parameter is varied with each tag moving with its own velocity

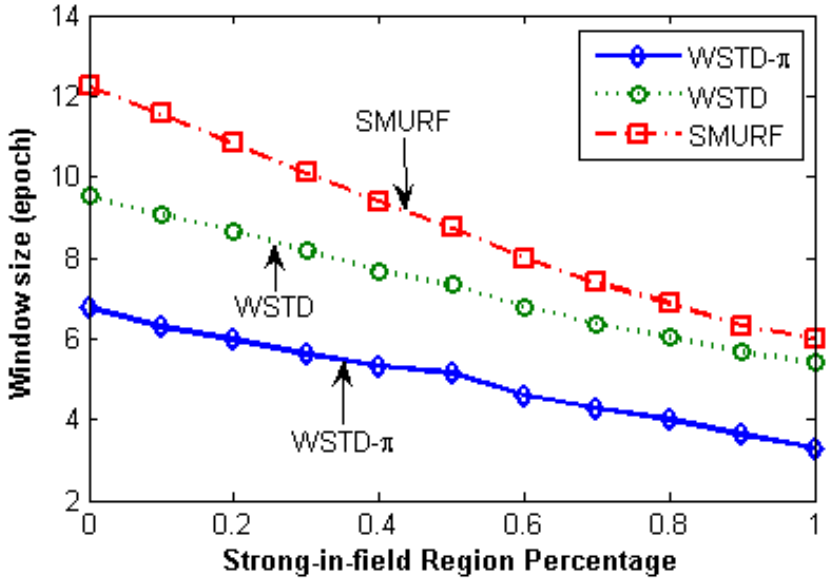


Figure 7-33: Comparison of variable window schemes' cleaning window sizes as the $StrongPercentage$ parameter is varied

From Figure 7-32, comparing the per-tag cleaning schemes; SMURF has consistently more overestimated errors than WSTD, while in a noisy environment, WSTD has more underestimated errors than SMURF. However, as the noise decreases and the reader produces more reliable data, the WSTD-undercount errors also decrease and at a highly controlled environment, its performance outperforms that of SMURF. WSTD-π, on the other hand, consistently produces relatively stable overestimated errors, irrespective of the environmental conditions; however, its underestimated errors decrease with the decrease of the environmental noise (see Figure 7-32). These constant overestimate errors might be caused by the fact that WSTD-π uses the same window size to clean all the tags. In this scenario, each tag moves randomly in and out of detection range with random velocities. The WSTD-π transition mechanism is not able to effectively detect transitions in this scenario; as a result, the π-estimator overestimates the tags' count.

The per-tag variable window-cleaning schemes SMURF and WSTD outperform a single variable window scheme WSTD-π because they clean and adjust their window size for each single tag independently, depending on its individual tag behaviour and how the environment is affecting that particular tag. Figure 7-34 shows the comparison of the reported estimated tags' count and that of the actual tag count for these three variable window schemes. The tags are operating in the semi-controlled environment with $StrongPercentage$ parameter set to 50%. Hence, these experimental results demonstrate that in the environment where each tag displays its own independent random behaviour, the best result is obtained by adjusting each individual tag cleaning window independently. On the other hand, comparing WSTD-π and the best performing per-tag cleaning scheme, in this scenario SMURF; while WSTD-π produces an average of 43% more overestimate errors than SMURF, SMURF uses an average of 43% more processing time (i.e. larger cleaning window sizes) than WSTD-π.

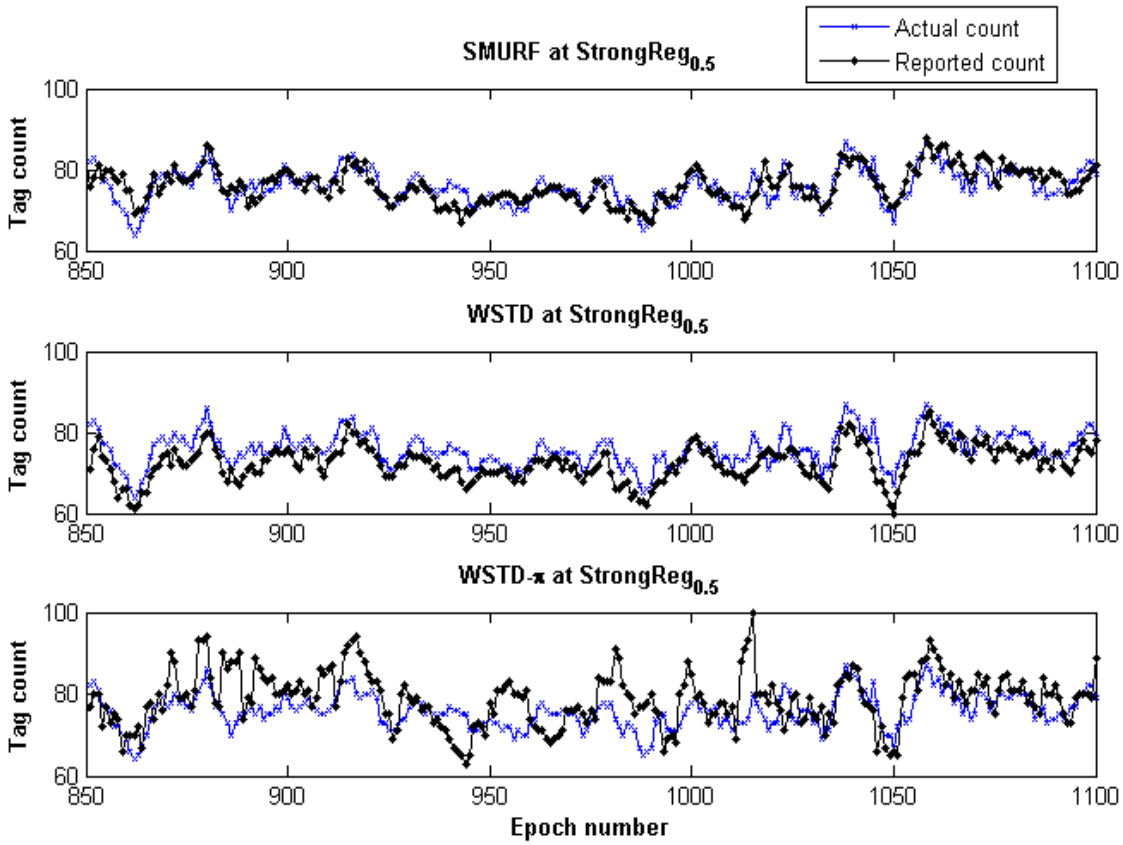


Figure 7-34: Comparison of variable window cleaning schemes' tags count with the actual tag count. Each tag moves with its own velocity

7.4.3 Experiment 6: Effect of Environment Reliability with static tags

The performance of different sliding-window based multi-tag cleaning schemes in the environment where tag(s) are stationary was also evaluated. To simulate this scenario, 100 tags were randomly distributed uniformly within the detection range and the $StrongPercentage$ parameter varied and the root mean square error produced by each scheme measured. Figure 7-35shows 100 tags randomly distributed within the reader's detection range (0 to 4.6 m). Figure 7-36 shows the result of this experiment; the poor performing traces 'raw' and '*Fxd2*' are truncated to enable a clear view of other traces.

