ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

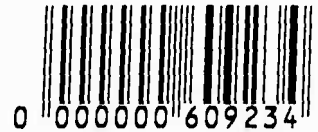
ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Αξιολόγηση και υλοποίηση RFID τεχνολογιών και προτύπων στο πλαίσιο διαχείρισης της εφοδιαστικής αλυσίδας»

Λεοντιάδης Νεκτάριος

M305005

ΑΘΗΝΑ, Φεβρουάριος 2007



ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Αξιολόγηση και υλοποίηση RFID τεχνολογιών και προτύπων στο πλαίσιο διαχείρισης της εφοδιαστικής αλυσίδας»

Λεοντιάδης Νεκτάριος

M305005

Επιβλέπων Καθηγητής: Λέκτορας Α. Πραματάρη
Εξωτερικός Κριτής: Λέκτορας Δ. Χατζηαντωνίου

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΘΗΝΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2007

Abstract

RFID is a powerful new technology with many possible applications. Whenever and wherever applied, it has the ability to extend the capabilities of existing applications in ways never imagined before.

This thesis starts by presenting the RFID technology, its core components, and the global standards that govern its applications and continues by introducing this technology in the field of supply chain management. It discusses certain issues in supply chain management that are to be resolved in the SMART project (FP6-IST-2005). It evaluates commercial RFID solutions in the context of supply chain management and, finally, it concludes with an implementation that handles the RFID need of the SMART project.

Table of Contents

1. Introduction

1.1 Research Context

The advent of e-business has created several challenges and opportunities in the supply chain environment. The Internet has made it easier to share information among supply chain partners and the current trend is to try to leverage the benefits obtained through information sharing across the supply chain to improve operational performance, customer service, and solution development. Furthermore, the emergence of new technologies, such as Radio Frequency Identification, is expected to revolutionize many of the supply chain operations by reducing costs, improving service levels and offering new capabilities for identifying unique product instances. The expected benefits from the use of both the Internet and new information technologies are to grow substantially if their scope of implementation is extended from internal warehouse and distribution processes to supply-chain processes involving collaborating partners. At the same time there is a clear turn and focus on the customer, on increasing consumer value and ultimately on building consumer enthusiasm.

The above trends represent major forces that are expected to revolutionize the supply chains of the future. In this context, is should be generally recognized that exploiting the possibility of unique identification of product instances in supply chain management processes through intelligent and innovative collaborative information systems and electronic services, and in supporting in-store processes to inform educated customers and build consumer loyalty, would be very beneficial.

Currently, automatic product identification implementations take place internally mainly within a company, with the objective to automate warehouse management processes in the first run. The priority and effort placed behind such implementations by the US Department of Defense and global retailers such as Wal-Mart, METRO, TESCO etc. combined with the pressure they put on their suppliers indicate that this technology has already become a market mandate. The Electronic Product Code, which is a global-standards-based implementation of the RFID technology (see www.gci-

net.org and www.epcglobalinc.com), is the standard adopted in all these initiatives. The EPC can be viewed as a continuation of barcode scanning, though EPC makes a significant step forward with the ability to support mass serialized identification.

In this context, the target of this thesis is to study the RFID technology as it currently stands through the EPCglobal RFID standards and then support the SMART research project with a solution which will address some of the project's requirements.
The SMART project utilizes the RFID technology in order to get information about objects that exist within the supply chain, objects that are characterized by their high mobility and their existence in variable environmental conditions. In this context, the thesis's specific targets are to explore the protocols, RFID equipment and existent middlewares that provide extra functionality, as we are going to see later on, and finally provide recommendation in these fields, for the project's needs.

1.2 Work approach

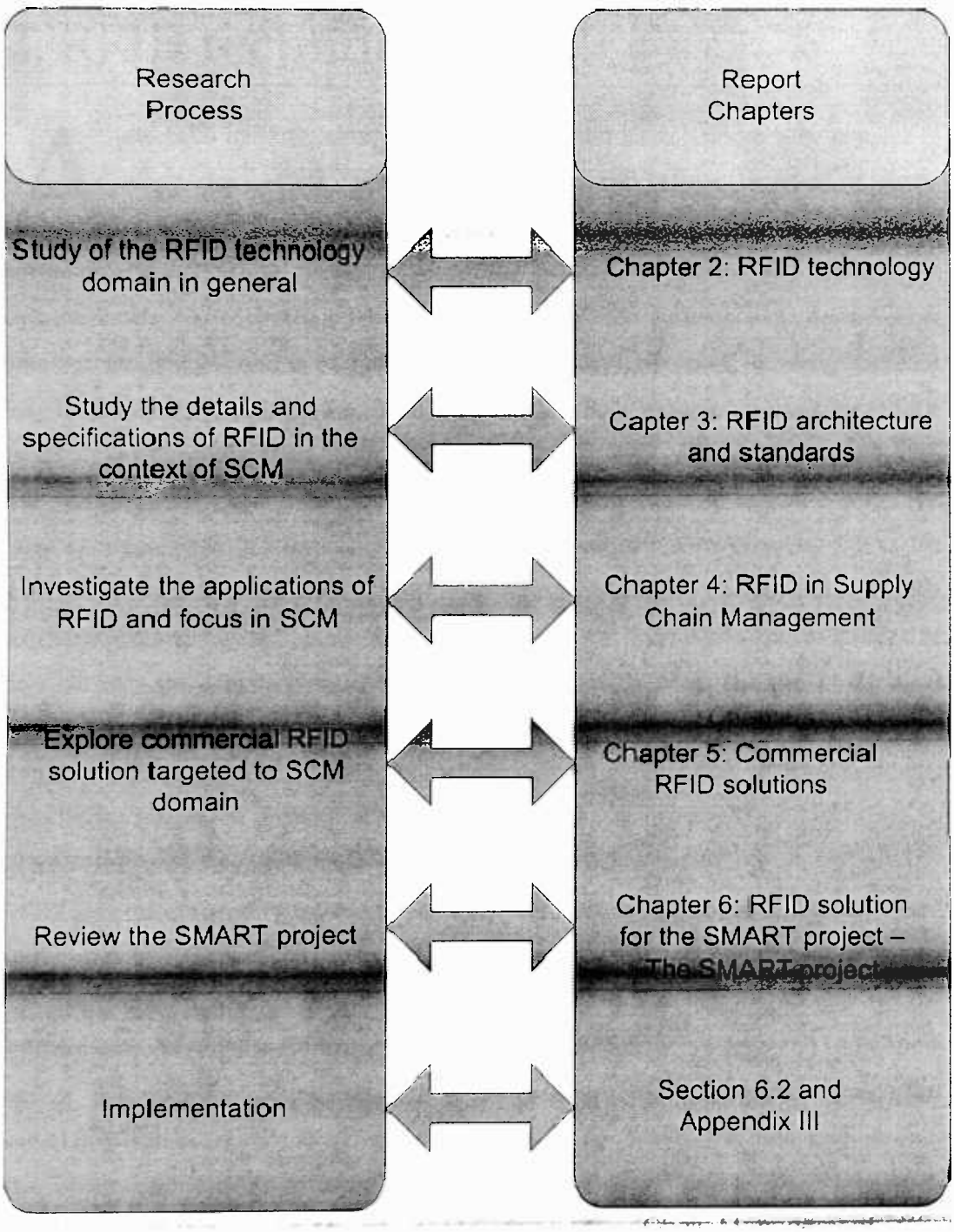
In this thesis, as it has been clearly depicted in this report, we first examined the RFID technology from a technological point of view, so that we could have a clear and confident understanding of this technology; understand its capabilities and its drawbacks, thus rendering ourselves capable of having a judicial opinion on the current trends and applications of the RFID technology. This understanding derives from the pondering of the EPCglobal standards that reign the field of RFID in the field of supply chain management. Then we studied specific matters in the field of supply chain management that can be addressed with the introduction of this technology that enables unique identification of products.
We then explored RFID solutions from major software vendors that target the field of supply chain management (SCM). The target of this phase was to evaluate these solutions against out needs within the SMART project. These solutions were tested as thoroughly as possible. The products available for evaluation purposes were installed and evaluated through their working interface, while the products which were not

available for evaluation were evaluated through their manufacturer's release information publicly available.

Finally, we proceeded with the design and implementation of a solution that can be used as a central point of building solutions that are capable of addressing issues in the field of supply chain management. Towards this objective, we contributed in an open source project named Accada Project. The Accada project, as it will be clearer later on, contributes in the field of RFID technology, by providing implementation for RFID standards which are published by EPCglobal. When the Accada project achieves a more mature state, it will provide a complete standards-based framework for building applications targeted in the field of SCM. Our contribution is part of the Reader Module of the Accada Project and this contribution will be discussed in further detail in chapter 6.

1.3 Structure of the thesis

This report is divided into three parts. In the first part, we examine thoroughly the RFID technology with the architecture and standards that are defined in the scope of supply chain management. In the second part, we examine the field of supply chain management, and the applications of RFID technology in SCM. Finally we discuss the SMART research project which tries to address collaboration issues in supply chain management with the use of RFID technology. In this part we also discuss the specific implementation that was carried out. In the following figure we visualize the association between the research process we followed during the thesis and the chapters of this report.



2. RFID technology

Automatic identification procedures (Auto-ID) have become very popular in many service industries, purchasing and distribution logistics, industry, manufacturing companies and material flow systems. They exist to provide information about people, animals, goods and products in transit. The omnipresent barcode labels that triggered a revolution in identification systems some considerable time ago are being found to be inadequate in an increasing number of cases. Barcodes may be extremely cheap, but their stumbling block is their low storage capacity and the fact that they cannot be reprogrammed.

The technically optimal solution would be the storage of data in a silicon chip. The most common form of electronic data-carrying device in use in everyday life is the smart card based upon a contact field (telephone smart card, bank cards). However, the mechanical contact used in the smart card is often impractical. A contactless transfer of data between the data-carrying device and its reader is far more flexible. In the ideal case, the power required to operate the electronic data-carrying device would also be transferred from the reader using contactless technology. Because of the procedures used for the transfer of power and data, contactless ID systems are called RFID systems (Radio Frequency Identification).

RFID systems are closely related to the smart cards described above. Like smart card systems, data is stored on an electronic data-carrying device — the transponder. However, unlike the smart card, the power supply to the data-carrying device and the data exchange between the data-carrying device and the reader are achieved without the use of galvanic contacts, using instead magnetic or electromagnetic fields. The underlying technical procedure is drawn from the fields of radio and radar engineering. The abbreviation RFID stands for radio frequency identification, i.e. information carried by radio waves. Due to the numerous advantages of RFID systems compared with other identification systems, RFID systems are now beginning to conquer new mass markets. One example is the use of contactless smart cards as tickets for short-distance public transport.

The EPC Network of enabling technologies started development at the Auto-ID Center

at the Massachusetts Institute of Technology and now is being commercialized under the leadership of EPCglobal. These technologies are the leading RFID-based solutions for global, open supply chain applications. The roster of companies committed to using EPC standards and mandating that their suppliers support EPC specifications is growing rapidly and currently includes the U.S. Department of Defense (DoD), with more than 43,000 suppliers; Wal-Mart, with more than 20,000 suppliers; The METRO Group; Albertson's; Target Corporation; Best Buy; Tesco; and others.

The EPC Network is comprised of five fundamental technologies:

- **Electronic Product Code**

  Like the Universal Product Code (UPC) bar code, the EPC identifies the manufacturer and product version, but has an additional serial number field to identify uniquely each identical item.

- **EPC ID System** (tags, printer/encoders, and readers)

  An EPC tag consists of a microchip attached to an antenna. The EPC data is typically stored in the EPC tag while embedded in a "smart" label being printed on an EPC printer/encoder. The label incorporating the tag is applied to an item during the manufacturing process. EPC tags communicate their data to EPC readers via radio waves and deliver information to local business information systems using EPC middleware.

- **EPC Middleware**

  This software specification for services enables data exchange between an EPC reader, or network of readers, and business information systems.

- **Object Name Service (ONS)**

  Business information systems need a way of matching the EPC to the database information about that item. The ONS is an automated networking service that provides this service by linking computers to sites on the World Wide Web. It is based on the popular and globally accepted DNS protocol.

- **EPC Information Systems (EPCIS)**

  EPC Information Systems enable users to exchange data with trading partners based on EPCs.

These technologies are going to be discussed in further detail in chapter 3, RFID Architectures and Standards.

The EPC Network grew out of perspectives that only open RFID systems and standards — paralleling the open EAN.UCC system of bar code standards — would support a global supply chain. RFID systems enable objects to "report" information about them in real time without human intervention and can give a company visibility into real-time information about inventory locations, histories and quantities. The goal of the EPC Network is to let companies leverage their internal RFID structures to gain exponentially by capturing and sharing real-time business information across entire companies and with trading partners. In other words, RFID deployments based on open, standardized EPC interfaces that enable interoperability and multi-vendor implementations may achieve the ultimate in global data sharing: Total asset visibility. That is why so many leading companies are supporting and driving the adoption of EPCglobal standards for the EPC Network.

The scale of this undertaking is huge, unlike anything attempted with RFID technologies before. However, many leading consumer packaged goods (CPG) companies and technology vendors are no longer saying "if" but "when" the total EPC Network vision becomes a reality. A great deal of investment already has been made in RFID pilots. As with all emerging technologies and grand visions, the reality is that initial discovery phases lead to new conclusions and results that modify original thinking and — in the case of RFID — raised expectations for this technology.

In effect, this is what occurred with the UHF Generation 1 (Gen1) specifications. The Class 0 and Class 1 protocols that came out early in the cycle when the Wal-Mart and DoD mandates were issued for case and pallet auto-identification. What was needed was a methodology that would allow readers to communicate to low-cost tags that would carry only the EPC "license plate" number. The EPC would tie back through ONS to databases that could hold an infinite amount of dynamic data about each item.

Gen1 UHF RFID technology stressed low cost and simplicity. The idea was to limit the EPC number to a "license plate" that pointed to the product-related data associated with

each item stored on databases held in back-office systems. In addition, the Gen 1 protocols provided guidelines for the operating characteristics of readers and tags, covering frequencies, emissions standards, anti-collision methods (to address contention where multiple tags are seen by the reader simultaneously), and secure data transmission practices.

Similar to the UPC code that provides for manufacture, product, version, and serial data, the EPC data structure was envisioned to contain:

- A header that identifies the EPC version number,
- The EPC manager, which typically would be the manufacturer's name,
- The object class or version information, which essentially is the product information,
- A serial number, which would be information specific to an item in an object class.

Originally both 64-bit and 96-bit EPC data structures were proposed. Since many object classes and serial numbers were not needed initially, a family of 64-bit data structures was proposed to keep down the price of Gen 1 RFID chips. The more robust 96-bit EPC provides unique identifiers for 268 million companies, each with up to 16 million object classes, with 68 billion available serial numbers in each object class. The Gen 1 Class 0 and Class 1 tags were specified by EPCglobal to contain the EPC data structure, a cyclic redundancy check (CRC) to verify the tag data, and a kill or destruct code that would deactivate the tag and no longer allow it to respond to reader commands. As originally specified by EPCglobal, Class 0 were read-only tags. Class 1 were specified as write once/read many tags, so in addition to the other functions, Class 1 needed a "lock" command to prevent any further modification of the tag information once written. Both classes had a "kill" function to ensure user privacy after product sale. No other data or tag functionality was considered as part of the Class 0 and Class 1 specifications, however important variations did occur.

As with all new technology, Gen 1 products rapidly evolved beyond the original specifications. There are now 96-bit versions of both Class 0 and Class 1 in use. There

are also versions that extend the Class 0 and Class 1 specifications published by EPCglobal:

- For example, while Class 1 specifies one-time-programmability of the EPC data field, in practice "Class 1 compliant" tags may be read/write.
- There are also "Class 0+" products that are one-time field programmable but are read using the specified Class 0 air interface protocol.

Gen 2 was developed in response to shortcomings of the Gen 1 technology that were discovered during pilot projects with early adopters. These early adopters soon found that it was more practical to leverage their significant investments in RFID to begin to transform their internal processes to be more productive. It became apparent that tags that could be read to and written to many times were more effective for many of their operations, such as pallet and case reshipment in distribution centers and third-party logistics operations. In addition, it made sense in many applications to allow the tags to carry more data than just a 64-bit EPC number. In effect, the tags themselves became mini databases of important information. They could integrate directly with their existing bar code-based IT infrastructure without any need to reference a back-office system. Also, the Gen 2 RF transmission protocol specification was specifically designed to provide robust, international operation under new European and Asian UHF radio regulations for RFID that were changed after Gen 1 products had been designed.
The Gen 2 specification stipulates more EPC memory on the chip, a minimum of 96 bits and up to 256 bits. Most of the organizations that have committed to Gen 2 technology have issued supplier requirements that specify the use of 96-bit identifiers. Therefore, suppliers and other participants in these supply chains must have an IT infrastructure capable of processing 96-bit data to produce and read Gen 2 tags. Differences between Gen 1 and Gen 2 technology are summarized in the following figure.

	Generation 1	Generation 2
Frequency	860MHz - 930MHz	860MHz - 930MHz
Field-programmable	64 or 96 bits	96 to 256 bits
Re-programmable (read/write)	Yes	Yes
Field-kill-able	Class 0 - Specified as read only Class 1 - Specified as Write once/Read many	Yes

Figure:Gen1 vs. Gen2

There are other differences among the Gen1 and Gen2 RF transmission protocols that for the most part are transparent to users and are unlikely to impact implementation planning. The complete specifications are available on the EPCglobal Web site (http://www.epcglobalinc.org). It is also worth noting that EPCglobal performed some interoperability testing to provide guidance to the market on the essential functions of "EPC compliant" products. For example, tags made by "Manufacturer X", are compliant with the Class 1 specification, and can be read using a reader made by "Manufacturer Y" when encoded by "Printer/Encoder Z."