All fixed window schemes have the same pattern. Their performance increases with the decrease in environmental noise and their performance also increases with the increase in window size. Since our synthetic data generator does not generate any false positive readings and also because all the tags are static within the detection region, all fixed

window schemes produce zero overestimate errors. Their underestimate err ors decrease as the environmental noise decreases and the reader produces more reliable data.

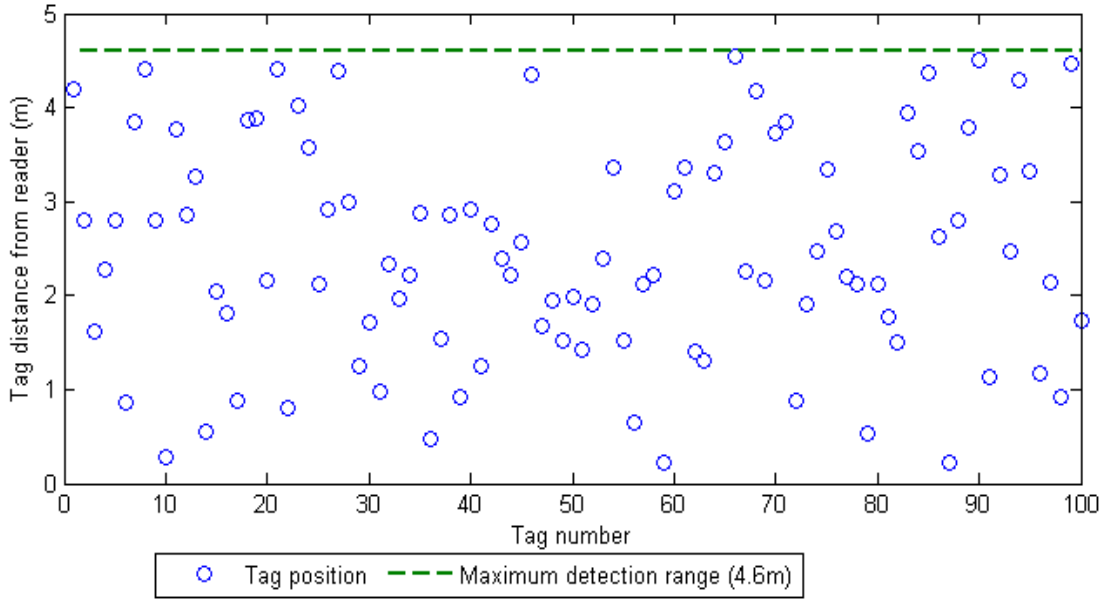


Figure 7-35: 100 static tags randomly distributed within the readers detection range

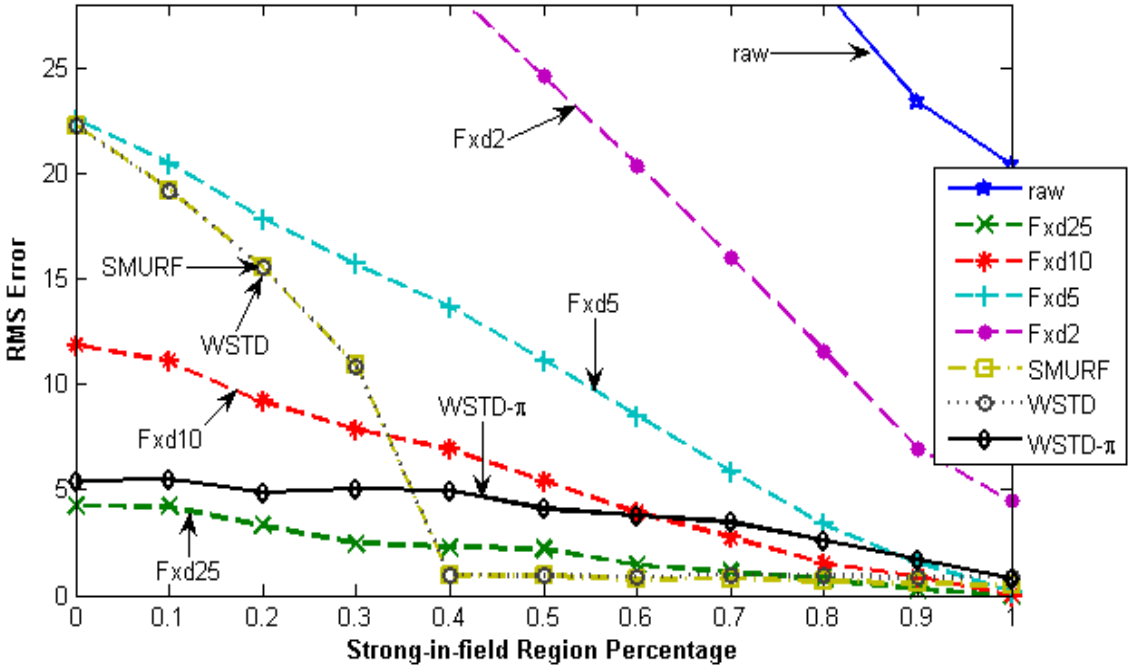


Figure 7-36: The RMS error of different cleaning schemes as the $StrongPercentage$ parameter varies in the static tags environment

Similar to the per-tag cleaning in the static environment scenario, SMURF and WSTD have the same performance in cleaning the tag aggregations in the static environment as shown in Figure 7-36. Since they both count the distinct tags available within each tag's cleaning window they have no overestimate errors and their underestimate decrease with the decrease in noise as shown in Figure 7-38.

In a very noisy environment the performance of per tag cleaning schemes are highly affected by the poor performing tags on the far edge of the detection region. In the noisy environment the far edge static tags cleaning window grows linearly with distance and becomes very large (results not shown). For example, in this experiment tag_x which was located at 437cm from the reader at $StrongPercentage$ of 30% its average cleaning window size grows to 81 epochs while the average cleaning window size in same environment is 21 epochs. In the controlled environment with $StrongPercentage$ of 80%, the cleaning window size for tag_x drops to 25 epochs while the average cleaning window size for all the tags in the same environment dropped to 9 epochs. Because the cleaning window sizes for individual tags might be different based on their detection rates, the decision on whether the tag is present or not is taken at different epochs. Therefore, the tags which are not ready for processing (i.e. the readings for all epochs in its window have not being accumulated) will delay the output and affect the performance of the scheme considerably. This situation is similar to the *blocking* complication in the data stream aggregation explained by Abadi *et al*. in [120], which means waiting for lost or late tuples to arrive in order to finish window calculation.

Abadi *et al*. [120], proposed that given the real time requirements of many stream applications, it is essential to make it possible to "time out" aggregate computations, even if this happens at the expense of accuracy. When an *Aggregate* is declared with a timeout t, each window's computation is time stamped with the local time when the computation begins. A windows' computation then times out if a result tuple for that window has not been emitted by the time that the local time exceeds the window's initial time +*t*. As a result of a timeout, a result tuple for that window gets emitted early and all tuples that arrive afterwards and that would have contributed to the computation had it not timed out are ignored. Therefore, using the same principle the WSTD per tag cleaning algorithm performance efficiency can also be improved by allowing the application user to specify tolerable imprecision in terms of range of acceptable window size as a time out

parameter. This can be very useful especially when dealing with aggregation with poor performing tags.

On the other hand, the WSTD-π scheme, which estimates the tag count based on tag detections and π-estimator, produces both underestimate and overestimate errors as shown in Figure 7-38. Its performance increases with the reduction of environmental noise, with both its overestimate and underestimate errors decreasing. In general, WSTD-π performs competitively well with large fixed window schemes. In the noisy environment with a $StrongPercentage < 20\%$, WSTD-π has a relatively close performance with that of the '*Fxd25*' scheme which uses a window size of 25 epochs, while WSTD-π according to the results in Figure 7-37 in same environment uses an average cleaning window of less than 12 epochs.

In addition, in the 60% strong detection region WSTD-π has the same performance with '*Fxd10*' scheme, which uses 10 epochs window size while WSTD-π in the same environment uses an average cleaning window size of about 7 epochs. In the highly controlled environment from 90% to 100% strong detection region WSTD-π has the same performance as '*Fxd5*' scheme using a slightly higher average window size of 5.5 epochs. We can conclude that the performance of WSTD-π strikes a balance between accuracy and processing speed, by providing a considerably good estimate in a much shorter time.

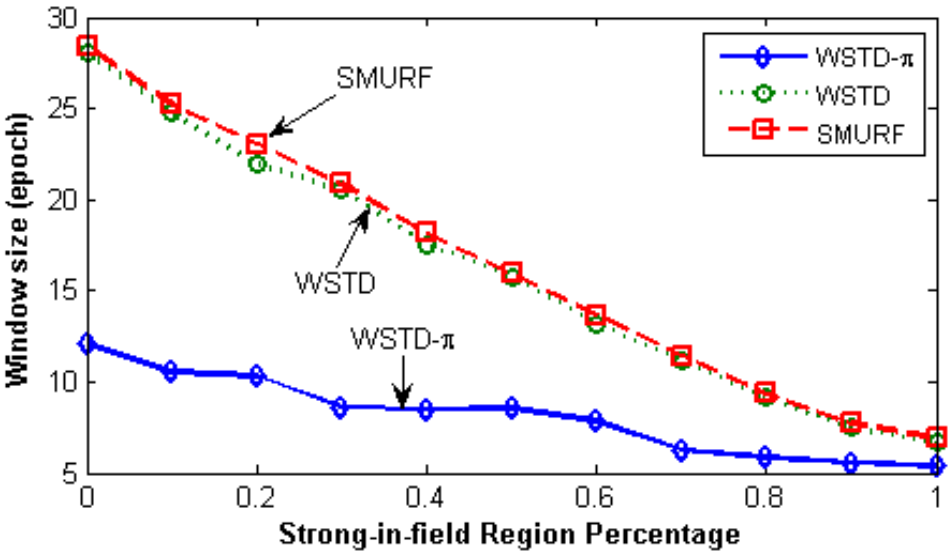


Figure 7-37: Comparison of variable window schemes' cleaning-window sizes as the $StrongPercentage$ parameter is varied in the static tag environment

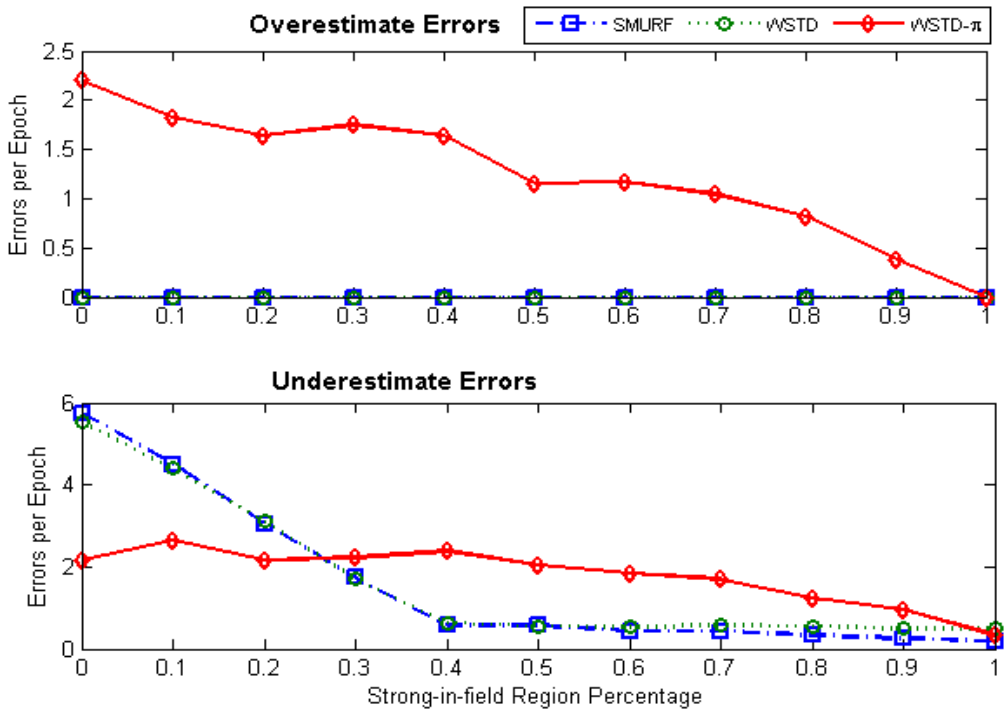


Figure 7-38: Variable window schemes average overestimate and underestimate error contributions as the $StrongPercentage$ parameter is varied in the static tags' environment

7.5 Summary

In this chapter the performance evaluation of the WSTD algorithms in comparison with that of SMURF and other fixed window-based cleaning methods using the generated synthetic data sets was presented. The challenges associated with setting the appropriate cleaning window size were also discussed.

In general, our experimental results show that there is no single static window size which can perform consistently well in all the environments. Large static window sizes perform well in static environment as they are able to compensate for the missed readings but their performance deteriorates in the mobile environment as they consistently introduce false positive readings due to interpolation of readings within the window. The false positive errors increase as the window size and tags speed increases. Small static window sizes are good in detecting transition and they introduce less false positive errors, however, they are unable to compensate for missed readings. Adaptive variable sliding window filters provides an alternative solution to cope with the tag-reader performance environment sensitivity and tag dynamics. Adaptive variable sliding window schemes automatically adjusts the cleaning window size based on the characteristic of the underlying data

stream. Experimental results show that variable window schemes have proven to provide consistent and good performance in all the environmental conditions.

In the mobile environment with variable environmental noise level, the WSTD scheme performs better in comparison to the SMURF scheme producing an improvement of approximately 30% less overall error compared to that produced by SMURF. This performance improvement is attributed to its improved transition-detection mechanism. The WSTD scheme uses smaller window sizes compared to SMURF, which means that WSTD also takes shorter processing time compared to SMURF.