3. RFID architecture and standards

In November 2003, the Global Commerce Initiative (GCI) published the "The EPC Roadmap", a report that outlined the combined technology and process initiatives that have the potential to revolutionize the consumer products/retail industry. In this report, the GCI Executive Board strongly recommended the global standards based implementation of radio frequency identification technology, supported by the use of standards-based tags, readers, tag content and information flows in the retail supply chain. In 2004, the first set of global standards were developed and established via EPCglobal, a worldwide, user-driven standards organization for the Electronic Product Code.

EPCglobal Inc. is a joint venture between EAN Intl. (European Article Numbering) and the UCC (Universal Code Council) and is governed by the EPCglobal Board of Governors. The Board of Governors of EPCglobal is obliged to guide the organization towards achieving worldwide adoption and standardization of EPC technology in an ethical and responsible way. It is a subscriber-driven organization comprised of industry leaders and organizations focused on creating global standards for the EPCglobal Network. Its goal is increased visibility and efficiency throughout the supply chain and higher quality information flow between companies and their key trading partners.

In few words EPCglobal is the de facto organization that is responsible for the standardization process of the RFID technology. As a consequence we will examine now the standards published by EPCglobal regarding the RFID technology and the proposed architecture and architectural elements. Standards and specifications provide the common definitions, functionality and language for the hardware and software components of the EPCglobal Network. They help advance the EPCglobal community toward a common objective, namely, implementing the EPCglobal Network to improve visibility and efficiency in today's global, multi-industry supply chain. EPCglobal Specifications result from the work that began under the auspices of the Auto-ID Center at MIT and form the foundation for the EPC/RFID technology that the EPCglobal community has begun implementing worldwide.

3.1 Class 1 Generation 2 UHF Air Interface Protocol Standard

The Generation 2 specification is a 94-page engineering document entitled "EPC™ Radio-Frequency Identity Protocols / Class-1 Generation-2 UHF RFID / Protocol for Communications at 860 MHz – 960 MHz". It describes in considerable detail how Generation2 RFID tags should communicate with RFID readers (1).

Some key points of the specification are:

- Tags must be able to communicate on any frequency between 860MHz to 960MHz. To account for variations in regional radio regulations, readers may operate using any permitted frequency within that range.
- Tags must be able to understand three different modulation schemes: Double Sideband-Amplitude Shift Keying (DB-ASK), Single Sideband-Amplitude Shift Keying (SS-ASK) and Phase-Reversal Amplitude Shift Keying (PR-ASK). Readers will determine which modulation scheme is used, within the context of government radio regulations.
- Tags must be able to transmit at several different speeds or data rates: 80kbits, 160kbits, 320kbit and 640kbits. Readers determine what speed to use. For perspective, Generation1 protocols communicated at between 70 and 149kbits. In theory, the higher data rates of Generation 2 will allow faster tag reads, but in practice many other factors beyond raw transmission speed contribute to real world read rates, and the higher data rates may sometimes result in lower reliability.
- Generation 2 tags support Electronic Product Codes up to 256 bits long, whereas Generation 1 tags support Electronic Product Codes up to 96 bits long.
- Generation 2 includes a method to support 'dense-interrogator channelized signaling' (sometimes called 'dense reader mode') which is an attempt to reduce interference between EPC readers, by making it less likely that reader signals will be drowning out tag responses. This mode is intended for use in locations where multiple readers are in operation at the same time. Implementation of this part of the specification is optional for reader makers. Again, real world

performance will depend on many factors, including external interference from other devices, such as UHF cordless telephones, industrial equipment and legacy UHF wireless LAN equipment.

The most obvious conclusion to draw from the Generation 2 specification document is that Generation 2 is something of a 'Chinese Menu' for RFID. The multiple options for modulation, for example, provide a number of different ways to do the same thing, which is to receive data from tags. The specification also provides for many optional commands, as well as vendor-specific custom commands. To be fully compliant with the specification, a tag must offer the full menu. The reader, on the other hand, can pick and choose among these options by, for example, determining the modulation scheme and data rate the tag should use in the context of a particular communication attempt. For a variety of reasons, including achieving good performance in a wide variety of end user applications and to function effectively in the presence of external interference, the reader is likely to dynamically choose among these many options to fit each particular circumstance. This is why EPC Generation 2 is called a 'multi-protocol protocol'.

EPC Generation 2 has a much higher potential for variation than any other RFID tag. Its 'Chinese Menu' approach to the underlying communication protocol is one reason for this, but there are other reasons too. For example, different tag vendors will understand the specification in slightly different ways. Some vendors may implement only some of the options due to schedule or cost constraints. Some will add proprietary 'extensions'. Some tags will have two antennas, while others will only have one antenna. Even among tags made by the same vendor, variations will be introduced over time, by design changes for cost or manufacturability reasons or attempts to deliver better performance.

While compliance tests and adherence to the specification will deliver a basic degree of interoperability between different 'Generation 2' flavors, RFID readers may need to address each variation differently to achieve optimal performance from each one, and among mixed populations of Generation 2 tags from different vendors. The unprecedented number of tag vendors competing in the Generation 2 market means that

a declaration of 'Generation 2' compliance may not, in itself, be an assurance of optimal, or even acceptable, performance from all tags.

Until Generation 2, the RFID industry has never seen this degree of support for a single platform before, and true interoperability will require a subtle, nuanced approach to RFID infrastructure. The potential for broad variation means missteps in purchasing RFID readers could cause significant and unforeseen problems for system integrators and end users.

3.1.1 Special Chip-Level Features: Higher Bit Rate

One new feature of Generation 2 is a higher speed or data transmission (or 'bit rate') than in Generation 1 tags. Generation 2 tags will be capable of sending data and responding to commands at speeds up to 8 times faster than Generation 1 tags. In theory, this higher bit rate will lead to a greater number of tags being read per second. The increase in real-world tags read per second will not keep pace with the increase bit rate, however, as higher speeds necessarily introduce higher error rates and greater noise susceptibility, and longer EPC codes (up to 256 bits) will also consume some of the extra communication capacity.

One analogy that for this situation is that buying a faster car does not guarantee your commute will be quicker – total commuting time is strongly dependent on externalities like traffic and road conditions. The same is true of tag data rate versus tag read rate. It is likely that Generation 2 will deliver faster tags reads than Generation 1, in many situations.

But the real-world performance increase will not be directly proportional to the increased data rate provided by Generation 2, and in some situations the best course of action may be for the reader to fall back on a lower data rate mode for communication reliability reasons.

3.1.2 *Special Chip-Level Features: 'Dense Reader Mode'*

Another new feature of Generation 2 is 'dense reader mode'. Dense reader mode is intended to help avoid reader interference with tag responses, by reserving certain sub-parts of the government allocated radio spectrum for tags to use. As tags are much weaker communicators than readers, this has some benefit in helping tags to work in situations where multiple readers are operating in close proximity.

However, 'dense reader mode' is not a panacea for every problem that can arise when many readers are operating in close quarters. There are several important aspects to interference, not all of which are addressed by 'dense reader mode'. These include:

- Other devices besides RFID readers may be operating in or around the UHF spectrum. These devices are free to hop into channels being used by tags, and can drown them out. In the US, for example, unlicensed FCC Part 15 devices, such as RFID readers, cordless telephones, wireless LAN equipment, industrial equipment, etc, are permitted the same priority in UHF spectrum use. The presence of even one non-RFID device in an end-user facility may partially or completely negate the potential benefits of dense-reader mode, because non-RFID devices will most likely not adhere to the channelization scheme required for dense-reader mode to be effective.
- Not all Generation 2 RFID readers must support 'dense reader mode. It is an optional part of the Generation 2 specification. Even a single non-dense-reader mode RFID reader, such as a handheld reader, a UHF active tag system, or a legacy proprietary RFID system, may also drown out tag responses and negate the potential benefits of dense-reader mode.
- Passive RFID tags, such as Generation 2 tags, are broadband backscatter devices.

  As such, when they respond to one reader's interrogation on a certain frequency, they are actually modulating a response across all frequencies simultaneously. This backscatter modulation can be picked up by any reader within view of the responding tag, which can lead to interference even at much longer distances

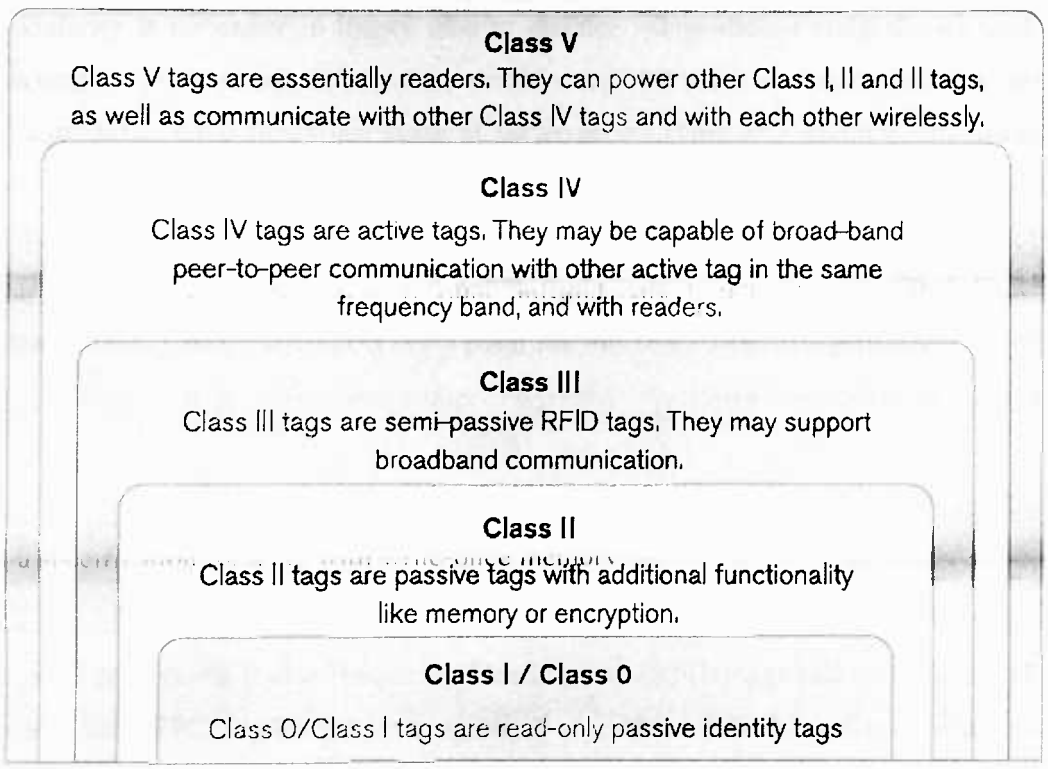
than those at which the tag can be effectively read. Thus, all tags being read by a given reader are potential interference generators to all nearby readers. This effect will be particularly pronounced in environments such as long rows of warehouse loading dock doors in which large numbers of tags are being interrogated by large numbers of readers, and all readers and tags are operating in close proximity.

- Most importantly, as several papers by Auto-ID Labs' Director Dr. Daniel Engels have pointed out, in addition to being broadband backscatter 'transmitters', tags are also 'broadband receivers' – tags have no knowledge of reader channels. The receiver on a tag is akin to a crystal radio set, in that its frequency selectivity is limited only by the properties of the tag's antenna. This is in fact enforced by the Generation 2 specification, because the Generation 2 specification requires tags to operate from 860-960MHz. Thus, a given tag will receive all transmissions being sent at the same time by all readers in view, even if the readers are operating in 'dense reader mode' and are therefore using different channels. This is a fundamental problem with contemporary RFID, and is best solved by precise, time-based reader synchronization that aims to prevent two readers trying to talk to the same tag at the same time. Because of this, time-based reader synchronization is the most important strategy for managing dense reader environments. Unfortunately, the Generation 2 'dense reader mode' does not address this important shortcoming of the broadband receiver on Generation 2 tags.
- 'Dense reader mode' should therefore be thought of as a first step toward operation in dense reader environments. But it is not a cure-all for the limitations of today's passive RFID technology. The fundamental problems of dense reader environments, including the tag's broadband transmission and reception, remain to be solved in future generations of passive RFID technology.

3.1.3 The EPC Class Structure

The EPC Tag Class Structure is often misunderstood. 'Class' is not the same as 'Generation'. Class describes a tag's basic functionality – for example whether it has memory or a battery. Generation refers to a tag specification's major release or version number. The full name for what is popularly called EPC Generation 2 is actually EPC Class 1 Generation 2, indicating that the specification refers to the second major release of a specification for a tag with write-once memory.

The full EPC Class Structure is:



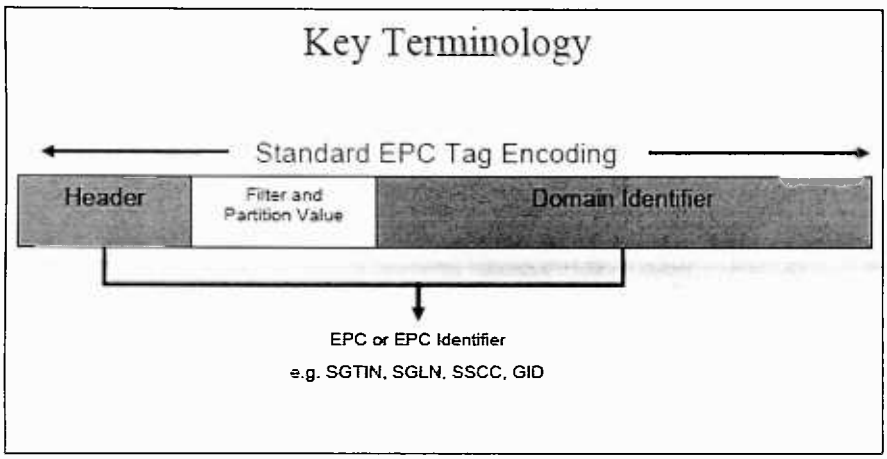
The Class 0 designation was added to the Generation 1 system long after the Class 1 specification was created. Class 1 tags, of which 'Generation 2' is an example, contain a write once memory for storing an Electronic Product Code. Class 2 tags add additional memory that can be changed frequently, for storing additional data – for example from an onboard sensor. Class 3 tags add batteries for longer read ranges and

higher reliability, but are fundamentally passive backscatter tags. Class 4 tags are essentially active tags that can communicate with other Class 4 tags as well as readers. Class 5 tags are not really tags at all - they are essentially wireless networked readers.
The intent of the EPC Class Structure is to provide a modular structure that covers a wide variety of possible types of tag functionality. For example, the communication protocol
for a battery-powered tag should be the same as the protocol for a battery-less tag, with only the addition of necessary commands to support the battery. This way, protocols are kept simple, and if the battery on a battery-powered tag fails or dies, the tag can simply behave like a battery-less tag and still have some utility to the end user. This kind of modularity is far easier in theory than in practice – nonetheless technologies tend to converge over time, and there is desire within the EPC movement to deliver the dream of a modular, multi-functional stack of tag protocols. This adds another dimension to the variety of tags that a reader infrastructure must be able to adjust to. Now that the disparity between Class 0 and Class 1 has been eliminated by Class 1 Generation 2, the next likely step is to add a battery-tag to the system. This may take the form of a Generation 2, Class 3 tag – or it could point the way to a whole new generation entirely.

3.2 EPC Tag Data Standard

The Electronic Product Code is an identification scheme for universally identifying physical objects via Radio Frequency Identification (RFID) tags and other means. The standardized EPC Tag Encodings consists of an EPC (or EPC Identifier) that uniquely identifies an individual object, as well as a Filter Value when judged to be necessary to enable effective and efficient reading of the EPC tags.
The EPC Identifier is a meta-coding scheme designed to support the needs of various industries by accommodating both existing coding schemes where possible and defining new schemes where necessary. The various coding schemes are referred to as Domain Identifiers, to indicate that they provide object identification within certain domains such as a particular industry or group of industries. As such, the Electronic

Product Code represents a family of coding schemes (or "namespaces") and a means to make them unique across all possible EPC-compliant tags. These concepts are depicted in the chart below.



This standard applies to RFID tags conforming to "EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz-960MHz Version 1.0.9" ("Gen2 Specification"). These standards define completely that portion of EPC tag data that is standardized, including how that data is encoded on the EPC tag itself (i.e. the EPC Tag Encodings), as well as how it is encoded for use in the information systems layers of the EPC Systems Network (i.e. the EPC URI or Uniform Resource Identifier Encodings).

The EPC Tag Encodings include a Header field followed by one or more Value Fields. The Header field defines the overall length and format of the Values Fields. The Value Fields contain a unique EPC Identifier and a required Filter Value when the latter is judged to be important to encode on the tag itself. The EPC URI Encodings provide the means for applications software to process EPC Tag

Encodings either literally (i.e. at the bit level) or at various levels of semantic abstraction that is independent of the tag variations. This document defines four categories of URI:

1. URIs for pure identities sometimes called "canonical forms." These contain

only the unique information that identifies a specific physical object, location or organization, and are independent of tag encodings.

2. URIs that represent specific tag encodings. These are used in software applications where the encoding scheme is relevant, as when commanding software to write a tag.
3. URIs that represent patterns, or sets of EPCs. These are used when instructing software how to filter tag data.
4. URIs that represent raw tag information, generally used only for error reporting purposes.

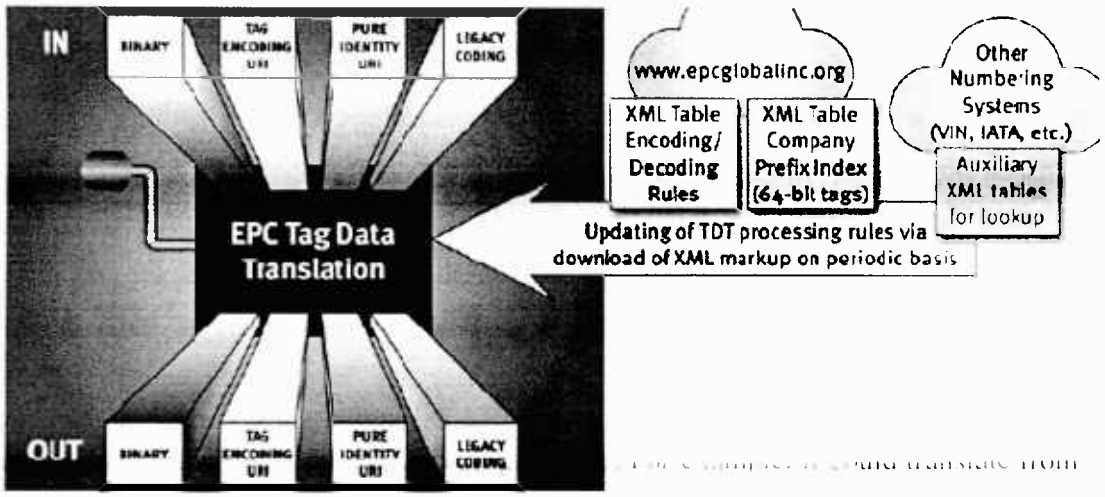
In the current version of the EPC – EPC Version 1.3 – the specific coding schemes include a General Identifier (GID), a serialized version of the EAN.UCC Global Trade Item Number (GTIN), the EAN.UCC Serial Shipping Container Code (SSCC), the EAN.UCC Global Location Number (GLN), the EAN.UCC Global Returnable Asset Identifier (GRAI), the EAN.UCC Global Individual Asset Identifier (GIAI) and the DOD Construct. (2)

3.3 EPC Tag Data Translation Standard

This EPC Tag Data Translation (TDT) specification (3) is concerned with a machine-readable version of the EPC Tag Data Standards specification, in contrast to the Tag Data Standard which describes in terms of human readable encoding and decoding rules for each coding scheme, how to translate between three representations of the electronic product code. The machine-readable version can be readily used for validating EPC formats as well as translating between the different levels of representation in a consistent way. This specification describes how to interpret the machine-readable version. It contains details of the structure and elements of the machine-readable markup files and provides guidance on how it might be used in automatic translation or validation software, whether standalone or embedded in other systems.

The Tag Data Translation process translates one representation of EPC into another

representation, within a particular coding scheme. For example, it could translate from the binary format for a GTIN on a 64-bit tag to a pure-identity URI representation of the same identifier, although it could not translate a SSCC into a SGTIN or vice versa. The Tag Data Translation concept is illustrated in Figure 1.



Tag Data Translation capabilities may be implemented at any level of the EPC Network stack, from readers, through filtering middleware, as a pre-resolver to the Object Name Service (ONS), as well as by applications and networked databases complying with the EPCIS interface. Tag Data Translation converts between different levels of representation of the EPC and may make use of external tables, such as the Company Prefix Index lookup table for 64-bit tags. It is envisaged that Tag Data Translation software will be able to keep itself up-to-date by periodically checking for and downloading TDT markup files, although a continuous network connection should not be required for performing translations or validations, since the TDT markup files and any auxiliary tables can be cached between periodic checks; in this way a generic translation mechanism can be extensible to further coding schemes or variations for longer tag lengths, which may be introduced in the future.

The current version of the TDT specification (version 1.0) is fully compatible with TDS Version 1.1 Rev. 1.27.

3.4 Reader Protocol (RP) Standard

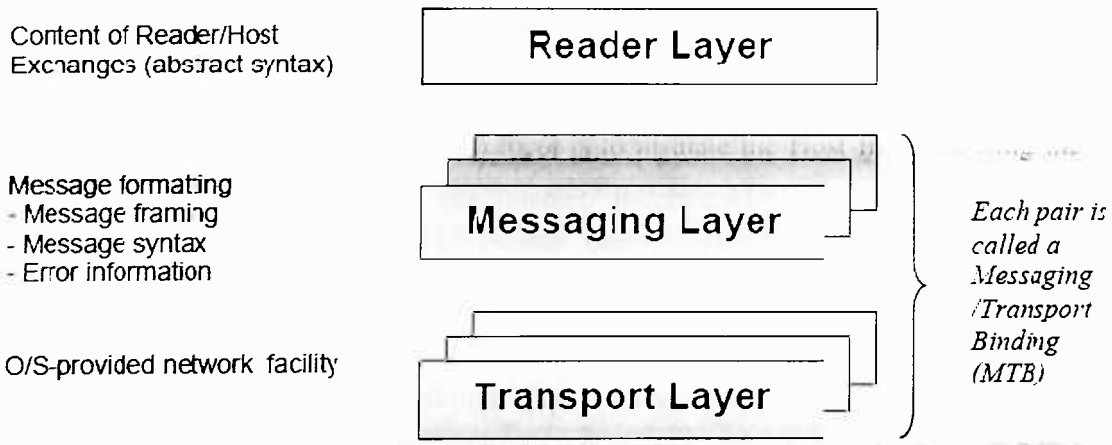
Reader Protocol (4) is an interface standard that specifies the interactions between a device capable of reading/writing tags and application software. In other words, this standard defines the way that EPCglobal compliant software applications interact with tag readers.

The Reader Protocol specifies the interaction between a device capable of reading (and possibly writing) tags, and application software. An example of a Host is an EPC-aware middleware or application, though the Reader Protocol itself does not require that any particular middleware or an application be used.

Note that the interaction between the Reader and RF tags is outside the scope of this specification. A goal of the Reader Protocol is to insulate the Host from knowing the details of how the Reader and tags interact. Readers MAY employ a variety of protocols to interact with tags (not exclusively RF tags: e.g., a Reader MAY be capable of reading optical bar codes), but the same Reader Protocol as specified in this standard is used between the Reader and Host.

The commands have been defined as SHALLs and CANs. The Reader Protocol Conformance Requirements will be developed to categorize readers by the functions they perform. Readers' profiles will be developed to help conduct fair and accurate conformance tests. It is possible that existing CANs may change to reflect the profiles. ,

The Reader Protocol is specified in three distinct layers, as illustrated below:



The layers are:

- **Reader Layer**

  This layer specifies the content and abstract syntax of messages exchanged between the Reader and Host. This layer is the heart of the Reader Protocol, defining the operations that Readers perform and what they mean.

- **Messaging layer**

  This layer specifies how messages defined in the Reader Layer are formatted, framed, transformed, and carried on a specific network transport.

- **Transport Layer**

  This layer corresponds to the networking facilities provided by the operating system or equivalent.

The Reader Protocol specification provides for multiple alternative implementations of the Messaging and Transport Layers. Each such permutation is called a Messaging/Transport Binding (MTB). Different MTBs provide for different kinds of transport, e.g., TCP/IP versus Bluetooth versus serial line. Different MTBs may also provide different means for establishing connections (e.g., whether the Reader contacts the Host or the Host contacts the Reader), initialization messages required to establish synchronization, and means for provisioning of configuration information. Multiple standard MTBs are defined in this specification. Others may be defined and specified in separate, companion specifications.

3.5 Reader Management (RM) Standard

The Reader Management Standard (5) defines Version 1.0 of the wire protocol used by management software to monitor the operating status and health of EPCglobal compliant RFID Readers. This document complements the EPCglobal Reader Protocol Version 1.1 specification. In addition, this document defines Version 1.0 of the EPCglobal SNMP RFID MIB and specifies the set of SNMP MIBII groups required to

provisioning of configuration information. Multiple

comply with this EPCglobal Reader Management Specification over SNMP. The terms "tag Reader" and "Reader" include RFID tag Readers, supporting any combination of RF protocols, fixed and hand-held, etc. It also includes Readers of other kinds of tags, such as bar codes. Tag Readers, despite the name, may also have the ability to write data into tag memory.

The Reader Management Protocol specifies the interaction between a device capable of interfacing with tags, and management software. The host may be a fully featured Management Console capable of processing SNMP messages, or a dedicated application capable of communicating with the Reader to interface with RFID tags and monitor its health.

The collection of tag data between the Reader and the Host is defined in the EPCglobal Reader Protocol Version 1.1. This document focuses on the communication protocol required to monitor the health of the Reader.

This standard defines two separate but related management protocol specifications.

1. Specifies the EPCglobal SNMP MIB for monitoring the health of a Reader.
2. Specifies the EPCglobal Reader Management Protocol for monitoring the health of a Reader.

Number 2 follows the same layering structure defined by the EPCglobal Reader Protocol Version 1.1 [RP1]. This specification defines an XML MTB which complements that defined by the Reader Protocol. The Transport details can be found in the Reader Protocol Specification 1.1. The SNMP MIB components are also defined in detail in this specification.

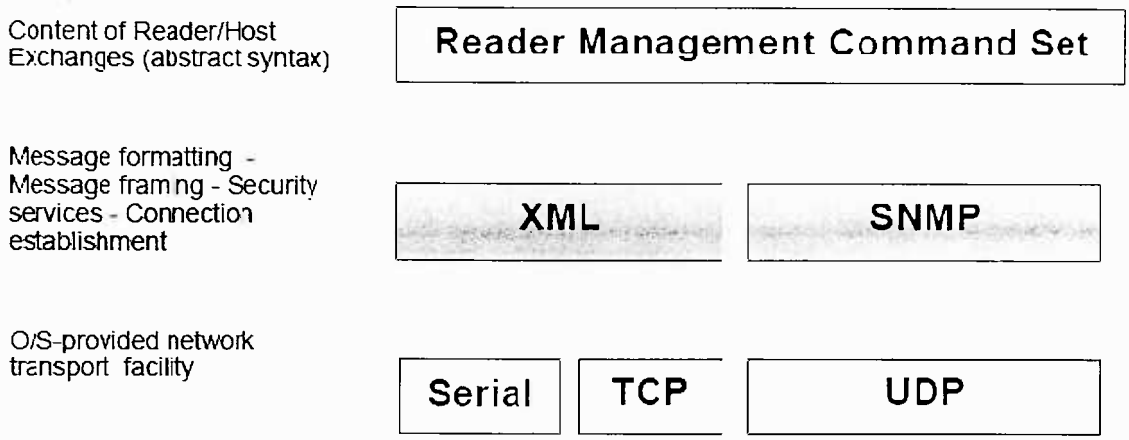


Figure: Protocol Layers Mapping

The SNMP MIB is an example of another Message/Transport binding. The MIB is a structuring and representation of Reader Object Model elements that conforms to the SNMP specification "Structure of Management Information Version 2 (SMIv2)" (6). The SNMP protocol has a well defined messaging protocol and transport layer for getting and setting information, event notification, and security facilities. The figure above depicts two MTB examples. On the left, the Reader Protocol Message Format using XML as input and output, and TCP as the transport. On the right, SNMP defining the Message formatting and UDP as the message transport. Note that the same Reader Management Command Set applies to both MTBs.

3.6 **Application Level** Events (ALE) Standard

This EPCglobal Board-ratified standard specifies an interface through which clients may obtain filtered, consolidated Electronic Product Code data from a variety of sources (7). The design of this interface recognizes that in most EPC processing systems, there is a level of processing that reduces the volume of data that comes directly from EPC data sources such as RFID readers into coarser "events" of interest to applications. It also recognizes that decoupling these applications from the physical layers of infrastructure offers cost and flexibility advantages to technology providers

and end-users alike.

The processing done at this layer typically involves: receiving EPCs from one or more data sources such as readers, accumulating data over intervals of time, filtering to eliminate duplicate EPCs and EPCs that are not of interest, counting and grouping EPCs to reduce the volume of data and reporting in various forms. The interface described in this standard and the functionality it implies, is called "Application Level Events", or ALE.

In early versions of the EPCglobal Network Architecture, originating at the Auto-ID Center at the Massachusetts Institute of Technology (MIT), these functions were understood to be part of a specific component termed "Savant." The term "Savant" has been variously used to refer generically to any software situated between RFID readers and enterprise applications, or more specifically to a particular design for such software as described by an MIT Auto-ID Center document "The Savant Specification Version 0.1" or to a later effort by the Auto-ID Center Software Action Group that outlined a generalized container framework for such software. Owing to the confusion surrounding the term, the word "Savant" has been deprecated by EPCglobal in favor of more definite specifications of particular functionality. The interface described is this standard is the first such definite specification.

The role of the ALE interface within the EPCglobal Network Architecture is to provide independence between the infrastructure components that acquire the raw EPC data, the architectural component(s) that filter & count that data, and the applications that use the data. This allows changes in one without requiring changes in the other, offering significant benefits to both the technology provider and the end-user. The ALE interface described in the specification achieves this independence through three means:

- It provides a means for clients to specify, in a high-level, declarative way, what EPC data they are interested in, without dictating an implementation. The interface is designed to give implementations the widest possible latitude in selecting strategies for carrying out client requests; such strategies may be influenced by performance goals, the native abilities of readers which may carry out certain filtering or counting operations at the level of firmware or RF protocol, and so forth.

- It provides a standardized format for reporting accumulated, filtered EPC data that is largely independent of where the EPC data originated or how it was processed.
- It abstracts the sources of EPC data into a higher-level notion of "logical reader," often synonymous with "location," hiding from clients the details of exactly what physical devices were used to gather EPC data relevant to a particular logical location. This allows changes to occur at the physical layer (for example, replacing a 2-port multi-antenna reader at a loading dock door with three "smart antenna" readers) without affecting client applications. Similarly, it abstracts away the fine grained details of how data is gathered (e.g., how many individual tag read attempts were carried out). These features of abstraction are a consequence of the way the data specification and reporting aspects of the interface are designed.

Unlike the earlier MIT "Savant Version 0.1" effort, this specification does not specify a particular implementation strategy or internal interfaces within a specific body of software. Instead, this specification focuses exclusively on one external interface, admitting a wide variety of possible implementations so long as they fulfill the contract of the interface. For example, it is possible to envision an implementation of this interface as an independent piece of software that speaks to RFID readers using their network wireline protocols. It is equally possible, however, to envision another implementation in which the software implementing the interface is part of the reader device itself.

3.7 Object Naming Service (ONS) Standard

The EPCglobal Network architecture provides a method for the inclusion of commercial (both physical and otherwise) products within a network of information services. This architecture makes several axiomatic assumptions, the most important being that it should leverage existing Internet technology and infrastructure as much as

possible. In most situations the EPC will denote some physical object. EPC identifiers are divided into groups, or namespaces. Each of these namespaces corresponds to a particular subset of items that can be identified. For example, XML Schemas are denoted using the 'XML' namespace, raw RFID tag contents are kept in the 'raw' namespace. The 'id' namespace is generally reserved for EPCs that can be encoded onto RFID tags and for which services may be looked up using ONS. This 'id' namespace is further subdivided into sub-namespaces corresponding to different naming schemes for physical objects, including Serialized GTINs, SSCCs, GLNs, etc. These namespaces are defined normatively in the EPCglobal Tag Data Standards (2).

Each of the sub-namespaces that are defined by the Tag Data Standard has a slightly different structure depending on what they identify, how they are used, and how they are assigned. The SGTIN is used to identify an individual product that is assigned by the company that creates that product. Thus the SGTIN contains an EPC Manager Number, an Object Class, and a Serial Number. Other sub-namespaces such as the SSCC go directly from the EPC Manager Number to the Serial Number and have no concept of an "Object Class". This document only specifies the ONS lookup mechanism for the SGTIN sub-namespace. As such, in many cases its statements about concepts such as "Object Class" or "Serial Number" are specific to the SGTIN namespace and should not be construed as applying to all EPC namespaces. Specifications for those other namespaces are the subject of future work within the ONS Working Group.

In order to further leverage the use of Internet derived technology and systems, the EPC is encoded as a Uniform Resource Identifier (URI). URIs are the basic addressing scheme for the entire World Wide Web and ensure that the EPC Network is compatible with the Internet going forward.

While an addressing scheme by itself is useful, it can only be used within a network when a mechanism is provided to authoritatively look up information about that identifier (8). This EPC 'resolution' mechanism is called the Object Naming Service or ONS and is what forms the core integrating, or 'truth' verifying, principle of the EPCglobal Network.

This standard specifies how the Domain Name System is used to locate authoritative

metadata and services associated with the SGTIN portion of a given Electronic Product Code. Its target audience is developers that will be implementing Object Naming Service resolution systems for applications. Future work by the ONS Working Group will address how ONS is used for the other namespaces that make up the EPC and that are outlined in the EPCglobal Tag Data Standard (9).

3.8 EPC Information Services (EPCIS)

This specification is currently an EPCglobal working draft and is currently available only to EPCglobal subscribers. The primary vehicle for data exchange between EPCglobal Subscribers in the EPCglobal Architecture Framework is EPC Information Services (EPCIS). As explained below, EPCIS encompasses both interfaces for data exchange and specifications of the data itself.

EPCIS data is information that trading partners share to gain more insight into what is happening to physical objects in locations not under their direct control. (EPCIS data may, of course, also be used within a company's four walls.) For most industries using the EPCglobal Network, EPCIS data can be divided into five categories, as follows:

- Static Data, which does not change over the life of a physical object. This includes: Class-level Static Data; that is, data which is the same for all objects of a given object class For consumer products, for example, the "class" is the product, or SKU, as opposed to distinct instances of a given product. In many industries, class-level static data may be the subject of existing data synchronization mechanisms such as the Global Data Synchronization Network (GDSN); in such instances, EPCIS may not be the primary means of exchange.
- Instance-level Static Data, which may differ from one instance to the next within a given object class. Examples of instance-level static data include such things as date of manufacture, lot number, expiration date, and so forth. Instance-level static data generally takes the form of attributes associated with specific EPCs.
- Transactional Data, which does grow and change over the life of a physical object. This includes:

- Instance Observations, which record events that occur in the life of one or more specific EPCs. Examples of instance observations include "EPC X was shipped at 12:03pm 15 March 2004 from Acme Distribution Center #2," and "At 3:45pm9 22 Jan 2005 the case EPCs (list here) were aggregated to the pallet EPC X at ABC Corp's Boston factory." Most instance observations have four dimensions: time, location, one or more EPCs, and business process step.
- Quantity Observations, which record events concerned with measuring the quantity of objects within a particular object class. An example of a quantity observation is "There were 4,100 instances of object class C observed at 2:00am 16 Jan 2003 in RetailMart Store #23." Most quantity observations have five dimensions: time, location, object class, quantity, and business process step.
- Business Transaction Observations, which record an association between one or more EPCs and a business transaction. An example of a business transaction observation is "The pallet with EPC X was shipped in fulfillment of Acme Corp purchase order #23 at 2:20pm." Most business transaction observations have four dimensions: time, one or more EPCs, a business process step, and a business transaction identifier.