The WSTD-π scheme which uses its π-estimator to estimate, the number of tags performs more efficiently in static and mobile environments in which all the tags are moving with the same speed. In the mobile environment where every tag exhibits its own random behaviour, the WSTD-π scheme is unable to handle transitions caused by randomly moving tags and consistently overestimate the tag population. In randomly moving tag environments, the per-tag cleaning schemes outperform the single variable window scheme WSTD-π, because they clean and adjust their window size for each single tag independently depending on its individual tag behaviour and how the environment is affecting that particular tag. However, the WSTD-π uses much smaller cleaning window sizes compared to those used by per-tag cleaning schemes. Hence, depending on the type of application selection of cleaning method can be a trade-off between efficiency and accuracy.

The developed middleware implements the described variable window cleaning techniques, WSTD and the WSTD-π, to clean the low level RFID data streams. Instead of the application setting the cleaning window size, the applications only configures the type of application scenario and whether it requires the actual tags or the tags count. Based on this information the middleware then chooses the best cleaning scheme to use and automatically adjust its window size according to the characteristics of the underlying data streams. For all applications which require the actual RFID data together with applications which involves mobile tags exhibiting random behaviours, the middleware uses the WSTD per-tag cleaning scheme. The WSTD scheme is used in applications which require the tags count in the environment where all the tags exhibit the same behaviour such as static tags environments, or all tags moving with the same velocity. If the application requires both actual RFID data and the tag count the WSTD per-tag cleaning scheme is used to produce both results.

Chapter 8: Conclusion

In conclusion, the contributions of this research are summarized and some important observations are noted. Areas of possible future work that would extend the present work beyond the limitations and constraints given are also suggested.

8.1 Contributions

The contributions of this research work can be summarized as follows:

1. **Temporal-based RFID data model**

   The development of the temporal-based RFID data model which provides an efficient support for the RFID event processing within the middleware. The data model developed considers both applications' temporal and spatial granules in the data model itself and extends the conventional Entity-Relationship constructs by adding a time attribute to the model. By maintaining the history of events and state changes, the data model captures the fundamental RFID application logic within the data model. Hence, this data model supports efficient generation of application level events; updating, querying and analysis of both recent and historical events.

2. **New Adaptive Sliding-Window data cleaning Scheme (WSTD)**

   The development of a new adaptive sliding-window based data cleaning scheme for reducing missed readings from RFID data streams called WSTD. The WSTD scheme models the unreliability of the RFID readings by viewing RFID streams as a statistical sample of tags in the physical world, and exploits techniques grounded in sampling theory to drive its cleaning processes. The WSTD scheme is capable of efficiently coping with both environmental variation and tag dynamics by automatically and continuously adapting its cleaning window size based on observed readings.

3. **RFID based on multi-agent system**

   The development of a multi-agent based RFID middleware which addresses some of the RFID data and device management challenges. The middleware developed abstracts the auto-identification applications from physical RFID device specific

details and provides necessary services such as device management, data cleaning, event generation, query capabilities and events persistence. The use of software agent technology offers a more scalable and distributed system architecture for our middleware.

4. An Ontology for RFID Domain Devices

The development of the RFID device ontology. In addition to task ontologies, which describe the entities relevant to problem solving tasks and methods within the middleware, the RFID domain device ontology was also developed. The device ontology developed is an application independent ontology and it can, hence, be used or extended in any application interested in the RFID domain ontology.

8.2 Conclusions

The widespread adoption of RFID with ultimate performance requires not only low cost tags and readers, but also appropriate software and architectural design. In addition, a justifiable return from RFID technology investments will only come from intelligent use of the data harvested from RFID systems. RFID middleware is the tool that helps to make sense of RFID tag reads; it cleans the unreliable RFID data streams, translates the simple read data into useful information and propagates it to the appropriate enterprise information systems. In this thesis, some common, fundamental characteristics of RFID data and devices, which pose significant challenges in the design of RFID middleware system, were discussed. A multi-agent based RFID middleware which addresses some of the data and device management challenges was also developed. The developed middleware abstracts the auto-identification applications from physical RFID device specific details and provides necessary services such as device management, data cleaning, event generation, query capabilities and event persistence.

8.2.1 Temporal-based RFID data model

One of the challenges in RFID middleware design addressed in this thesis is RFID event processing. Raw data generated from an RFID system carries implicit information about business processes such as changes of states, locations, and containment relationships among objects. Extracting this implicit information from the incoming simple raw data

(i.e. a tuples containing reader ID, tag ID, and the timestamp) is the most interesting and challenging issue in an RFID middleware system. To address event processing within the middleware, in Chapter 3, a temporal-based RFID data model which considers both applications' temporal and spatial granules in the data model itself for efficient event processing was developed.

This data model extends the conventional ER construct by adding a time attribute to the model. The ordinary relationship types are given temporal semantics, making their instances record variation over time, rather than just single states. By maintaining the history of events and state changes, the data model captures the fundamental RFID application logic into the data model itself. Although state changes information can be derived from events data, explicit modelling of the state changes information into the data model provides a better support for complex queries. Most of the RFID queries are time based queries with temporal constraints such as history, temporal snapshot, temporal slicing, temporal joins and temporal aggregates. History queries retrieve the history information of an object, for example location history, temperature measurements history, or object transaction. Snapshot queries retrieve the snapshot information of an object, for example location at time t, temperature at time t, etc. Temporal slicing queries retrieve the information of the object during the time interval (t_1, t_2). Temporal join queries find information by joining multiple relations on a certain constraint. Temporal aggregation queries summarize aggregation information at certain snapshot or interval. Hence, the data model supports efficient generation of application level events, updating, querying and analysis of both recent and historical events.

8.2.2 New Adaptive Sliding-Window based data cleaning Scheme (WSTD)

Another challenge in RFID middleware design addressed in this thesis is cleaning of unreliable data streams generated by RFID readers. Like any other sensor device, RFID readers produces data that are unreliable, low-level and seldom able-to be used directly by applications. The RFID tag-reader performance is highly sensitive to the operating environment. Therefore, despite the improvements on tag detection rates of passive UHF RFID systems by using of EPC Class 1 Generation-2 protocol, factors such as tag-reader configurations, multipath and unpredictable interferences in the deployment environment still contributes to degradation of the performance and reliability of the RFID system leading to noisy and incomplete data.

RFID data cleaning is, therefore, essential in order to correct the reading errors, and to allow these data streams to be used to make correct interpretations and analysis of the physical world they are representing. One of the general methods used in many commercial RFID-middleware solutions to clean the RFID data streams is the use of a fixed temporal-based sliding window data-smoothing filter, and applications are required to set the window size. The goal is to reduce or eliminate dropped readings by giving each tag more opportunity to be read within the smoothing window. The main disadvantage of this method is that setting the smoothing-window size is a non-trivial task, especially in the mobile environment. It requires a careful balance between the two opposing application requirements, which are (1) to *ensure completeness* for the set of tag readings due to tag-reader system unreliability, and (2) to *capture tag dynamic* due to tag movement in and out of the reader's detection region.

Large window sizes are good in ensuring completeness by smoothing out the missed readings, but they are not efficient in detecting tag transitions. On the other hand, small window sizes are able to detect transitions, but they are not capable of compensating for the missed readings. Small windows lead to false *negative* errors in which the tag is mistakenly assumed to be absent while it is actually present. In the mobile tag environment big window sizes, while trying to compensate for the missed readings, introduce other errors which are known as *false positive* errors. Our experimental results in chapter seven show that there is no single window size that can perform consistently well in variable environmental and dynamic conditions. Taking into consideration the sensitivity of the tag-reader performance on the deployment environment, this means that a small change in the environment can render the initially optimised cleaning window unable to smooth the data.

In Chapter 6, the adaptive sliding-window based data cleaning scheme for reducing missed readings from RFID data streams called WSTD was presented. Adopting and extending the concepts proposed in SMURF, the WSTD models the unreliability of the RFID readings by viewing RFID streams as a statistical sample of tags in the physical world, and exploits techniques grounded in sampling theory to drive its cleaning processes. The WSTD scheme is capable of efficiently coping with both environmental variation and tag dynamics by automatically and continuously adapting its cleaning window size based on observed readings. Compared to SMURF, WSTD provides the

same performance in the static environment but performs better than SMURF in the mobile environment.

The experimental results show that, in the mobile environment in variable environmental noise levels, the WSTD scheme performs better than SMURF; producing an improvement of approximately 30% less overall errors than that produced by SMURF. This performance improvement is attributed to its improved transition detection mechanism. The WSTD scheme uses smaller window sizes compared to SMURF, which means that WSTD also requires shorter processing time than SMURF.

8.2.3 RFID middleware based on multi-agent system

Taking into consideration the deployment nature of most of RFID systems, and all the complex processes that are supposed to take place within the RFID middleware system (data capturing, cleaning, event processing and integrating RFID events with client applications) the middleware prototype was developed using multi-agent system (MAS) methodologies. There is a strong correlation between RFID system deployments and the types of applications supported by multi-agent systems. Most of RFID system deployments are distributed in nature; RFID devices installed in different strategic locations within the organizations are linked together creating a distributed network of devices. MAS, on the other hand, is characterized by the ability to solve problems in which data, expertise or control is distributed and it also allows for an easy integration of the new system with the existing legacy system. MAS offers system scalability and load balancing, which are the features desired in RFID systems. Therefore, by the use of software agent technology, the developed middleware offers a more scalable and distributed system architecture.

8.2.4 An Ontology for RFID Domain

Ontologies play a vital role in the development of multi-agent systems. For the agents to interoperate, cooperate or coordinate, they need a shared understanding of the domain they inhabit. That common representation of the objects, concepts, entities and relationships within the domain is referred to as an Ontology. All the information to be exchanged between the agents must conform to a common ontology. In addition to task ontologies, which describe the entities relevant to problem-solving tasks and methods

within the middleware, an RFID device ontology which is application-independent ontology was also developed. This ontology can be used or extended in any application interested with RFID domain ontology.

8.3 Recommendations for Future Research

In section 1.6 of Chapter 1, the limitations of this study were mentioned. In order to address these limitations and other potential extensions to this research work identified, the following are suggested as future research areas.

8.3.1 Reducing False Positive Readings

There is a need to investigate possible cleaning methods for reducing false positive readings from RFID data streams to be incorporated within the middleware. As explained in section 6.2, false positive readings are readings in which the tag is mistakenly present within a certain location while it is not. This can be caused by the RFID tags outside the normal reading scope of the reader being captured by the reader, and it can also be produced as a by-product of the cleaning scheme. Window-based cleaning schemes when used in mobile environment tend to produce false positive readings, by interpolation of readings within the big cleaning-window sizes. It will be an interesting study to try to minimize the false positive readings produced by both RFID reader and the cleaning methods.

8.3.2 Integrating the RFID Middleware with legacy Systems

Different ways in which the middleware system can be integrated with the legacy application systems and security measures to prevent access of the middleware resources and services by unauthorized applications could also be an area of possible future research work. The current, implemented prototype is limited and incomplete in the sense that it uses a graphical user interface in which a user can configure the desired type of data and the rules for generating the data from the RFID system. However, for the middleware to be efficient, it should be automatically integrated into other legacy enterprise applications, which require RFID data for further analysis and decision-making processes. The agents within the middleware use XML as their content language, hence one of the ways in which messages with application clients can be handled is by using

XML and TCP or HTTP message-transfer-binding. Another approach that could be explored is the use of web services and service-oriented architectures.

8.3.3 Support for other Sensor Types

To enable automations, RFID system should be integrated with other types of sensors and actuators. In this case, RFID facilitates detection and identification of objects that are not easily detectable or distinguishable by sensor technologies while sensors provide information about the condition of objects as well as the environment, which are not provided by RFID technology. In addition to sensors, applications often have to interact with the physical world quickly using different kinds of actuators such as locks, buzzer or even simple traffic lights to signal an application state to an operator. Therefore, the current prototype middleware should be extended to provide a support for other types of sensors and actuators, and test the middleware with different pervasive computing applications.

8.3.4 Dynamic Load management

Exploring the load balancing benefit offered by the use of multi-agent technology within the middleware is another possible area for future work. In many applications, RFID generated data are considered to be asynchronous in nature. The asynchronous incoming data can cause the system to handle a large amount of data over a particular interval of time while the system is under-loaded in a different time window. An RFID middleware must be able to dynamically manage unexpected data load to make sure the applications are not affected. Multi-agent technology by using mobile agents can be exploited to offer a resource migration based dynamic load management by balancing assignment of load from heavily loaded components to lightly loaded components for processing.

List of scientific Publications

The research work presented in this thesis has been presented, discussed and published in various national and international forums as indicated below.

1. Libe Valentine Massawe, Farhad Aghdasi and Johnson Kinyua, "A Multi-Agent Based System RFID Middleware for Data and Device Management", *CUT Interdisplinary Journal, Interim* **2008** Year 7 No.2, ISSN: 1684-498X, pp.115-125

2. Libe Valentine Massawe, Farhad Aghdasi and Johnson Kinyua, "The Development of a Multi-agent based Middleware for RFID Asset Management System using the PASSI Methodology", *IEEE Computer Society, in Proceedings of the Sixth International Conference on Information Technology: New Generations (ITNG)*, Las Vegas, USA, April 2009, ISBN: 978-0-7695-3596-8, pp.1042-1048

3. Libe Valentine Massawe, Johnson Kinyua and Farhad Aghdasi, "An Implementation of a Multi-Agent Based RFID Middleware for Asset Management System Using the Jade Platform", *In Proceedings of the IST-Africa 2010 Conference*, Durban South Africa , May 2010, ISBN: 978-1-905824-15-1

4. Libe Valentine Massawe, Herman Vermaak and Johnson Kinyua, "An Adaptive Data Cleaning Scheme for Reducing False negative reads in RFID Data Streams", in *Proceedings of 6th IEEE International conference on RFID*, Orlando Florida USA, April 2012, ISBN: 978-1-4673-0328-6/12, pp.157-164

5. Libe Valentine Massawe, Johnson Kinyua and Herman Vermaak, "Reducing False Negative Reads in RFID Data Streams Using an Adaptive Sliding-Window Approach", *Journal of Sensors* **2012**, *12*, 4187-4212; doi:10.3390/s120404187

References

[1] A. N. Nambiar, "RFID Technology: A Review of its Applications", in *Proc. World Congress on Eng. and Comp. Sci.* 2009, *vol. 2,* San Francisco, USA, pp. 1253-1259.