The EPCIS Data Specifications provide a precise definition of all the types of EPCIS data, as well as the meaning of "event" as used above. Transactional data differs from static data not only because as it grows and changes over the life of a physical object, but also because transactional data for a given EPC is typically generated by many distinct enterprises within a supply chain. For example, consider an object that is manufactured by A, who employs transportation company B to ship to distributor C, who delivers the object by way of 3rd party logistics provider D to retailer E. By the time the object reaches E, all five companies will have gathered transactional data about the EPC. The static data, in contrast, often comes exclusively from the manufacturer A. A key challenge faced by the EPCglobal Network is to allow any participant in the supply chain to discover all transactional data to which it is authorized, from any other participant.

3.9 EPCglobal Certificate Profile Standard

The EPCglobal Architecture Framework document (10) describes how security functions such as authentication, access control, validation and privacy protection of individuals and corporations will be distributed across many of the roles/interfaces operating within the EPCglobal network. For example, EPCIS Interface responsibilities include a means for mutual authentication of two parties exchanging EPCIS data across that interface. Another example is the securing of communications between RFID readers and filtering/collection middleware, or reader management systems, when those elements are operating within a non-trusted network environment.

The authentication of entities (subscribers, services, physical devices) operating within the EPCglobal network serves as the foundation of any security function incorporated into the network. The EPCglobal architecture allows the use of a variety of authentication technologies across its defined interfaces. It is expected, however, that the X.509 authentication framework will be widely employed within the EPCglobal network.

To ensure broad interoperability and rapid deployment while ensuring secure usage, this standard defines a profile of X.509 certificate issuance and usage by entities in the EPCglobal network. The profiles defined in this document are based upon two Internet standards, defined in the IETF's PKIX Working Group, that have been well implemented, deployed and tested in many existing environments. The first of these specifications is RFC3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (11). RFC3280 profiles the format and semantics of certificates and certificate revocation lists (CRLs) for the Internet PKI, and is itself a profile of the ITU X.509 standard. The second is RFC 3279 - Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (12). This specification defines algorithm identifiers and ASN.1 encoding formats for digital signatures and subject public keys used in Internet PKI as defined in RFC3280. The goals of this specification are as follows:

- Ensure compatibility with, and thus fully leverage, existing deployed PKI infrastructure. As such, the intent of the profiles defined below is not to define any new functionality that may require updates to existing infrastructure, but to simply clarify and narrow (profile) functionality that already exists.
- Ensure compatibility with existing deployed applications currently used in the supply chain.
- Define a minimum set of capabilities that shall be supported to ensure broad interoperability, while still allowing interested parties to extended and/or further refine to suit their individual requirements.

3.10 Architectural Framework Document

This document defines and describes the EPCglobal Architecture Framework. The EPCglobal Architecture Framework is a collection of interrelated standards for hardware, software, and data interfaces ("EPCglobal Standards"), together with core services that are operated by EPCglobal and its delegates ("EPCglobal Core Services"), all in service of a common goal of enhancing the supply chain through the use of Electronic Product Codes. The primary beneficiaries of the EPCglobal Architecture Framework are **EPCglobal Subscribers** and other **Solution Providers**. An **EPCglobal Subscriber** is any organization that uses EPCglobal Core Services, or participates in the **EPCglobal Standards Development Process** to develop EPCglobal Standards. EPCglobal Subscribers may be further classified as End-users or Solution Providers (or both). An End-user is an EPCglobal Subscriber that employs EPCglobal Standards and EPCglobal Core Services as a part of its business operations. **A Solution Provider** is an organization that implements for End-users systems that use EPCglobal Standards and EPCglobal Core Services. (A Solution Provider may or may not itself be an EPCglobal Subscriber.) Informally, the synergistic effect of EPCglobal Subscribers interacting with EPCglobal and with each other using elements of the EPCglobal Architecture Framework is called the "**EPCglobal Network.**" This document has several aims:

- To enumerate, at a high level, each of the hardware, software, and data standards that are part of the EPCglobal Architecture Framework and show how they are related. These standards are implemented by hardware and software systems in the EPCglobal Network, including components deployed by individual EPCglobal subscribers as well as EPCglobal Core Services deployed by EPCglobal and its delegates.
- To define the top level architecture of EPCglobal Core Services, which provide common services to all subscribers of the EPCglobal Network, through interfaces defined as part of the EPCglobal Architecture Framework.
- To explain the underlying principles which have guided the design of individual standards and Core Service components within the EPCglobal Network. These underlying principles provide unity across all elements of the EPCglobal Architecture Framework, and provide guidance for the development of future standards and new Core Services.
- To provide architectural guidance to end users and technology vendors seeking to implement EPCglobal Standards and to use EPCglobal Core Services, and to set expectations as to how these elements will function.

This document exists only to describe the overall architecture, showing how the different components fit together to form a cohesive whole. It is the responsibility of other documents to provide the technical detail required to implement any part of the EPCglobal Architecture Framework. Specifically:

- Individual hardware, software, and data interfaces are defined normatively by EPCglobal specifications, or by standards produced by other standards bodies. EPCglobal specifications are developed by EPCglobal membership through the EPCglobal Standards Development Process (SDP). EPCglobal specifications are normative, and implementations are subject to conformance and certification requirements. An example of an interface is the UHF Class 1 Gen 2 Tag Protocol, that specifies a radio-frequency communications protocol by which a Radio Frequency Identification (RFID) tag and an RFID reader device may interact. This

interface is defined normatively by the UHF Class 1 Gen 2 Tag Protocol Specification.

- The design of hardware and software components that implement EPCglobal specifications are proprietary to the vendors and end users that create such components. While EPCglobal specifications provide normative guidance as to the behavior of interfaces between components, implementers are free to innovate in the design of components so long as they correctly implement the interface specifications. An example of a component is an RFID tag that is the product of a specific tag manufacturer. This tag may comply with the UHF Class 1 Gen 2 Tag Protocol Specification.

A special case of components that implement EPCglobal specifications are components that are operated and deployed by EPCglobal itself (or by other organizations to which EPCglobal delegates responsibility). These components are referred to as EPCglobal Core Services, and provide services to all EPCglobal subscribers. The design of these components is the responsibility of EPCglobal or its delegates, and design details may be made public at EPCglobal's discretion. An example of an EPCglobal Core Service is the Object Name Service (ONS), which provides a logically centralized registry through which an EPC may be associated with information services. The ONS is logically operated by EPCglobal; from a deployment perspective this responsibility is delegated to a contractor of ONS that operates the ONS "root" service, which in turn can delegate responsibility to other services operated by other organizations.

At the time of this writing, there are many parts of the EPCglobal Architecture Framework that are well understood, and for which EPCglobal standards already exist or are currently in development. There are other parts of the EPCglobal Architecture Framework that are less well understood, but where a need is believed to exist based on the analysis of known use cases. In these cases, the architectural approach has not yet been finalized, and therefore work on developing standards or designing additional Core Services has not yet begun (though architectural analysis is underway within the Architecture Review Committee). This document clearly identifies which parts of the

EPCglobal Architecture Framework are understood architecturally and which parts need further work. This document will be the basis for working through and ultimately documenting the architectural decisions around the latter parts as work continues. (13) The following figure visualizes the core components of the EPCglobal proposed architecture.

3.11 Drivers of Generation3 Development

The EPC system will not stop at Generation 2. The mass market for RFID that Generation 2 will create, should result in new waves of innovation as vendors compete and users become more sophisticated and demanding in terms of reduced cost and increased functionality. As was the case with EPC Generation 1 these competitive pressures will quickly lead to ideas and inventions that cannot be contained within the Generation 2 specification. At this point, most likely around early 2007, the development of a Generation 3 tag will be introduced to the EPC standards development process. Just as occurred with the Generation 1 to Generation 2 transition, successful Generation 2 incumbents will resist this transition in turn, while entrepreneurs, start ups and large corporations coming late to the EPC market will jump in. The momentum will build, and around 2008 or 2009 Generation 3 tags will appear, starting the cycle described here all over again. The RFID reader infrastructure must be ready for more change.

3.12 Software Defined Radio: Flexible, Future-proof RFID Infrastructure

3.12.1 Solving the Problem of Tag Variation & Iteration

The problem of continuous change in the EPC market posed in presented in this thesis is vitally important for all RFID users and especially those responsible for buying and

installing RFID reader infrastructure. While tags are the consumables of the RFID systems, constantly varying, iterating and regenerating, the RFID reader infrastructure is a deployed capital expense that cannot easily or cost effectively be replaced every time a new tag variant appears. Further, the comings and goings of tags are not neatly synchronized. Generation 1 did not turn into Generation 2 at the stroke of midnight – the two generations must coexist, perhaps for a reasonable amount of time. And once Generation 1 has disappeared, the reality (and hype) surrounding Generation 3 will begin, as well as the introduction of new classes of tag. All this change is good – in fact crucial – for the RFID user. It will deliver ever-improving performance, and ever-decreasing costs. But it can only flourish is the RFID reader infrastructure is not an impediment to change – if it can endlessly and easily adapt and adjust as new variations of tag appear and disappear. The technology that enables this endless adaptation is called Software Defined Radio or SDR.

3.12.2 A Definition of SDR

A software-defined radio (SDR) uses software for the modulation and demodulation of radio signals. An SDR performs the majority of its signal processing in the digital domain, most commonly in a digital signal processor (DSP), which is a type of microprocessor specifically optimized for signal processing functions. The magic of an SDR based RFID reader is that it can receive and transmit a new form of RFID communication protocol simply by running new software on existing SDR hardware.

A software-defined RFID reader consists of an RF analog front end ("AFE") that converts RF signals to and from the reader's antennas into an analog baseband or intermediate frequency signal, and analog to digital converters and digital to analog converters which are used to convert these signals to and from a digital representation that can be processed in software running on the reader's digital signal processor.

Put simply, RFID readers based on SDR technology store all protocol information in software, and use protocol-neutral hardware to generate and detect radio waves. This is in contrast to conventional RFID readers, where protocols are created by adding hardware, and changing protocols requires changing components in the reader.

3.12.3 Brief History of SDR

Software defined radio technology has long been important in the military context, where new radio equipment must inter-operate with legacy equipment, much of which is used for many years beyond its design lifetime. Additionally, the US military is often called upon to work together with allies that have old, outdated equipment that is incompatible with the more modern US communication hardware. This is exactly analogous to the Generation 1 to Generation 2 (and beyond) transition in RFID.

Military SDR projects date back to the early 1990s, and several were fielded in that time frame. Aware of these developments, in 1999 the MIT Auto-ID Center began exploring the idea of using SDR in RFID readers. This resulted in a development contract to ThingMagic LLC in 2001. ThingMagic demonstrated the first SDR-based RFID reader at the Auto-ID Center's Board Meeting in Cambridge, Massachusetts in November 2001. Commercial SDR-based RFID readers were first introduced by ThingMagic in 2003.

3.12.4 Benefits of SDR in RFID Readers

SDR-based readers offer significant benefits to RFID users. In particular, SDR-based designs have a unique ability to adjust and upgrade to new protocols, and to be constantly improved via firmware upgrades. With the addition of standard networking capabilities such as remote management, large networks of deployed SDR-based RFID readers can be upgraded in a matter of minutes from a remote console in a customer's Network Operations Center, instantly becoming capable of reading new types of tag.

SDR-based RFID readers must be able to support via firmware upgrades, at least the current, the previous and the next RFID generation.

4. RFID in supply chain management

RFID is a technology that can be integrated to unlimited environments. It can make our lives a lot easier in day-to-day situations and processes. It can alter processes in our work, in our personal life, in the enterprise. As it has been clearly stated in the introduction, this report is focused in the applications of the RFID technology in the field of supply chain management. Before we continue discussing this kind of applications, we will see in little detail what the most common applications of RFID are in this early and immature state of this technology.

While this technology has grown over the last three years, much of the growth came from traditional, established applications such as security/access control, automobile immobilization, toll collection, and animal tracking. At present, many emerging applications remain in the early adopter phase (i.e., pallet tracking, POS/m-commerce, baggage handling, etc.). However, the increase in available RFID applications suggests that the technology is moving beyond traditional application niches. Manufacturers have proven that RFID technology works in many application environments and end users have developed a better (yet incomplete) understanding of the benefits of RFID technology.

While the potential for viable RFID applications appears virtually limitless, the lack of implemented standards and high RFID system costs often become the decisive barriers to greater adoption. While the traditional application segments will continue to enjoy solid growth, opportunities are rapidly unfolding in the emerging application segments. The most important of these segments are:

- Baggage Handling
- Rental Item Tracking
- POS m-Commerce
- Real time Location System (RTLS)
- Supply Chain Management

Applications are constantly being developed to streamline data capture applications. Whether in the supply chain or in mobile computing, RFID applications typically fall into one of the categories below:

- **Closed Loop Systems**

  Closed loop systems are systems that are traditionally considered standalone and they are used in the following:

  - Assembly operations
  - Manufacturing processes
  - Work in Progress
  - Animal tracking
  - Healthcare (Inventory and equipment control)

  For example, an automotive manufacturer may use RFID technology in their manufacturing plants to track car frames as they move through the paint stations. The information is used for internal purposes only to track inventory and quality control.

- **Open Systems**

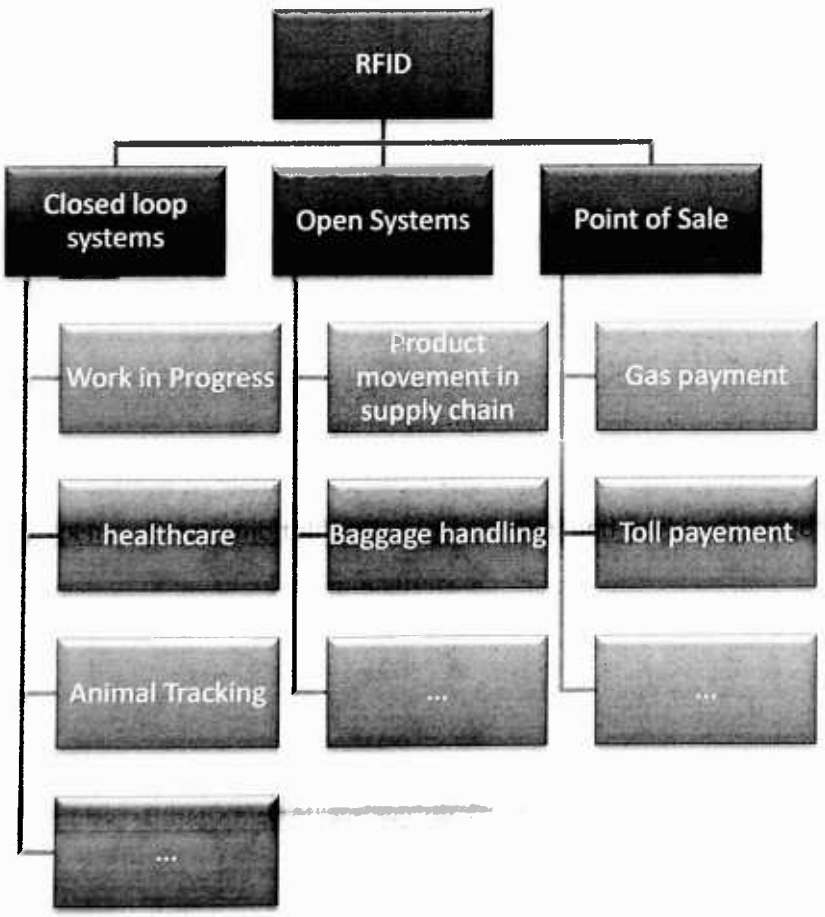
  When information concerning the movement of products that incorporate RFID tags is shared with others, this is considered an open system. A prime example of this would be sharing movement information of products as they travel through the supply chain. Pallet tags being loaded on a truck from a manufacturer would be read and that information could be shared with the appropriate downstream warehouse or distribution center. Likewise, when the warehouse or distribution center unloads the trucks, the same pallet tags are read and that information can be shared with the upstream manufacturer.

  Open systems are rarely used now because of the lack of database and protocol standards that define how this information will be shared. However, their potential is tremendous, as they would enable companies to track a single pallet, case or item

throughout the supply chain, instead of relying upon input from each touch point. The EPC initiative within the supply chain is a good example of an open system.

- **Point of Sale**

  These systems are used as fast payment systems such as toll road applications, gasoline payment, or parking garage applications. Mobil/Exxon Speedpass is a POS application example that allows customers to pay for their gas (and purchases at some grocery stores) by passing an RFID-enabled ID card over an RFID reader at the gas pump. The system automatically charges the customer's credit or debit card with the expense. Electronic toll collection systems and parking garage access are other examples of point-of-sale applications.



4.1 Common **Applications**

The following are only a sampling of the applications using RFID. The applications receiving the most press these days are those associated with pallet and case tracking in the supply chain. The following application examples should indicate there are seemingly unlimited opportunities to implement RFID technology in a variety of markets and for numerous uses.

- **Transportations**

  New passports are now being manufactured with an integrated RFID chip which contains encrypted information about the passport's owner. When all passports are replaced with the new ones, automated passport checking could be possible with the use of face recognition systems. Additionally, passport forgery will now be very difficult.

- **Baggage Handling**

  Most airlines currently use bar coding systems in their baggage handling operations, but unfortunately baggage still arrives at the wrong location on occasion. Airlines are developing RFID tags embedded into the bar coded baggage tags to interact with the conveyance systems. These RFID systems ensure that baggage reaches its intended location the first time.

- **Library Information Systems**

  Libraries are using RFID systems for tracking books and other properties. Books are being tagged with a separate RFID tags to accelerate checkout, and they control theft. A book leaving the premises without being checked out will set off an alarm. Even if the book is not recovered, the system is automatically updated to show that it is not available for checkout.