[2] Rockwell Automation. "RFID in Manufacturing." (2004, Oct.). Available: http://www.glbinc.com/RFID_whitepaper.pdf [Nov. 10, 2011].

[3] C. Floerkemeier and M. Lampe, "RFID middleware design – addressing application requirements and RFID constraints," in *Proc. Joint SOC-EUSAI Conf.*, Grenoble, 2005, pp 219–224.

[4] F. Wang and P. Liu, "Temporal Management of RFID Data," in *Proc. 31st VLDB Conf.*, Trondheim, Norway, 2005, pp. 1128-1139.

[5] L.V. Massawe, F. Aghdasi and J. Kinyua, "A Multi-Agent Based System RFID Middleware for Data and Device Management," *CUT Interdisplinary Journal, Interim,* **2008,** Year 7 No.2, pp.115-125.

[6] S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma, "Managing RFID Data," in *Proc. 30th VLDB Conf.*, Toronto, Canada, 2004, pp. 1189-1195.

[7] R. Derakhshan, M. E. Orlowska and Xue Li, "RFID Data Management: Challenges and Opportunities," in *Proc. 2007 IEEE Int. Conf. on RFID*, TX, USA, pp.175-182.

[8] S. Sarma, "Integrating RFID," *ACM QUEUE*, October, 2004.

[9] J. Brusey, C. Floerkemeier, M. Harrison and M. Fletcher, "Reasoning about uncertainty in location Identification with RFID," in *Proc. IJCAI-03 Workshop on Reasoning with uncertainty in Robotics,* Acapulco, Mexico, 2003, pp.23-30.

[10] S. R. Jeffery, M. Garofalakis, and M. J. Franklin,"Adaptive cleaning for RFID data streams," *Proc. 32nd Int. Conf. on VLDB*, Seoul, Korea, 2006, pp. 163-174.

[11] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom, "Declarative Support for Sensor Data Cleaning," in *Proc. Int. Conf. on Pervasive Computing, vol. 3968*, 2006, pp. 83-100.

[12] A.S. Pang. Smart Homes & Social Devices: RFID Takes off. The future of RFID: A memo series, Technology Horizon Program, Nov 2005, SR-926D.

[13] B. Carbunar, M. Ramanathan, M. Koyoturk, C. Hoffmann and A. Grama, "Redundant reader elimination in RFID systems," in *Proc. 2nd Annual IEEE Communication Society Conf. on Sensor and Ad Hoc Communications and Networks*, 2005, pp. 176 – 184.

[14] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom, "A pipelined framework for online cleaning of sensor data streams," in *Proc. 22nd Int. Conf. on Data Eng. (ICDE 06)*, Atlanta, Georgia , April 2006, pp. 140-143.

[15] C. Floerkemeier, M. Lampe and C. Roduner, "Facilitating RFID Development with Accada Prototyping Platform," in *Proc. 5th IEEE Int. Conf. Pervasive Computing and Communication Workshops*, 2007, pp.495-500.

[16] C. Bornhoevd, T. Lin, S. Haller, and J. Schaper, "Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure," in *30th Int. Conf. VLDB*, Toronto, Canada, 2004, pp.1182-1188.

[17] European Telecommunications Standard Institute (ETSI). (2010, Feb). Radio Frequency identification equipment operating in the band 865 MHz to 868 MHz. Doc. No. EN 302 208-1.

[18] M. Lampe and C. Floerkemeier, "High-Level System Support for Automatic-Identification Applications," in *Proc. of Workshop on Design of Smart Products*, Furtwangen, Germany, 2007, pp.55-64.

[19] F. Wang, S. Liu, P.Liu, and Y. Bai, "Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Streams," *EDBT, LNCS vol. 3896*, 2006, pp. 588–607.

[20] L. Sullivan. "RFID Implementation Challenges Persist, All This Time Later," Information *Week*, Oct 2005.

[21] W. Hansen and F. Gillert, *RFID for the Optimization of Business Processes, 1st ed*. John Wiley & Sons Ltd, England, 2008.

[22] B. Glover and H. Bhatt, *RFID essentials, 1st ed*. O'Reilly Media, Inc, 2006.

[23] H. Barthel. "Regulatory status for using RFID in the UHF spectrum." (Nov. 2011). Available: http://www.gs1.org/docs/epcglobal/UHF_Regulations.pdf

[24] M. Usami "An ultra small RFID chip: μ-chip," in *Asia-Pacific Conf. on Advanced System Integrated Circuits,* Fukuoka, Japan, 2004, pp. 2–5.

[25] J. Banks, D. Hanny, M. A. Pachano and Les G. Thompson, *RFID Applied*. John Wiley and Sons Inc, 2007.

[26] International Organization for Standardization. Internet: http://www.iso.org

[27] Electronic Product Code Global Inc. Internet: http://www.epcglobalinc.org

[28] K. Finkenzeller, *RFID Handbook, Fundamentals and Application in Contactless Smart Cards and Identification, 2nd ed*. John Wiley & Sons Ltd, 2003.

[29] ITU, Radio Communication Study Groups. (2008, November). Maritime Usage of Radio Frequency Identification Tags for Freight Containers. Annex 14 to Doc. 5B/ 175-E.

[30] M. C. O'Connor, "Gen 2 EPC Protocol approved as ISO 18000-6C," *RFID Journal*, July 11, 2006

[31] EPCglobal Standards Overview. Internet: http://www.epcglobalinc.org/standards/, [April, 6, 2009].

[32] EPCglobal Specifications. Internet: http://www.epcglobalinc.org/standards/specs/, [April, 6, 2009].

[33] N. R. Jennings, K. Sycara, M. Wooldridge, "A Roadmap of Agent Research and Development", *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 1, no.1, pp. 7-38, 1998.

[34] J. Ferber, *Multi-Agent Systems: An Introduction to Artificial Intelligence*. Addison-Wesley, 1999.

[35] E. H. Durfee and V. Lesser, "Negotiating task decomposition and allocation using partial global planning," in *Distributed Artificial Intelligence* vol. 2, L. Gasser and M. Huhns, Eds. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1989, pp. 229–244.

[36] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci, "A Survey of Programming Languages and Platforms for Multi-Agent Systems," *Informatica*, vol. 30, pp. 33-44, 2006.

[37] C. Bernon, M. Cossentino and J. Pav'on, "An Overview of Current Trends in European AOSE Research," *Informatica,* vol. 29, pp. 379-390, 2005.

[38] F. Bergenti, M. P. Gleizes and F. Zambonelli, Eds., "Methodologies and Software Engineering for Agent System," in *The Agent Oriented Software Engineering Handbook,* vol. 11. Kluwer Academic Publisher, New York, 2004.

[39] N. R. Jennings and M. Wooldridge, "Applications of Intelligent agents," in *Agent Technology: Foundations, Applications, and Markets*, N. R. Jennings and M. Wooldridge Eds., Springer-Verlag: Berlin, Germany, 1998.