- **Returnable Containers Tracking**

  By tracking pallets, totes and other containers with RFID, and building a record of what is stored in the container. Returnable Product Containers (RPCs) are used to

package and transport produce. A grower packs and ships fruits and vegetables in RPCs for travel through distribution to a store's produce department. When the product container is empty, it is returned for cleaning and reuse. The RFID tag is used not only to keep track of the location of the RPC, but also to document its cleaning history, from the date and temperature of the washing to the chemicals used.

- **Car Dealerships**

  RFID systems can be used to manage inventory of automobiles in new and used car dealerships and in rental car lots.

- **Rental Cars**

  For rental car companies, RFID would allow fast and easy access to maintenance and service records.

- **Hospitals**

  RFID can be useful in checking out and tracking expensive medical equipment. Placing an RFID tag on the diagnostic or monitoring machine could read these devices as they leave one area and enter another. This would allow quick and efficient location of these most important devices.

- **Animal Identification**

  RFID is being used in the cattle industry in the hopes of helping identify the source of disease and medical histories. Secure identification of cattle by means of a tag inserted into the stomach of an animal provides accurate records for automated farm management.

- **Point of Sale**

  An RFID-enabled ID card (Mobil/Exxon Speedpass) allows customers to pay for their gas by passing a card over an RFID reader at the gas pump. The system

automatically charges the customer's credit or debit card with the expense

- **Toll Roads and Parking Garages**

  Electronic toll collection systems and parking garage access are other examples of point-of-sale applications. Having an RFID tag within your automobile allows frequent users of toll roads to simply drive through the toll stations. The tag will be read and they can be billed each month. The same could be true for charging for use at parking garages.

- **Utility Companies**

  RFID could be used to identify where buried cables, pipes, etc are located.

- **Security and Access Control**

  The movement and use of valuable equipment and personnel resources can be monitored through RF tags attached to tools, computers, etc. or embedded in credit-card-size security badges.

- **Supply Chain Management**

  Large corporations which wish to remove costs and inefficiencies in supply chain operations are driving much of the recent interest in RFID and smart label technology. The idea of scanning pallets as they arrive in distribution centers or back store docks and instantly updating the system with all of the contents of the pallet is appealing to many companies that are currently paying a small army of people to accomplish this task as a full-time job. Reallocating those personnel resources could potentially provide a substantial increase in productivity and efficiency for these operations. This is the area where current attention is being focused. This is because of the mandates from Wal-Mart and the Department of Defense to their top suppliers to implement RFID when shipping products to them.

4.2 Applications in supply chain management

Having these in mind we will now bring the focus in the field of SCM. Operating an integrated supply chain requires continuous information flows, which in turn help to create the best product flows. The customer remains the primary focus of the process. Achieving a good customer-focused system requires processing information both accurately and in a timely manner for quick response systems that require frequent changes in response to fluctuations in customer demand. Controlling uncertainty in customer demand, manufacturing processes, and supplier performance are critical to effective SCM.

Successful SCM requires a change from managing individual functions to integrating activities into key supply chain processes. Traditionally, both upstream and downstream portions of the supply chain have interacted as disconnected entities receiving sporadic flows of information over time.

For example, the purchasing department placed orders, as requirements became necessary and marketing, responding to customer demand, interfaced with various distributors and retailers and attempted to satisfy this demand. Orders were periodically given to suppliers and their suppliers had no visibility at the point of sale or use. Satisfying the customer, often translated into demands for expedited operations throughout the supply chain as member firms reacted to unexpected changes in demand.

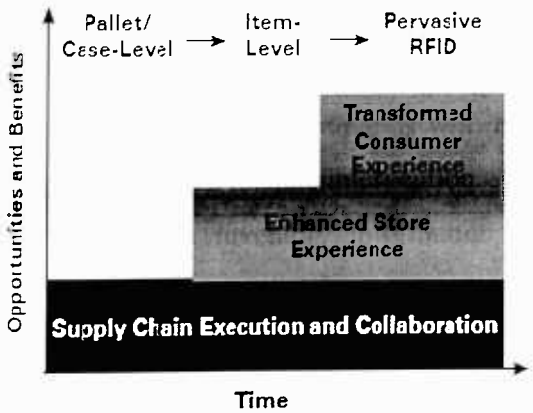
In many major corporations, such as 3M, management has reached the conclusion that optimizing the product flows cannot be accomplished without implementing a process approach to the business. (14)

The consumer products/retail industry has long taken advantage of new digital technologies to transform its operations. Going back to early mainframes in the 1950s, through the adoption of the barcode and electronic point-of-sale (POS) devices in the 1970s, up to more recent innovations such as electronic data interchange (EDI) and Internet-based commerce, successive waves of new technologies have helped companies better manage operations and improve service to suppliers, customers and consumers.

RFID technology, and, in particular, the standardized EPC and its associated information

flow via the EPCglobal Network, are poised to enable the next wave of evolution in the way manufacturers, retailers and their business partners share information and work together to satisfy consumer demand. EPC can be thought of as an extended barcode containing a serialized item key that enables individual products to be uniquely identified. Unlike existing barcode technology, EPC systems, based on the use of radio frequencies, do not require line-of-sight scanning. This fundamental change improves the speed and potential accuracy of data collection and provides the following new capabilities:

- Faster scanning and product handling, with the capability to support hundreds of tag "reads" per second (versus one-at-a-time as with barcodes) and to conduct automated scanning with limited manual intervention.
- New opportunities to collect inventory information and "see" the flow of products, potentially in real time and in locations not previously feasible across the supply chain and in the store.
- Automated "triggering" of appropriate actions (e.g., replenishment orders, stock alerts) with less manual intervention.
- Identification of discrete items, for example by flagging duplicate or invalid codes, thus enhancing the execution of promotions, track and trace, product authentication and other activities.



Envisioned evolution of EPC adoption

The focus for most companies and in most product categories today is on pallet- and case-level tagging. Even at this level, EPC adoption can lead to a better consumer shopping experience by enabling companies to improve supply chain execution and collaboration. Looking further out, the broader vision for EPC is to tag individual products at the item level. While technology costs remain too high, in the industry to make this feasible in the near term, for most product categories, these costs will inevitably come down as EPC adoption scales across various industries including consumer products/retail. As this occurs, consumers will likely see increased value and noticeable enhancements to the shopping experience, enabled by new supply chain and store management practices. For example:

- Product out-of-stocks would become very rare, as "intelligent", EPC-capable store fixtures provide retailers and manufacturers with stock visibility all the way to the shelf and enable more dynamic restocking procedures.
- The assortment and presentation of products would be more aligned with consumer shopping preferences, as EPC data is used to improve category management and automated shelf-level monitoring helps ensure compliance with plan-o-grams.
- Shoppers would only see "fresh" products available for purchase, as item-level stock monitoring helps retail employees quickly and efficiently identify aging or obsolete products that should be removed from the sales floor.
- Store employees, with access to item level product inventory information, could quickly help a customer find a specific size, color or model anywhere in the store or throughout the retail chain.
- Consumers could obtain valuable information that helps them make better shopping decisions (for example, product features, usage instructions and promotions on complementary products), through digital displays or information kiosks that interact with EPC tagged products.
- The checkout process would no longer be a primary source of consumer pain, as EPC-tagged products make possible rapid, automated tallying of purchases.

EPC might ultimately become pervasive throughout the consumer environment, as a wide range of consumer devices become capable of interacting with tagged products. Glimpses of potential future applications can be seen today in other markets, such as RFID-based toll collection systems for automobiles and identification systems for pets. As successive generations of EPC technology are developed and deployed, they will likely become the basis of unimagined shopping experiences and product/service offerings. While it is difficult to predict with certainty what new applications and services will emerge, the successful ones will surely be those that respond best to consumers' needs.

We will now see some commercial software solutions provided by major software vendors, in the field of SCM that utilize or focus on the RFID technology and then we will conclude by describing our approach and solution proposed for the RFID part of the SMART project.

5. Commercial RFID solutions

In this chapter we will discuss in little detail some commercial solutions in the field of SCM. These solutions are provided by major vendors in the field of SCM and they use RFID either as the central point of the solution or as a plug-in of existent proven solutions.

5.1 IBM Solution

The solution proposed by IBM is based on Java Enterprise Edition (J2EE) technology and the infrastructure can be supported by the **RFID Premises Server** or by the **Remote Server**. The selection between these two servers should depend on the company's needs.

- **RFID Premises Server**

  It can be implemented at either remote locations or a central data center. It is designed to collect and filter data from RFID devices, execute business processes and integrate the RFID information with third-party software applications for ERP (enterprise resource planning) and warehouse management.

  An IBM RFID solution is made up of three different elements: Devices, WebSphere® RFID Premises Server V1.1, and a WebSphere integration server:

  - Devices, such as readers, scanners, printers, etc., embedded with WebSphere RFID Device Infrastructure. Device Infrastructure is an RFID-enabled middleware product that IBM provides to select partners to place on their devices.
  - IBM WebSphere RFID Premises Server, a middleware product that aggregates, monitors, interprets, and escalates RFID events to detect critical operational events.

- An IBM integration server, such as WebSphere Business Integration Server or Server Foundation, is recommended to allow customers to fully integrate the information flowing in from the edge of their business with their enterprise operations.

WebSphere RFID Premises Server is targeted for deployment in warehouses, distribution centers, or manufacturing plants. Premises Server is a robust, extensible J2EE application platform for the integration of RFID events from RFID devices into business processes. In addition to providing the ability to filter, aggregate, monitor and escalate RFID events to detect critical business operational events or to track the location of physical objects, Premises Server:

- Delivers starter kits for commonly needed functions, including Dock Door Receiving and Print-Verify-Ship
- Supports SUSE Linux 8 and Windows 2003 Server
- Provides integration with new RFID printers and readers
- Provides material for capacity planning and performance planning, so that you can more effectively plan for your RFID deployment

(15)

- **Remote Server**

It is designed to be deployed remotely. It is tailored mainly toward integrating the newer handheld, kiosk and self-service environments with legacy devices such as IBM POS (point-of-sale) terminals. It provides:

- Mobile shopping devices with personalized information at the consumer's fingertips.
- Transaction log (Tlog) handling for non-IBM 4690 applications to move POS data to headquarters.
- RFID technology to speed the checkout process and locate merchandise in the store.

- Digital merchandising solutions that can provide consumers with dynamic product information and comparisons on demand.
- Accurate inventory information to help prevent erroneous reordering of merchandise.

WebSphere® Remote Server delivers a fully integrated platform to help manage remote environments such as retail stores and branch offices. This infrastructure offering extends the IBM Enterprise Business Integration technology to distributed locations, enabling integration from the enterprise to the edge of your business. A retailer manage the business more cost effectively, help increase employee productivity, and create a unique shopping experience for the consumers. Employees have better access to customer, product, and sales information, increasing productivity and providing better service to consumers. WebSphere Remote Server is a key component of the Store Integration Framework and the IBM on demand operating environment (16).

The Premises and Remote servers each come with a built-in WebSphere Application Server, an IBM Universal DB2 database repository and IBM Tivoli management tools, among other components. The Premises server can be used for remote server management, too.

5.2 SAP solution

The mySAP Supply Chain Management (mySAP SCM) application is software that can help your organization transform a linear supply chain into an adaptive supply chain network, in which communities of customer-centric, demand-driven companies share knowledge, intelligently adapt to changing market conditions, and proactively respond to shorter, less predictable life cycles (17).

Based on the SAP NetWeaver™ platform, SAP RFID is designed to handle the massive amount of additional data accumulated from scanning RFID tags. It minimizes the

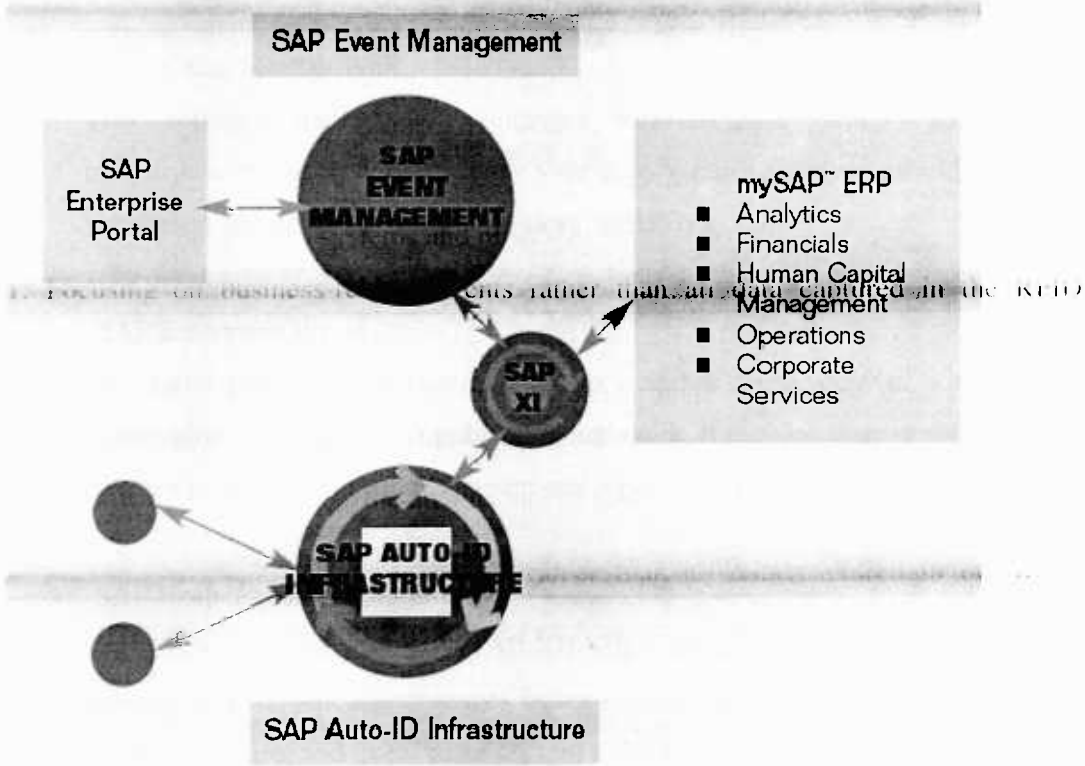
replication of data across systems and processes by:

1. Focusing on business-related events rather than all data captured in the RFID infrastructure

2. Basing decision-support applications on aggregated, rather than granular data as much as possible.

The following figure depicts the architecture of SAP's solution.

The SAP® Solution Package for Massive Auto-ID Data



It is composed by the following components:

- SAP Auto-ID Infrastructure

  Integrates all automated communication and sensing devices including bar-code devices, Bluetooth devices, and RFID readers and printers. It does not only filter, buffer, verify, and aggregate data coming from different hardware sources, but

also gives business context to the data, drawing on the applications in mySAP™ ERP that support the business processes in question – inventory management and warehouse management, for example.

- ERP adapters

  ERP adapters are responsible for synchronizing electronic product code number ranges across multiple auto-ID infrastructures. They also support packing, unpacking, loading, unloading, and advanced shipping notification (ASN) processes that involve RFID data.

- SAP Event Management

  This software tracks and exchanges auto-ID data through the various infrastructures and systems used by your supply chain partners, providing cross-warehouse and cross-company inventory visibility.

- SAP Enterprise Portal

  The portal provides role-based access to – and a single view of – key RFID information coming from supply chain activities. It enables your supply network partners to access event management and report information.

- SAP Exchange Infrastructure

  SAP Exchange Infrastructure (SAP XI) helps minimize administrative effort by queuing and sequencing messages being exchanged among mySAP ERP, auto-ID infrastructure, and event management software.

5.3 SUN Solution

The SUN's solution is named Java System RFID and it is based, obviously, on the java language. The Java System RFID Software is Sun's standards-based RFID Middleware that provides all this basic functionality and more along with the ability to operate at

high levels of reliability and scalability in mission critical deployments. In addition to device and event management, the Java System RFID Software provides a RFID Information Server and the requisite interfaces to enterprise information systems to facilitate the storage and integration of business events with the enterprise's business processes (18).

The solution is composed by the following components:

- Java System RFID Event Manager

  It is designed to process streams of tag or sensor data (event data) coming from one or more reader devices. The RFID Event Manager has the capability to filter and aggregate data prior to sending it to a requesting application. The RFID Event Manager can also be programmed with other types of filters to enforce specific business rules.

- Java System RFID Information Server

  Provides access to significant business events generated by the Java System RFID Event Manager. It also serves as an integration layer that offers several options for integrating the RFID Event Manager with existing EIS or custom enterprise applications.

Using software and application programming interfaces (APIs) that are part of the Java Enterprise System enables developers to quickly and flexibly integrate EPC data with enterprise applications. Data from the RFID Event Manager feeds into Sun's RFID Information Server, where it is stored and made available in a consistent manner to any application that needs it.

Java System RFID Software is shipped with built-in support for a variety of industry-leading printers and readers. Developers and integrators can create adapters for other devices with the Java System RFID Software Toolkit.

5.4 Oracle Solution

Oracle's solution is named RFID and Sensor-Based Services.

Based on Oracle Database 10g, Oracle Application Server 10g, Oracle Enterprise Manager 10g, and Oracle E-Business Suite 11i.10, Oracle Sensor-Based Services enable companies to quickly and easily integrate sensor-based information into their enterprise systems. Oracle's solution includes a Compliance package, an RFID pilot kit, and integrated support in Oracle E-Business Suite and Oracle Application Server (19).

The key components of the Oracle's solution are:

- Capture Information: Oracle Application Server 10g
- Manage Information: Oracle Database 10g
- Analyze Information: Oracle Business Intelligence
- Access Information: Oracle Application Server Portal and Wireless

5.5 General evaluation of commercial solutions

The solutions that are discussed in this chapter were evaluated as thoroughly as possible given the fact that some of them were available free with an evaluation license, while others were not, so we had to evaluate them based completely on the vendor provided release information.

Based on this mixed experience, the outcome of the evaluation process is that currently very few solutions are designed based on the RFID technology. Most commercial solutions are the common SCM solutions provided by software vendor with the addition of an RFID module. As a consequence of this fact, if one wants to use the functionality provided exclusively by the RFID, he has to purchase the full package, which contains the tenfold functionality in comparison with the functionality needed. That is because the RFID part is a module and needs the full package in order to work.