[40] M. Cossentino, and C. Potts, "A CASE tool supported methodology for the design of multi-agent systems," in *Proc. 2002 Int. Conf. on Software Eng. Research and Practice (SERP'02)*. Las Vegas, NV, USA, 2002.

[41] Java Agent Development Framework (JADE). Website: http://jade.tilab.com/

[42] FIPA – Foundation for Intelligent and Physical Agents. Website: www.fipa.org

[43] T. R. Gruber, "A translation approach to portable ontologies," *Knowledge Acquisition*, vol. 5(2), pp. 199-220, 1993.

[44] M. Uschold, and M. Gruninger, "Ontologies: principles, methods, and applications," *Knowledge Engineering Review* vol. 11(2), pp. 93-155, 1996.

[45] M. J. R. Shave, "Ontological Structures for Knowledge Sharing," *New Review of Information Networking* vol. 3, pp. 125-133, 1997.

[46] C. A. Knoblock, A. Arens, and C.N. Hsu, "Cooperating Agents for Information Retrieval," *in Proc. 2nd Int. Conf. on Cooperative Information Systems*, Toronto, Canada, 1994.

[47] V. Sugumaran and V.C. Storey, "Creating and Managing Domain Ontologies for Database Design," in *Proc. 6th Int. Workshop on Applications of Natural Language to Information Systems*, Madrid, Spain, 2001, pp.17-26.

[48] S. Falasconi, G. Lanzola, and M. Stefanelli, "Using Ontologies in Multi-Agent Systems," in *Proc. 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, Banff, Canada, 1996.

[49] J DiLeo, T. Jacobs and S. DeLoach, "Integrating Ontologies into Multi-agent System Engineering," in *Proc. 4th Int. Bi-Conf. Workshop on Agent Oriented Information Systems (AOIS 2002)*, Bologna, Italy, 2002.

[50] A. Malucelli and E. Oliveira, "Ontology-Services Agent to Help in the Structural and Semantic Heterogeneity," in *Proc. 5th IFIP Working Conference on Virtual Enterprises and Collaborative Networks*, Toulouse, France, 2004, pp.175-182.

[51] S. T. Yuan, "Ontology-Based Agent Community for Information Gathering and Integration," in *Proc. National Science Council: Physical Sci. and Eng. (Part A)*, vol. 23, no. 6, 1999, pp. 766-780.

[52] A Finkelstein, "Interoperable Systems: An introduction," in *Information Systems Interoperability,* B. Kramer, M. Papazoglou and M. Schmidt, Eds., England: Research studies press, 1998.

[53] P. Sheth and J.A. Larson, "Federated database systems for managing distributed, heterogeneous and autonomous databases," *ACM Computing Surveys*, vol. 22, no. 3, pp. 183-236, 1990.

[54] H. Tout, "An Informational Model for Cooperative Information Gathering," *in Proc. Workshop on Ontologies in Agent Systems, 5th Int. Conf. on Autonomous Agents*, Montreal, Canada, 2001.

[55] H. Wache, U. Visser, and T. Scholz, "Ontology construction – an iterative and dynamic task," in *Proc. 15th Int. Florida Artificial Intelligence Research Society Conf.*, Florida, USA, 2001, pp. 445-449.

[56] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, "What are ontologies, and why do we need them?" *J. IEEE Intelligent Systems*, vol. 14, no. 1, pp. 20-26, 1999.

[57] D. Fensel, E. Motta, S. Decker, and Z. Zdrahal, "Using Ontologies For Defining Tasks, Problem-Solving Methods and Their Mapping," in *Proc. 10th European Workshop on Knowledge Acquisition, Modelling, and Management (EKAW-97)*, Heidelberg, Germany, 1997, pp. 113-128.

[58] T. Gruber, "Towards Principles for the Design of Ontologies Used for Knowledge Sharing," *International Journal of Human-Computer Studies,* 43(5-6): 907-928, 1993.

[59] *FIPA Agent Software Integration Specification*, XC00079B, 2001. Available: http://www.fipa.org/specs/fipa00079/XC00079B.html.

[60] K. Mahalingam and M.N. Huhns, "An ontology tool for query formulation in an agent-based context," in *Proc. 2nd Int. Conf. on Cooperative Information Systems (CoopIS '97)*, Kiawah Island, USA, 1997, pp. 170-178.

[61] N. Guarino, "Formal Ontology in Information Systems," in *Proc. 1st Int. Conf. on Formal Ontology in Information Systems*, Trento, Italy, 1998, pp. 3-15.

[62] J. Gamper, W. Nejdl, and M. Wolpers, "Combining Ontologies and Terminologies in Information Systems," in *Proc. 5th Int. Congress on Terminology and Knowledge Engineering*, Innsbruck, Austria, 1999, pp. 152-168.

[63] D. Fensel, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin: Springer-Verlag, 2001.

[64] O. Corcho, M. Fernandez-Lopez and A. Gomez-Perez, "Methodologies, tools and languages for building ontologies. Where is their meeting point?," *J. Data and Knowledge Engineering,* vol. 46, pp. 41-64, 2003.

[65] M. Fernandez Lopez, "Overview of Methodologies for Building Ontologies," in *Proc. IJCAI99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends*, Stockholm, Sweden, 1999, pp. 1-13.

[66] M. Cristani and R Cuel, "A survey on Ontology Creation Methodologies," *International Journal on semantic Web & Information Systems*, vol. 1, no. 2, pp. 49-69, 2005

[67] M. Harrison. Technical Report, (2003, October). EPC Information Service - Data Model and Queries. Auto-ID Centre Institute of Manufacturing, University of Cambridge, UK.

[68] Savi Technologies, Inc. (2005). "Continuous Cargo Management System Links RFID, GPS, Satellite." *Food & Beverage Industry E-Newsletter.* [Online]. Available: http://www.foodprocessing.com/vendors/products/2005/28.html

[69] B. Bacheldor, "Active Tag Include Active RFID, GPS, Satellite and Sensors," RFID *Journal*, Feb 24, 2009.

[70] P. P-S. Chen, "The Entity-Relationship Model –Toward a Unified View of Data," *ACM Transaction on Database Systems*, vol. 1 no. 1, pp. 9-36, March 1976.

[71] S. Jensen and R. T. Snodgrass, "Semantics of Time-Varying Information," *Journal of Information Systems*, vol. 21, pp.311-352, 1996.

[72] R. T. Snodgrass and I. Ahn, "Temporal Databases," *J. IEEE Computer*, vol. 19, no. 9, pp. 35-42, Sept, 1986.

[73] S. Ferg, "Modelling the Time Dimension in an Entity-Relationship Diagram," in Proc. *4th Int. Conf. on the Entity-Relationship Approach*, 1985, pp. 280-286.

[74] N. Ahmed and U. Ramachandran, "RFID middleware systems: a comparative analysis," in *Unique Radio Innovation for the 21st Century: Building Scalable and Global RFID Networks,* part 3, pp.257-278, Springer, Berlin, 2010.

[75] J. Al-Jaroodi, J. Aziz, and N. Mohamed, "Middleware for RFID Systems: An Overview," in *Proc. 33rd Annual Int. Conf. on Computer Software Applications*, 2009, pp. 154-159.