Our general impression of the domain of commercial RFID solutions, is that it is

characterized by great immaturity. There is a lot if distance that needs to be covered so that they can be characterized as standards-based and fully modular. The paradigm that is introduced with the RFID technology, named Internet Of Things, can only be manipulated with RFID-oriented and RFID-centric solutions, which take into account the continuous data streams of RFID tagged objects.

6. RFID solution for the SMART research project

6.1 The SMART Research Project

The practices described in chapter 4 (information-sharing work practices and infrastructures between trading partners to support the use of free, standards-based information exchange and enable transformed business processes) which are identified by a GCI and IBM joint report of a shared vision for transforming business processes (20), are totally in line with the objectives of the SMART research project.

The SMART Research Project is partly funded by the European Commission through the IST program (No. ST-5-034957-STP) and aims at the intelligent integration of supply chain processes and consumer services based on unique product identification in a networked business environment.

More specifically, the SMART project will provide the infrastructure, electronic services and software applications to enable supply chain collaboration and innovative consumer services in the above context, based on a scalable-distributed-architecture and building on the possibilities provided by peer-to-peer networks, web-service orchestration and choreography, data-stream systems and smart tagging technologies.

The SMART collaboration infrastructure shall be in close integration with the GDSN and EPC Network information infrastructures and will operate as an additional layer on top of them, in order to provide a complete and solid collaboration framework offering innovation to specific supply chain processes and consumer services.

The project specifically focuses on the following application areas, where some indicative usage scenarios are given as examples below:

- Consumer information services: For example, the consumer is guided through the store to find the products he/she is looking for; the consumer can track-down to a product instance the quality and history in front of the store-shelf; the

consumer can see other useful information while in front of the shelf, such as whether the product is available in the back-room and request it or whether there is a joint/ personalized promotion, etc.

- Store management: For example, shelf replenishment practices change based on real-time information about shelf availability and product shortages; both the retailer and the supplier are notified in case a product is missing from the shelf; the store personnel but also the supplier can dynamically monitor the shelf appearance and the extent to which a shelf planogram is applied, etc.

- Promotion/ event execution: For example, retailers and suppliers are able to dynamically manage their promotional offerings to consumers, e.g. lower the price of specific product instances in real-time due to freshness limitations or poor consumer response; they run promotion pilots and get real-time information about promotion effectiveness before rollout; they monitor product availability in relation to promotion events and take corrective actions, etc.

- Product traceability and reverse logistics: For example, suppliers can track down to the point-of-sales the location of their products/ displays/ pallets so that they can withdraw products from the market in case of a food crisis or close expiration date as well as get pallets/ displays back after an event, delivery to the store, etc.

- Inventory management and collaborative replenishment (CRP/VMI, CPFR): For example, demand forecasting algorithms are updated to take into account the extra information provided by unique product identification in combination with promotion events and shelf availability information that was not available before; retailers use this information to manage store and warehouse inventories more efficiently; suppliers also get this information in real-time to more efficiently manage their buyers inventory in a CRP/VMI or CPFR collaboration process.

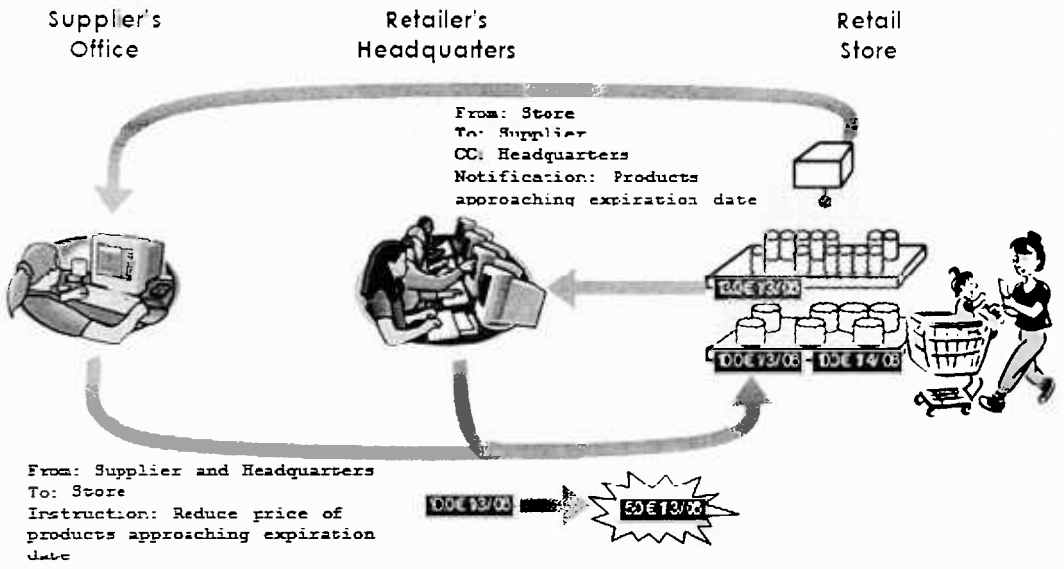
The above processes have been identified by industry leaders as high-priority areas where the possibility to uniquely identify product instances is expected to have the greatest business impact and render the most immediate results.

The project's specific objectives include:

- To enable innovative in-store consumer services and new supply-chain collaboration scenarios which exploit the capabilities for unique-product identification and real-time information.
- The provision of a scalable, reliable and secure infrastructure supporting information sharing, collaboration and electronic services in the above context
- The development of new decision-support algorithms and software tools, taking advantage of unique product identification capabilities and real-time information flows, to support store operations and supply chain processes
- The establishment of a collaborative services repository to enable the open and dynamic integration of supply chain processes in a global environment
- The provision of reliable and real-time end-to-end information about product quality and history to supply chain partners as well as to educated consumers through innovative electronic services.
- The development of a technological framework incorporating inter-organizational standardized processes in the above context, exploiting the innovative technologies of web-service orchestration and choreography and supporting the required process transformation.
- The assessment from a business and marketing perspective of the impact that the developed services will have on increasing consumer value and building consumer loyalty.

As an example, the following figure presents an indicative scenario enabled by the respective SMART service supporting dynamic pricing, under the "Promotion/ event execution" application area, where the supplier collaborates with the retailer in order to

reduce the price of some products approaching their expiration date.



Indicative dynamic-pricing scenario enabled by SMART

The main technologies employed in the course of the SMART project include the following:

- Smart-tagging technologies and radio-frequency identification (RFID), enabling unique product instance identification in an automatic way
- Web-service orchestration and/or choreography in order to enable inter-organizational process integration in a distributed network environment.
- Data-stream management systems supporting continuous queries based on transient data streams regarding product movement across various stages of the supply chain, e.g. store, warehouse etc.
- Real-time analytics and decision support supporting critical supply chain process and consumer services based on unique product identification information

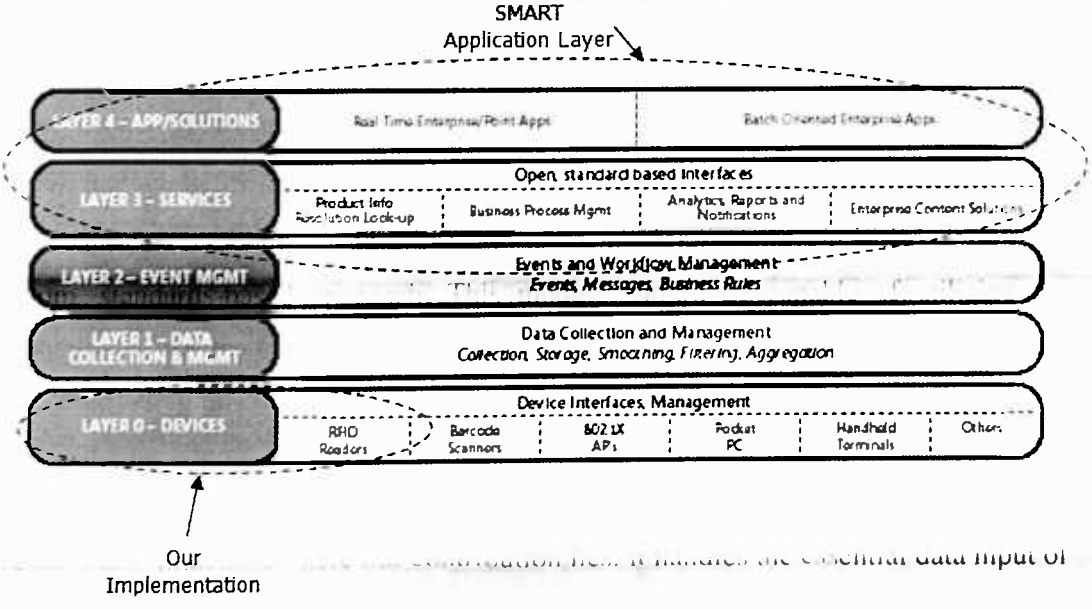
Our contribution in this project, which is the basis of this thesis, covers the first bullet. It evaluates the RFID technology, within the current context, it proposes a standards-

based solution to enable unique product instance identification in an automatic way and to provide updates for context-dependent important information and, finally, it provides an implementation for a part of this solution. The matters of implementation are discussed in the following chapter.

6.2 Implementation

Due to the fact that for the RFID module of the SMART project we want total control over the capabilities and the configuration of the module, we have decided to follow a custom, standards-based approach. This module should be capable of utilizing the available readers, reading data from them, perform cleaning operations on that data and finally providing this information to the upper levels of the architecture.

The following figure represents a typical RFID based information system. On this figure we have indicated where the application layer of the SMART project is located. We have also indicated where our contribution lies. It handles the essential data input of the SMART application layer.



The ratified Reader Protocol, which has been discussed previously in this report, was released in the second semester of 2006. As a consequence, each vendor that manufactures and provides RFID readers to the market has been, until now, equipping these readers with a custom, proprietary communication interface protocol. There are even many cases, that readers from the same vendor are equipped with a different protocol. So if someone wants to communicate, for example, with an Intermec reader he has to read the reader's specification, understand the protocol, which includes messages and commands, and then be able to get data from this (and only this) type of reader. The situation becomes much more complicated, when we add to the environment many RFID readers from different vendors.

So, for our solution to be in line with the EPCglobal standards, we decided to implement a proxy which would be able to translate the proprietary protocol commands and messages of the readers we used, to the standard messages and commands of the Reader Protocol.

EPC technology, especially when implemented using RFID, generates a very large number of object reads throughout the supply chain and eventually into consumer usage. Many of those reads represent non-actionable "noise." Thus we need to build a filtering module, which will perform cleaning operation on the data provided by the lower level, the RFID Readers. This module should be able to trigger events, based on rules that we would set. The best way to do this is by implementing the Application Level Events Specification. The Application Level Events (ALE) provides a flexible interface to a standard set of accumulation, filtering, and counting operations that produce "reports" in response to client "requests." The client is responsible for interpreting and acting on the meaning of the report (*i.e.*, the "business logic").

During our research in the field of the RFID technology, we ran across the Accada Project. Accada is an open source RFID prototyping platform that implements the EPC Network specifications. *It is intended to foster the rapid prototyping of RFID applications and to accelerate the development of an Internet of Things.* The Accada project was initiated by Christian Floerkemeier, Matthias Lampe, and Christof Roduner of the Distributed Systems Group at ETH Zurich led by Friedemann Mattern and the

Auto-ID Lab at ETH Zurich/University of St. Gallen led by Elgar Fleisch.
The Accada project has three working groups that work on the different levels of the EPCglobal architecture stack. So within the Accada project, exist:

- The Accada Reader Project

  The objective of the Reader Project is to implement the reader role in the EPC Network and to develop the appropriate tools that facilitate communication with the reader instance.

- The Accada Filtering & Collection project

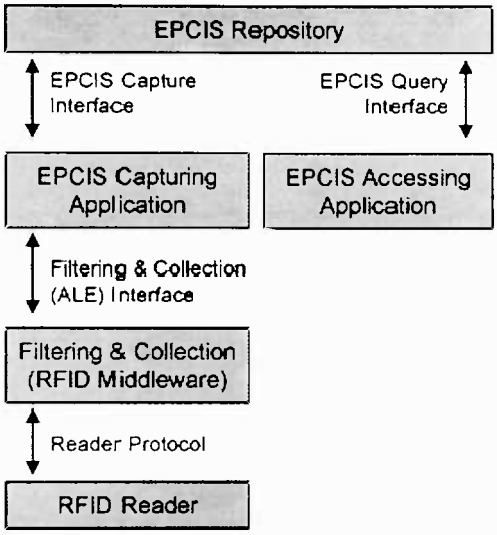
  The objective of the **Filtering & Collection** Project is to implement the Filtering & Collection role in the EPC Network and to develop the appropriate tools that facilitate communication with the Filtering & Collection instance.

- The Accada EPCIS project

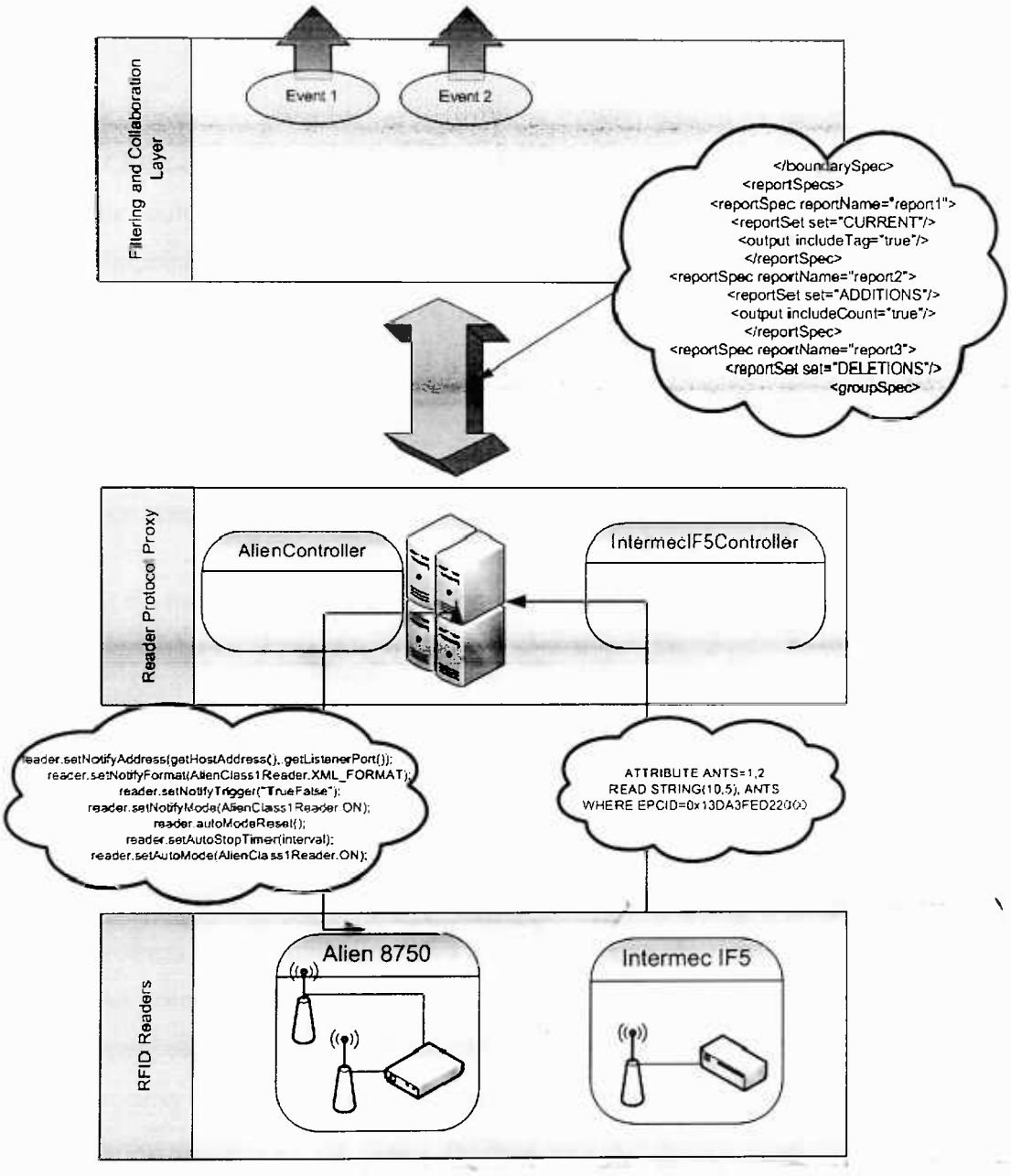
  The objective of the EPCIS Project is to implement the EPCIS role in the EPC Network and to develop the appropriate tools that facilitate communication with an EPCIS Repository instance.

Each of these projects implements the corresponding protocols that were described in chapter 3.

Our decision to build an application using the same protocol stack (except from the highest level – EPCIS) with the same protocols per stack level guided us to join Accada and contribute to this project. The components that will be utilized in the SMART project when this is complete are represented in the following figure:



The Accada Reader Project utilizes a hardware abstraction layer (HAL), which is a point of integration of all the different types of readers and their supported protocols. It provides an application programming interface (API) that needs to be implemented for every RFID reader that should be available under the HAL. **Our contribution** to the project was the HAL implementation for two readers; The Alien 8750 RFID Reader and the Intermec IF5 RFID reader. In that way, these two readers became accessible via the Reader Protocol. One can have a better perspective of the contribution via the following figure. The middle layer (reader protocol proxy) is a translation layer between the lower (proprietary) protocols and higher (standardized) protocol.



It must now be clear that our implementation is positioned in level 0 of the system stack, as it was depicted previously in this chapter.

The Filtering and Collection project utilizes the Reader protocol in order to generate the events defined by the Application Level Events protocol. So, by making these readers RP-enabled we accomplished the readers' ability to generate clean and useful

information.

The method communication between the SMART RFID subsystem and the rest of the SMART Collaboration system has not yet been defined and thus remains as a future work. This could be accomplished via a database, a flat file of data stream, or, even better, a combination of those, with a dynamic selection of the type of communication based on criteria like the importance of the timeline of an event or the available storage capacity or the type of usage (e.g. long-term inventory management).

6.3 Recommendations

Regarding the hardware specifications that need to be met if the SMART case, so that the proposed solution is possible to be implemented, we should state that the following are mandatory:

- RFID Readers implementing the EPCglobal Specification for RFID Air Interface Protocol (1), thus reading in the UHF band,
- RFID Tags compliant with the EPCglobal Specification for RFID Air Interface Protocol (1). Additionally, due to the fact that in the field of supply chain, the products move within populated areas with many electromagnetic interferences and noise, the RFID tags should also have high reading rate and should be equipped with two or more embedded antennas for more accurate readings.
- A proxy server running the reader protocol. This proxy will run the Accada's implementation of the reader protocol and will translate the reader protocol messages received by the upper levels of the EPCglobal Architecture stack, to the proprietary messages of each RFID reader that will be used.

7. Appendix

I. Abbreviations

A

ALE - Application Level Events
API - Application Programming Interface
ASN - Abstract Syntax Notation

B

C

CGP - Consumer Packaged Goods
CRC - Cyclic Redundancy Check
CRL - Certificate Revocation List

D

DNS - Domain Name Service
DoD - Department of Defence (USA)

E

EAN - European Article Numbering
EDI - Electronic Data Interchange
EPC - Electronic Product Code
EPCIS - EPC Information Systems

F

G

GCI - Global Commerce Initiative
GDSN - Global Data Synchronization Network
Gen1 - Generation 1
Gen2 - Generation 2
GIAI - Global Individual Asset Identifier
GID - General Identifier
GLN - Global Location Number
GRAI - Global Returnable Asset Identifier
GSCF - Global Supply Chain Forum
GTIN - Global Trade Item Number

H

HAL - Hardware Abstraction Layer

I

IAB - Internet Architecture Board
IETF - Internet Engineering Task Force
IP - Internet Protocol

J

K

L

M

MIB - Management Information Base
MTB - Message Transport Binding

N

O

ONS - Object Name Service
OSU - Ohio State University

P

PKIX - Public-Key Infrastructure X.509 group

Q

R

RFID - Radio Frequency Identification
RM - Reader Management
POS - Point Of Sale
RP - Reader Protocol
RPC - Returnable Product Container

S

SCM - Supply Chain Management
SDR - Software Defined Radio
SMART - Intelligent Integration of Supply Chain Processes and Consumer Services based on Unique Product Identification in a Networked Business Environment
SMI - Structure of Management Information
SNMP - Simple Network Management Protocol
SSCC - Serial Shipping Container Code

T

TCP - Transfer Control Protocol

U

UCC - Uniform Code Council
UDP - User Datagram Protocol
UHF - Ultra High Frequency
UNF - University of North Florida
UPC - Universal Product Code

V

W

X

XML - eXtensible Markup Language

Y

Z

II. Bibliography

1. **EPCglobal Inc.** *Specification for RFID Air Interface.* 2005. version 1.0.9.

2. —. *Tag Data Standards.* version 1.3.

3. —. *EPC Tag Data Translation Standard.*

4. —. *Reader Protocol Standard.* version 1.1.

5. —. *Reader Management Standard.* version 1.0.

6. **IETF Network Working Group.** *Structure of Management Information Version 2 (SMIv2).* s.l. : IETF, 1999. RFC 2578.

7. **EPCglobal Inc.** *Application Level Events Standard.* version 1.0.

8. **IETF Network Working Group.** *IAB Technical Comment on the Unique DNS Root.* s.l. : IETF, 2000. RFC 2826.

9. **EPCglobal Inc.** *Object Naming Service (ONS) Standard.* version 1.0.

10. —. *Certificate Profile Standard.*

11. **IETF Network Working Group.** *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile.* s.l. : IETF, 2002. RFC 3280.

12. —. *Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile.* s.l. : IETF, 2002. RFC 3279.

13. **EPCglobal Inc.** *Architectural Framework Document.*

14. *Issues in Supply Chain Management.* **Lambert, Douglas and Cooper, Martha.** 2000, Industrial Marketing Management, pp. 65-83.

15. WebSphere RFID Premises Server. *IBM.* [Online] http://www-306.ibm.com/software/pervasive/ws_rfid_premises_server/.

16. WebSphere Remote Server. *IBM.* [Online] http://www-306.ibm.com/software/webservers/remoteserver/.

17. mySAP SCM. *SAP.* [Online] http://www11.sap.com/solutions/business-suite/scm/index.epx.

18. Sun Java System RFID Software. *SUN.* [Online] http://www.sun.com/software/products/rfid/index.xml.

19. **Oracle.** RFID and Sensor-Based Services. [Online] http://www.oracle.com/technologies/rfid/index.html.

20. **GCI and IBM.** A Shared Vision for Transforming Business Processes.

21. *Supply Chain Management: an analytical framework for critical literature review.* **Croom, Simon, Romano, Pietro and Giannakis, Mihalis.** 2000, European Journal of Purchasing & Supply Management, pp. 67-83.

22. *Supply Chain Management: the idustrial organization perspective.* **Ellram, L.** 1, 1991, International Journal of Physical Distribution and Materials Management, Vol. 21, pp. 13-22.

23. *Using inventory for competitive advantage through supply chain management.*

Jones, T. and Riley, D. 5, 1985, International Journal of Physical Distribution and Materials Management, Vol. 15, pp. 16-26.

24. *Introduction to the special issue on global supply chain management.* **Lee, H. and Ng, S.** 3, 1997, Production and Operations Management, Vol. 6, pp. 191-192.

25. *Supply Chain Management: supplier performance and firm performance.* **Tan, K., Kannan, V. and Handfield, R.** 1998, International Journal of Purchasing and Material Management 34 (3), pp. 2-9.

III. Source Code

AlienController.java

```
package org.accada.reader.hal.impl.alien;

import com.alien.enterpriseRFID.notify.Message;
import com.alien.enterpriseRFID.notify.MessageListener;
import com.alien.enterpriseRFID.notify.MessageListenerService;
import com.alien.enterpriseRFID.reader.AlienClass01Reader;
import com.alien.enterpriseRFID.reader.AlienClass1Reader;
import com.alien.enterpriseRFID.reader.AlienReaderException;
import com.alien.enterpriseRFID.tags.Tag;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Vector;
import org.accada.reader.hal.HardwareAbstraction;
import org.accada.reader.hal.ControllerProperties;
import org.accada.reader.hal.DiagnosticInfo;
import org.accada.reader.hal.HardwareException;
import org.accada.reader.hal.Observation;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
```

```
import static org.accada.reader.hal.HardwareException.*;

/**
 *
 * @author Nektarios Leontiadis, Athens University of Economics and Business
 */
public class AlienController implements HardwareAbstraction, MessageListener{

    private static Log log;
    private static MessageListenerService listener;
    private static AlienClass1Reader reader;

    private static final String LOCK_CODE = String.valueOf(0xAB);
    private static final int EVALUATION_INTERVAL = 1000; //1sec

    private ControllerProperties props;
    private String readerId;
    private String [] activeAntennas;
    private int numOfServices;
    private String[] serviceList;
    private boolean continuousIsRunning;
    private int interval;
    private String prefix;
    private String mode;
    private String diagnostic;
    private String[] sourceIds;
```

```
static{
    log = LogFactory.getLog(AlienController.class);
    listener = new MessageListenerService(4000);
    reader = new AlienClass1Reader();
}

/** Creates a new instance of AlienContoller */
public AlienController(String readerId) {
    super();
    this.readerId = readerId;
    continuousIsRunning = false;
    sourceIds = null;
    initialize();
}

private void initialize(){
    String address,username,password,port;
    String errMessage;
    String interval;

    props = new ControllerProperties(readerId);
    if(reader!=null && reader.isOpen())
        reader.close();
    try {
        address = props.getParameter("ipAddress");
        username = props.getParameter("username");
        password = props.getParameter("password");
```

```
            port = props.getParameter("port");
            interval = props.getParameter("interval");
            this.interval = (interval == null)?EVALUATION_INTERVAL:Integer.parseInt(interval);
            activeAntennas = props.getParameter("activeAntennas").split(",");
            numOfServices = Integer.parseInt(this.props.getParameter("numberOfServices"));
            serviceList = new String[numOfServices];
            for (int i=0; i<numOfServices; i++){
                serviceList[i]= this.props.getParameter("service_"+(i+1));
            }
        } catch (Exception ex) {
            errMessage = "Couldn't initialize the reader:" + ex.getMessage();
            ex.printStackTrace();
            log.fatal(errMessage);
            return;
//            throw new HardwareException(errMessage, SERVICECODE_INITIALIZE, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_NOERROR, readerId);
        }

        try {

            reader.setUsername(username);
            reader.setPassword(password);
            reader.setConnection(address+":"+port);
            reader.open();
            reader.setMask(AlienClass1Reader.ALL_MASK);
        } catch (Exception ex) {
            ex.printStackTrace();
            errMessage = "Could not connect to the reader: "+ex.getMessage();
```

```
            log.fatal(errMessage);
        }
    }

    public Observation[] identify(String prefix, int estimateNoTags, String mode, String diagnostic)
throws HardwareException {
        return identify(null, prefix, estimateNoTags, mode, diagnostic);
    }

    /**
     *  This is used for synchronous and asynchronous identifies. In the latter, the taglist parameter
will
     *  be  filled with the tags that have been asynchronously read
     */
    public Observation[] identify(Message taglist, String prefix, int estimateNoTags, String mode, String
diagnostic) throws HardwareException {
        DiagnosticInfo info;
        HashMap <Integer, Vector> elements = new HashMap <Integer, Vector>();
        Observation [] observations;
        Observation observation;
        Vector temp;
        Tag [] tags;
        Tag tag;
        Long time;

        synchronized(reader){
            if(reader.isOpen()){
                observations = new Observation [activeAntennas.length];
```

```
try {
    if(taglist == null){
        if(prefix != null)
            reader.setTagMask(prefix);
        time = System.currentTimeMillis();
        tags = reader.getTagList();
    } else{
        tags = taglist.getTagList();
        time = taglist.getDate().getTime();
    }
    //Mute tags if requested
    if(mode!=null && mode.equals("MUTE"))
        reader.sleepTag((prefix!=null)?prefix:"");

    if(tags != null){
        for (int i = 0; i < tags.length; i++) {
            tag = tags[i];
            if(elements.containsKey(tag.getAntenna())){
                elements.get(tag.getAntenna()).add(tag.getTagID());
            }else{
                temp = new Vector();
                temp.add(tag.getTagID());
                elements.put(tag.getAntenna(), temp);
            }
        }
    }
```

```
                for (int i = 0; i < activeAntennas.length; i++) {
                    temp = elements.get(Integer.parseInt(activeAntennas[i]));

                    observation = new Observation(diagnostic);
                    observation.setReaderId(readerId);
                    observation.setSourceId(activeAntennas[i]);
                    observation.setTimestamp(time);
                    if(temp != null)
                        observation.setTagIds(temp);

                    observations[i] = observation;
                }
                return observations;
            } catch (Exception ex) {
                ex.printStackTrace();
                log.error(ex.getMessage());
                throw new HardwareException(ex.getMessage());
            }finally{
                try {
                    reader.setMask(AlienClass1Reader.ALL_MASK);
                } catch (AlienReaderException e) {
                    e.printStackTrace();
                }
            }
        } else
            throw new HardwareException("Not connected", SERVICECODE_IDENTIFY,
EXECUTIONCODE_COMMUNICATIONERROR, COMPLETIONCODE_NOERROR, readerId);
```

```
        }//synchronized
    }

    public void continuousIdentify(String prefix, int estimateNoTags, String mode, String diagnostic)
throws HardwareException {
        if(!listener.isRunning()){
            listener.setMessageListener(this);
            listener.startService();
        }
        try {
            if(prefix!=null)
                reader.setTagMask(prefix);
            // Set up Notification.
            // Use this host's IPAddress, and the port number that the service is listening on.
            reader.setNotifyAddress(InetAddress.getLocalHost().getHostAddress(),
listener.getListenerPort());
            reader.setNotifyFormat(AlienClass1Reader.XML_FORMAT); // Listener only supports XML messages
            reader.setNotifyTrigger("TrueFalse"); // Notify whether there's a tag or not
            reader.setNotifyMode(AlienClass1Reader.ON);

            // Set up AutoMode
            reader.autoModeReset();
            reader.setAutoStopTimer(interval);
            reader.setAutoMode(AlienClass1Reader.ON);

            this.prefix = prefix;
            this.mode = mode;
            this.diagnostic = diagnostic;
```

```
            continuousIsRunning = true;
        } catch (UnknownHostException ex) {
            ex.printStackTrace();
        } catch (AlienReaderException ex) {
            ex.printStackTrace();
        }
    }

    public Observation[] multiplexIdentify(String[] sourceIds, String prefix, int estimateNoTags, String
mode, String diagnostic) throws HardwareException {
        return multiplexIdentify(null, sourceIds, prefix, estimateNoTags, mode, diagnostic);
    }

    public Observation[] multiplexIdentify(Message taglist, String[] sourceIds, String prefix, int
estimateNoTags, String mode, String diagnostic) throws HardwareException {
        DiagnosticInfo info;
        Observation [] observations;
        Observation observation;
        HashMap <Integer,Vector> observed;
        Vector tagIds;
        Tag [] tags;
        Tag tag;
        Long time;
        HashSet <Integer> sources;

        synchronized(reader){
            if(reader.isOpen()){
```

```
observed = new HashMap <Integer,Vector>();
sources = new HashSet<Integer>();
for (int i = 0; i < sourceIds.length; i++) {
    sources.add(Integer.parseInt(sourceIds[i]));
}
try {
    observations = new Observation [sourceIds.length];
    if(taglist == null){
        if(prefix != null)
            reader.setTagMask(prefix);

        time = System.currentTimeMillis();
        tags = reader.getTagList();
    } else{
        tags = taglist.getTagList();
        time = taglist.getDate().getTime();
    }

    if(mode!=null && mode.equals("MUTE"))
        reader.sleepTag((prefix!=null)?prefix:"");

    if(tags != null){
        for (int i = 0; i < tags.length; i++) {
            tag = tags[i];
            if(sources.contains(tag.getAntenna())){
                if (observed.containsKey(tag.getAntenna())){
                    observed.get(tag.getAntenna()).add(tag.getTagID());
```

```
                    } else{
                        tagIds = new Vector();
                        tagIds.add(tag.getTagID());
                        observed.put(tag.getAntenna(),tagIds);
                    }
                }
            }
        }

        for (int i = 0; i < sourceIds.length; i++) {
            tagIds = observed.get(Integer.parseInt(sourceIds[i]));
            observation = new Observation(diagnostic);
            observation.setReaderId(readerId);
            observation.setSourceId(sourceIds[i]);
            observation.setTimestamp(time);
            if(tagIds != null)
                observation.setTagIds(observed.get(Integer.parseInt(sourceIds[i])));

            observations[i] = observation;
        }

        return observations;
    } catch (Exception ex) {
        ex.printStackTrace();
        log.error(ex.getMessage());
        throw new HardwareException(ex.getMessage());
    }finally{
```

```
                    try {
                        reader.setMask(AlienClass1Reader.ALL_MASK);
                    } catch (AlienReaderException e) {
                        e.printStackTrace();
                    }
                }
            } else
                throw new HardwareException("Not connected", SERVICECODE_IDENTIFY,
EXECUTIONCODE_COMMUNICATIONERROR, COMPLETIONCODE_NOERROR, readerId);
        }//synchronized
    }

    public void multiplexContinuousIdentify(String[] sourceIds, String prefix, int estimateNoTags, String
mode, String diagnostic) throws HardwareException {
        continuousIdentify(prefix, estimateNoTags,mode, diagnostic);
        this.sourceIds = sourceIds;
    }

    public void stopContinuousIdentify() throws HardwareException {
        try {
            reader.autoModeReset();
            listener.stopService();
            sourceIds = null;
        } catch (AlienReaderException ex) {
            ex.printStackTrace();
            throw new HardwareException("Could not stop continuous identify");
        }
        continuousIsRunning = false;
```

```
}

public boolean isContinuousIdentifying() throws HardwareException {
    return continuousIsRunning;
}

public void messageReceived(Message message) {
    System.err.println(message.getXML());
    try {
        if(sourceIds == null)
            identify(message, prefix, -1, mode, diagnostic);
        else
            multiplexIdentify(message, sourceIds, prefix, -1, mode, diagnostic);
    } catch (HardwareException ex) {
        ex.printStackTrace();
    }
}

public String[] getSourceIds() throws HardwareException {
    return activeAntennas;
}

public void setParameter(String param, String value) throws HardwareException {
    try {
        String [] paramNames = props.getParameterNames();
        for (int i = 0; i < paramNames.length; i++) {
            if(paramNames[i].equalsIgnoreCase(param)){
```

```
                    props.setParameter(param, value);
                    initialize();
                    break;
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            log.debug(ex.getMessage(),ex);
            log.error(ex.getMessage());
            throw new HardwareException(ex.getMessage());
        }
    }

    public String[] getParameterNames() throws HardwareException {
        try {
            return props.getParameterNames();
        } catch (Exception ex) {
            ex.printStackTrace();
            log.debug(ex.getMessage(),ex);
            log.error(ex.getMessage());
            throw new HardwareException(ex.getMessage());
        }
    }

    public String getParameter(String param) throws HardwareException {
        try {
            return props.getParameter(param);
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
            log.debug(ex.getMessage(),ex);
            log.error(ex.getMessage());
            throw new HardwareException(ex.getMessage());
        }
    }

    public String[] getServices() throws HardwareException {
        return serviceList;
    }

    public String[] getIdentifyModi() throws HardwareException {
        return new String [] {"ContinuousIdentify"};
    }

    public String[] getDeviceInfo() throws HardwareException {
        try {
            if(reader.isOpen()){
                return new String[] {reader.getReaderName(), reader.getReaderType(),
reader.getReaderVersion()};
            }
        } catch (AlienReaderException ex) {
            ex.printStackTrace();
            log.debug(ex.getMessage(),ex);
            log.error(ex.getMessage());
            throw new HardwareException(ex.getMessage());
        }
```

```
        return null;
    }

    public void programTagId(String idOld, String idNew) throws HardwareException {
        synchronized(reader){
            try {
                if(reader.isOpen()){
                    reader.setTagMask(idOld);
                    programTagId(idNew);
                }
            } catch (Exception ex) {
                try {
                    reader.setMask(AlienClass1Reader.ALL_MASK);
                } catch (AlienReaderException e) {
                    e.printStackTrace();
                }
                ex.printStackTrace();
                log.error(ex);
                throw new HardwareException(ex.getMessage());
            }
        }//synchronized
    }

    public void programTagId(String idNew) throws HardwareException {
        String currFunction;
        try {
            synchronized(reader){
```

```
            if(reader.isOpen()){
                currFunction = reader.getReaderFunction();
                reader.setReaderFunction(AlienClass1Reader.FUNCTION_PROGRAMMER);
                reader.programTag(idNew);
                reader.setReaderFunction(currFunction);
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        log.error(ex);
        throw new HardwareException(ex.getMessage());
    }
}

public void reset() throws HardwareException {
    try {
        initialize();
        synchronized(reader){
            reader.wakeTag("");
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        log.error(ex);
        throw new HardwareException(ex.getMessage());
    }
}
```

```
public void kill(String id) throws HardwareException {
    synchronized(reader){
        try {
            if(reader.isOpen()){
                reader.setTagMask(id);
                reader.lockTag(LOCK_CODE);
                reader.killTag(id+LOCK_CODE);
            }
        } catch (Exception ex) {
            try {
                reader.setMask(AlienClass1Reader.ALL_MASK);
            } catch (AlienReaderException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
            log.error(ex);
            throw new HardwareException(ex.getMessage());
        }
    }
}

public void conceal(String id) throws HardwareException {
    synchronized(reader){
        try {
            if(reader.isOpen()){
                reader.setTagMask(id);
                reader.sleepTag(id);
```

```
                }
            } catch (Exception ex) {
                try {
                    reader.setMask(AlienClass1Reader.ALL_MASK);
                } catch (AlienReaderException e) {
                    e.printStackTrace();
                }
                ex.printStackTrace();
                log.error(ex);
                throw new HardwareException(ex.getMessage());
            }
        }
    }


    /************** NOT IMPLEMENTED ***************/

    public int getBlockSize() throws HardwareException {
        throw new HardwareException("Not supported: getBlockSize", 0, EXECUTIONCODE_SERVICE_NOTSUPPORTED,
COMPLETIONCODE_NOERROR, readerId);
    }
    public byte[] readBytes(String id, int from, int length) throws HardwareException {
        throw new HardwareException("Not supported: readBytes", SERVICECODE_READ,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, COMPLETIONCODE_NOERROR, readerId);
    }

    public byte[] multiplexReadBytes(String[] sourceIds, String id, int from, int length) throws
HardwareException {
```

```
        throw new HardwareException("Not supported: multiplexReadBytes", SERVICECODE_READ,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, COMPLETIONCODE_NOERROR, readerId);
    }

    public void writeBytes(String id, int from, byte[] data) throws HardwareException {
        throw new HardwareException("Not supported: writeBytes", SERVICECODE_WRITE,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, COMPLETIONCODE_NOERROR, readerId);
    }

    public void multiplexWriteBytes(String[] sourceIds, String id, int from, byte[] data) throws
HardwareException {
        throw new HardwareException("Not supported: multiplexWriteBytes", SERVICECODE_WRITE,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, COMPLETIONCODE_NOERROR, readerId);
    }

}
```