[76] A. Chella, M. Cosentinno, and L. Sabutucci, "Tools and patterns in designing multi-agent systems with PASSI," *WSEAS Transactions on Communications*, vol. 3, no.1, pp.352-358, 2004.

[77] M. Jackson, *Problem Frames: Analyzing and structuring software development problems*. Addison Wesley, 2001.

[78] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969

[79] *FIPA Interaction Protocol Library Specification*, Doc. No. XC00025E, 2001. Available: http://www.fipa.org/specs/fipa00025/

[80] J. Davies, R. Studer and P. Warren, *Semantic Web Technologies Trends and research in ontology-based systems*, John Wiley & Sons Inc., West Sussex, England, 2006.

[81] Y. Ding, and D. Fensel, "Ontology Library Systems: The key to successful Ontology Re-use", in *Proc. 1st Int. Semantic Web Working Symposium (SWWS'01)*, California, USA, 2001, pp. 93-112.

[82] DAML Ontology Library. Internet: http://www.daml.org/ontologies

[83] Schema Web. Internet: http://www.schemaweb.info/default.aspx

[84] Protégé Ontology Library. Internet: http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library

[85] Swoogle Semantic Web Search. Internet: http://swoogle.umbc.edu/

[86] OntoSelect Ontology Library. Internet: http://olp.dfki.de/ontoselect

[87] Ontaria Easy Access to Semantic Web. Internet: http://www.w3.org/2004/ontaria/

[88] Open Mobile Alliance - User Agent Profile. Internet: http://www.openmobilealliance.org/Technical/release_program/uap_v2_0.aspx

[89] Wireless Universal Resource File. Internet: http://wurfl.sourceforge.net/

[90] Device Ontology. Internet: http://www.schemaweb.info/schema/SchemaDetails.aspx?id=185

[91] *FIPA Device Ontology Specification*, Doc. No. PC00091A, 2001. Available: http://www.fipa.org/specs/fipa00091/PC00091A.html

[92] OntoViz. Internet: http://protegewiki.stanford.edu/wiki/OntoViz

[93] F. L. Bellifemine, G. Caire and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology, 2007.

[94] *FIPA SL content languages specification*, Doc. No. SC00008I, 2002. Available: http://www.fipa.org/specs/fipa00008/.

[95] S. W. Ambler, *The Object Primer*, 3rd ed., Cambridge University Press, 2004.

[96] The Java EE6 Tutorial - Part VI Persistence. Oracle PartNo: 821–1841–11, November 2010

[97] J. Chamberlain, *IBM WebSphere RFID Handbook: A Solution Guide*, 1st ed. IBM Redbooks, May 2006.

[98] D. D. Deavours. RFID Alliance Lab. Available: http://www.rfidalliancelab.org, 2004.

[99] D. D. Deavours, "A performance analysis of commercially available UHF RFID tags based on EPCglobal's class 0 and class 1 specification. Report 1," RFID Alliance Lab, Lawrence, KS, 2004.

[100] D. D Deavours, "UHF EPC tag performance evaluation. Report 2," RFID Alliance Lab, Lawrence, KS, 2005.

[101] D. M. Dobkin, *The RF in RFID, Passive UHF RFID in Practice*. Elsevier: Newnes, 2008

[102] Symbol Technologies. "Two RF Inputs make a better RFID Tag," White paper, Jan. 2005.

[103] Alien Technology, "Alien ALN-9640 Squiggle Inlay data sheet," Jan, 2012.

[104] Avery Dennison, "UHF RFID Inlay, Product Detail: AD-833," 2012.

[105] *EPC Radio Frequency Identification protocols class-1 generation-2 UHF RFID protocol for communications at 860 MHz–960 MHz*, version 1.2.0, 2008.

[106] M. Boli´c, A. Athalye, and T. Hao Li, "Performance of Passive UHF RFID Systems in Practice," in *RFID Systems: Research Trends and Challenges*, M. Boli'c, D. Simplot-Ryl and I. Stojmenovi´c, Eds., John Wiley & Sons Ltd, 2010.

[107] M. Buettner and D. Wetherall, "An empirical study of UHF RFID performance," in *Proc. 14th ACM Int. Conf. on Mobile Computing and Networking MobiCom'08*, San Francisco, California, USA, 2008, pp. 223-234.

[108] S. Aroor and D. Deavours, "Evaluation of the state of passive UHF RFID: An experimental approach," *IEEE Systems Journal*, vol. 1, pp.168–176, 2007.

[109] Y. Kawakita and J. Mitsugi, "Anti-collision performance of Gen2 Air Protocol in Random error Communication Link," in *Proc. Int. Symposium on Applications and Internet Workshops (SAINT'06)*, 2006, pp.68-71.

[110] P. Darcy, B. Stantic and A. Sattar, “A Fusion of Data Analysis and Non-Monotonic Reasoning to Restore Missed RFID Readings,” in *Proc. Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2009)*, 2009, pp. 313-318.

[111] M. S. Trotter and G. D. Durgin, "Survey of Range Improvement of Commercial RFID Tags with Power Optimized Waveforms," in *IEEE Int. Conf. on RFID, April 2010*, pp. 195-202.

[112] A. Rahmati, L. Zhong, M. Hiltunen, and R. Jana, “Reliability Techniques for RFID-Based Object Tracking Applications,” in *Proc. 37th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN ’07). IEEE Computer Society*, 2007, pp.113–118.

[113] H. Chen, W. Ku, H. Wang and M. Sun, “ Leveraging spatio-temporal redundancy for RFID data cleansing,” in *Proc. Int. Conf. on Management of Data, SIGMOD '10*, Indianapolis, Indiana, USA, 2010, pp. 51-62.

[114] H. Gonzalez, J. Han, and X. Shen, “Cost-conscious Cleaning of Massive RFID Data Sets,” in *Proc. 2007 Int. Conf. on Data Engineering (ICDE'07)*, Istanbul, Turkey, 2007, pp. 1268-1272.

[115] B. Song, P. Qin, H. Wang, W. Xuan, G. Yu, “bSpace: A Data Cleaning Approach for RFID Data Streams Based on Virtual Spatial Granularity,” in *Proc. 9th Int. Conf. on Hybrid Intelligent Systems (HIS’09)*, vol. 3, 2009, pp.252~256.

[116] J. Rao, S. Doraiswamy, H. Thakkar and L. S. Colby, “A Deferred Cleansing Method for RFID Data Analytics,” in *Proc. Int. Conf. on VLDB’06*, Seoul, Korea, 2006, pp. 175-186.

[117] A. Gupta and M. Srivastava, “Developing Auto-ID Solutions using Sun Java System RFID Software,” Sun Microsystems, White paper, Oct 2004.

[118] C. Floerkemeier and M. Lampe, “Issues with RFID usage in ubiquitous computing applications,” in *Proc. 2nd Int. Conf. on Pervasive Computing*, Linz/Vienna, Austria, 2004, pp.188-193.

[119] S. L. Lohr, *Sampling: Design and analysis.* New York: Duxbury Press, 1999.

[120] J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul and S. Zdonik, “Aurora: a new model and architecture for data stream management,” *VLDB Journal*, vol. 2, no. 2, pp. 120-139, 2003