IntermecIF5Controller.java

```
package org.accada.reader.hal.impl.intermec;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Vector;
import org.accada.reader.hal.ControllerProperties;
import org.accada.reader.hal.HardwareAbstraction;
import org.accada.reader.hal.HardwareException;
import org.accada.reader.hal.Observation;
import org.apache.commons.logging.Log;
import static org.accada.reader.hal.HardwareException.*;
import org.apache.commons.logging.LogFactory;

/**
 *
 * @author Nektarios Leontiadis, Athens University of Economics and Business
 */
public class IntermecIF5Controller implements HardwareAbstraction{

    private static Log log;
    private static final String LOCK_CODE = String.valueOf(0xBB);
    private static Socket socket;

    private ControllerProperties props;
    private BufferedReader in;
    private DataOutputStream out;
    private String readerId;
    private String activeAntennas[], activeAntennasString;
    private String currMask;
```

```
private int numOfServices;
private String address;
private int port;
private String[] serviceList;
private String readerInfo[];
private String eol;

static{
    log = LogFactory.getLog(IntermecIF5Controller.class);
}

/** Creates a new instance of IntermecIF5Controller */
public IntermecIF5Controller(String readerId) {
    super();
    this.readerId = readerId;
    initialize();
}

private void initialize(){
    String errMessage, line;

    props = new ControllerProperties(readerId);

    try {
        address = props.getParameter("ipAddress");
        port = Integer.parseInt(props.getParameter("port"));
        activeAntennasString = props.getParameter("activeAntennaIds");
        activeAntennas = activeAntennasString.split(",");
        eol = props.getParameter("endOfLineChars");
        if(eol==null)
            eol="\r\n";
        numOfServices = Integer.parseInt(this.props.getParameter("numberOfServices"));
        serviceList = new String[numOfServices];
        for (int i=0; i<numOfServices; i++){
            serviceList[i]= this.props.getParameter("service_"+(i+1));
        }
```

```
            connect();

            if(readerInfo == null){
                readerInfo = new String [] {"Intermec IF5 RFID reader"};

            }
        } catch (Exception ex) {
            errMessage = "Couldn't initialize the reader:" + ex.getMessage();
            ex.printStackTrace();
            log.fatal(errMessage);
        }
    }

    private final void connect() throws UnknownHostException, IOException, HardwareException{
        if(in==null || out==null || socket==null || !socket.isConnected()){
            String charsRead;
            socket = new Socket(address,port);
            in= new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out=new DataOutputStream(socket.getOutputStream());
            for(int i=0; i<8; i++)
                System.err.println(in.readLine());
        }
    }

    private final void disconnect(){
        try {
            if(socket!=null && socket.isConnected())
                socket.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private ArrayList <String> read(String prefix, String antennas) throws HardwareException{
        ArrayList <String> tags=new ArrayList <String>();

        try {
            String charsRead;
```

```
            int tagsRead=0;

            connect();
            synchronized(out){
                if(antennas!=null){
                    antennas = antennas.trim().replace(' ',',');
                    out.writeBytes("ATTRIBUTE ANTS="+antennas+eol);
                } else{
                    out.writeBytes("ATTRIBUTE ANTS="+activeAntennasString+eol);
                }

                do {
                    charsRead=in.readLine();
                    if(charsRead.equals("OK>"))
                        break;
                }while(true);

                out.writeBytes("READ ANT");
                if(prefix!=null && !prefix.trim().equals("")){
                    out.writeBytes(" WHERE EPCID="+prefix);
                }

                out.writeBytes(eol);

                do {
                    charsRead=in.readLine();
                    if(charsRead.startsWith("EVT:") || charsRead.trim().equals("")
||charsRead.equals("ERR") || charsRead.contains("NOTAG"))
                        continue;

                    if(charsRead.equals("OK>"))
                        break;

                    tags.add(charsRead);
                }while(true);
            }//synchronized
            return tags;
        } catch(Exception ex) {
```

```
            ex.printStackTrace();
            log.error(ex);
            throw new HardwareException(ex.getMessage());
        }
    }

    public Observation[] identify(String prefix, int estimateNoTags, String mode, String diagnostic) throws
HardwareException {
        ArrayList <String> tags;
        Observation [] observation = new Observation[activeAntennas.length];
        Vector [] tagIds = new Vector[activeAntennas.length];
        String [] line;

        for (int i = 0; i < activeAntennas.length; i++) {
            tagIds[i] = new Vector();
        }

        tags = read(prefix, null);
        for(String reading : tags){
            line = reading.split(",");
            //line[0]: antenna id
            //line[1]: epcid
            tagIds[Integer.parseInt(line[1])-1].add(line[0]);
        }

        for (int i = 0; i < activeAntennas.length; i++) {
            observation[i] = new Observation(diagnostic);
            observation[i].setReaderId(readerId);
            observation[i].setSourceId(activeAntennas[i]);
            observation[i].setTimestamp(System.currentTimeMillis());
            observation[i].setTagIds(tagIds[i]);
        }

        return observation;
    }

    public Observation[] multiplexIdentify(String[] sourceIds, String prefix, int estimateNoTags, String
mode, String diagnostic) throws HardwareException {
```

```
        ArrayList <String> tags;
        Observation [] observation = new Observation[activeAntennas.length];
        Vector [] tagIds = new Vector[activeAntennas.length];
        String [] line;
        StringBuilder antennas = new StringBuilder();
        final char SPACE = ' ';


        for (int i = 0; i < activeAntennas.length; i++) {
            tagIds[i] = new Vector();
        }

        for (int i = 0; i < sourceIds.length; i++) {
            antennas.append(sourceIds[i]);
            //in read() the spaces are going to be replaced with commas except from the last one which is
ommited
            antennas.append(SPACE);
        }

        tags = read(prefix, antennas.toString());

        for(String reading : tags){
            line = reading.split(",");
            //line[1]: antenna id
            //line[0]: epcid
            tagIds[Integer.parseInt(line[1])-1].add(line[0]);
        }

        for (int i = 0; i < sourceIds.length; i++) {
            observation[i] = new Observation(diagnostic);
            observation[i].setReaderId(readerId);
            observation[i].setSourceId(sourceIds[i]);
            observation[i].setTimestamp(System.currentTimeMillis());
            if(tagIds != null && tagIds.length!=0)
                observation[i].setTagIds(tagIds[Integer.parseInt(sourceIds[i])-1]);
        }

        return observation;
```

```
    }

    public byte[] readBytes(String id, int from, int length) throws HardwareException {
        return multiplexReadBytes(null, id, from, length);
    }

    public byte[] multiplexReadBytes(String[] sourceIds, String id, int from, int length) throws
HardwareException {
        String read=new String(),temp;
        String antennas;

        if(sourceIds!=null){
            antennas = sourceIds[0];
            for (int i = 1; i < sourceIds.length; i++) {
                antennas += ","+sourceIds[i];
            }
        } else{
            antennas = activeAntennasString;
        }

        try {
            connect();
            synchronized(out){
                out.writeBytes("ATTRIBUTE ANTS="+antennas+eol);
                in.readLine();
                out.writeBytes("READ STRING("+from+","+length+") WHERE EPCID="+id+eol);

                System.err.println("ATTRIBUTE ANTS="+antennas+eol);
                System.err.println("READ STRING("+from+","+length+") WHERE EPCID="+id+eol);

            }
            do {
                temp = in.readLine();
                if(temp.equals("OK>"))
                    break;
                try{
                read=new String(temp.split(",")[1].replace("\"",""));
                }catch(Exception e){
```

```
                    //If we have not read from a tag and we have an error or something else an exception
will be thrown
                    //and we'll get here
                    read = temp;
                }
            }while(true);
            return read.getBytes();

        } catch (Exception ex) {
            ex.printStackTrace();
            log.error(ex);
            throw new HardwareException(ex.getMessage());
        }
    }

    public void writeBytes(String id, int from, byte[] data) throws HardwareException {
        multiplexWriteBytes(null, id, from, data);
    }

    public void multiplexWriteBytes(String[] sourceIds, String id, int from, byte[] data) throws
HardwareException {
        String read;
        String antennas;
        StringBuilder command;

        //Build the write command
        command = new StringBuilder("WRITE ");
        //The format is WRITE STRING(addr,length)=data, ...
        //StringBuilder is very fast in concatenations
        for (int i = 0; i < data.length; i++) {
            command.append("STRING(");
            command.append(from+i);
            command.append(",1)=");
            command.append(data[i]);
            //if this is the last byte don't add a comma
            if(i+1 == data.length)
                break;
            command.append(", ");
```

```
        }
        command.append(" WHERE EPCID=");
        command.append(id);

        if(sourceIds!=null){
            antennas = sourceIds[0];
            for (int i = 1; i < sourceIds.length; i++) {
                antennas += ","+sourceIds[i];
            }
        } else{
            antennas = activeAntennasString;
        }

        try {
            connect();
            synchronized(out){
                out.writeBytes("ATTRIBUTE ANTS="+antennas+eol);
                in.readLine();
                out.writeBytes(command+eol);
                read = in.readLine();
                System.err.println(command);
                System.err.println(read);
            }

        } catch (Exception ex) {
            ex.printStackTrace();
            log.error(ex);
            throw new HardwareException(ex.getMessage());
        }
        if(read.contains("WRERR"))
            throw new HardwareException("Write incomplete", SERVICECODE_WRITE,
COMPLETIONCODE_EXECUTIONERROR, EXECUTIONCODE_INVALIDPARAMETER, readerId);

    }

    public String[] getSourceIds() throws HardwareException {
        return activeAntennas;
    }
```

```
public void setParameter(String param, String value) throws HardwareException {
    try {
        String [] paramNames = props.getParameterNames();
        for (int i = 0; i < paramNames.length; i++) {
            if(paramNames[i].equalsIgnoreCase(param)){
                props.setParameter(param, value);
                disconnect();
                initialize();
                break;
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        log.error(ex);
        throw new HardwareException(ex.getMessage());
    }
}

public String[] getParameterNames() throws HardwareException {
    try {
        return props.getParameterNames();
    } catch (Exception ex) {
        ex.printStackTrace();
        log.debug(ex.getMessage(),ex);
        log.error(ex.getMessage());
        throw new HardwareException(ex.getMessage());
    }
}

public String getParameter(String param) throws HardwareException {
    try {
        return props.getParameter(param);
    } catch (Exception ex) {
        ex.printStackTrace();
        log.error(ex);
        throw new HardwareException(ex.getMessage());
    }
```

```
    }

    public String[] getServices() throws HardwareException {
        return serviceList;
    }

    public String[] getDeviceInfo() throws HardwareException {
        return readerInfo;
    }

    public int getBlockSize() throws HardwareException {
        return 2;
    }

    public void programTagId(String idOld, String idNew) throws HardwareException {
        String read;

        try {
            connect();
            synchronized(out){
                out.writeBytes("ATTRIBUTE ANTS="+activeAntennasString+eol);
                in.readLine();
                out.writeBytes("WRITE EPCID="+idNew+" WHERE EPCID="+idOld+eol);
                read = in.readLine();
                System.err.println(read);
            }
            do {
                read = in.readLine();
                System.err.println(read);
                if(read.equals("OK>") || read.contains("ERR"))
                    break;
            }while(true);
            if(read.contains("WRERR"))
                throw new HardwareException("Write programTagId", SERVICECODE_WRITE,
COMPLETIONCODE_EXECUTIONERROR, EXECUTIONCODE_INVALIDPARAMETER, readerId);

        } catch (Exception ex) {
            ex.printStackTrace();
```

```
            log.error(ex);
            if(ex instanceof HardwareException)
                throw (HardwareException)ex;
            throw new HardwareException(ex.getMessage());
        }
    }

    public void programTagId(String idNew) throws HardwareException {
        programTagId("000", idNew);
    }

    public void reset() throws HardwareException {
        disconnect();
        initialize();
    }


    /**
     *  NEEDS REFINEMENT
     */
    public String[] getIdentifyModi() throws HardwareException {
        return new String [] {"NON_MUTE"};
    }


    /************** NOT IMPLEMENTED ***************/

    /**
     * This command should work according to the reader's manual. During the tests it was apparent
     * that it didn't work and so it was deactivated. The code that should work is commented out.
     */
    public void kill(String id) throws HardwareException {
        throw new HardwareException("Not supported", 0, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, readerId);
//        try {
//            //first lock (protect) then kill
//            connect();
//            synchronized(out){
```

```
//                  out.writeBytes("PROTECT ON PERMANENT WHERE EPCID="+id+" PASSWORD="+LOCK_CODE+eol);
//                  System.err.println("PROTECT ON PERMANENT WHERE EPCID="+id+" PASSWORD="+LOCK_CODE+eol);
//                  if(!in.readLine().startsWith("OK")){
//                      System.err.println();
//                      throw new HardwareException("Kill error",SERVICECODE_WRITE,
COMPLETIONCODE_EXECUTIONERROR, EXECUTIONCODE_TAGCOMMUNICATIONERROR, readerId);
//                  }
//                  out.writeBytes("KILLTAG WHERE EPCID="+id+eol);//+" PASSWORD="+LOCK_CODE+eol);
//              }
//              if(!in.readLine().startsWith("OK"))
//                  throw new HardwareException("Kill error",SERVICECODE_WRITE,
COMPLETIONCODE_EXECUTIONERROR, EXECUTIONCODE_TAGCOMMUNICATIONERROR, readerId);
//          } catch (Exception ex) {
//              log.error(ex);
//              if(ex instanceof HardwareException)
//                  throw (HardwareException)ex;
//              else{
//                  ex.printStackTrace();
//                  throw new HardwareException(ex.getMessage());
//              }
//          }
      }

      public void continuousIdentify(String prefix, int estimateNoTags, String mode, String diagnostic)
throws HardwareException {
          throw new HardwareException("Not supported", 0, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, readerId);
      }

      public void multiplexContinuousIdentify(String[] sourceIds, String prefix, int estimateNoTags, String
mode, String diagnostic) throws HardwareException {
          throw new HardwareException("Not supported", 0, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, readerId);
      }

      public void stopContinuousIdentify() throws HardwareException {
          throw new HardwareException("Not supported", 0, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, readerId);
```

```
    }

    public boolean isContinuousIdentifying() throws HardwareException {
        throw new HardwareException("Not supported", 0, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, readerId);
    }

    public void conceal(String id) throws HardwareException {
        throw new HardwareException("Not supported", 0, COMPLETIONCODE_NOERROR,
EXECUTIONCODE_SERVICE_NOTSUPPORTED, readerId);
    }
}
```


ATHENS UNIVERSITY OF ECONOMICS & BUSINESS
ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