DISTRIBUTED INTELLIGENCE SYSTEMS FOR DEVICE INTEGRATION AND CONTROL

by

Kevin Thomas
Bachelor of Engineering, Anna University, 2005

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School of Engineering Science

SIMON FRASER UNIVERSITY

Spring 2010

APPROVAL

Name: **Kevin Thomas**

Degree: **Master of Applied Science**

Title of Thesis: **Distributed Intelligence Systems for Device Integration and Control**

Examining Committee:

Chair: **Dr. Mirza Faisal Beg**

_____

Dr. William A. Gruver
Senior Supervisor

_____

Dorian Sabaz
Supervisor

_____

Dr. Ljiljana Trajkovic
Supervisor

_____

Dr. Craig Scratchley
Internal Examiner

Date Defended/Approved: _____April 20, 2010_____

Declaration of Partial Copyright Licence

ABSTRACT

Over the last quarter of a century, society has moved towards an immersion into digital technology. Despite the sophistication of current technology, integration and interoperability between digital and non-digital devices and equipment is both an existing problem, and a growing one. The root cause is the conventional paradigm based on highly centralized platforms for integration and operability.

This thesis proposes an alternate approach for device control using a Distributed Intelligence System (DIS), presenting it as a viable alternative that enables entities to work together in an intelligent manner to achieve user-defined global and local system objectives. It integrates prior research efforts to develop hardware and software for a real-world DIS, to form a complete system architecture, and applying the same to a practical scenario of a simple security system. Additionally, it examines some of the practical challenges and obstacles experienced over the entire systems development cycle of developing a DIS.

Keywords: peer-to-peer, P2P, distributed intelligence systems, holonic, distributed control, holon, holonic system

Subject Terms: systems engineering, systems integration, cybernetics

DEDICATION

To:

My only true and eternal God and Father, revealed to all humanity as Jesus Christ His only begotten Son, Alpha and Omega, the Grand Weaver of all life and the Ultimate Systems Architect and Engineer, who brought into being by the power of His Spirit all that exists with three words: "Let there be…"

And there was.

ACKNOWLEDGEMENTS

To Dr. William A. Gruver: Thank you for taking me on as a student, and giving me the opportunity to work with you to accomplish all that I have been able to.

To Dorian Sabaz: Thank you for your unwavering support, guidance and patience in showing me what it means to set the bar high, and jump.

To Dr. Ljiljana Trajkovic: Thank you for the wonderful lab facilities you allowed me to use during the course of my research work, and your dedication in your teaching.

To Colin: Thank you for all your prior work on HTP, that set proved invaluable in my own work, for your help and encouragement.

To my friends in SFU Engineering Science: Thank you for all your friendship and help over the last 3.5 years of my MASc program.

To my brothers and sisters at SFU C4C and UBC C4C: Thank you for all that you have been to me!

To Rick and Barb Pearson: Thank you for being my parents-away-from-home, for setting the Gold Standard to which I aspire, through your hospitality, love and support during my time here.

To my beloved Father, Mother and sister: Thank you for all your encouragement, patience and support during the uphill battle of my Master's degree.

To my great God and Heavenly Father: My utmost efforts to say 'Thank You' for Your faithfulness and all that You have taught me would be stammering at best. There could not have been a greater lesson throughout the whole experience of graduate school, than the assurance that whatever happens, I am and always will be my Father's son.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: SYSTEMS ENGINEERING

1.1 Introduction

Since the inception of the Information Age over the last quarter of a century, society has moved towards an increasing immersion into digital technology. Despite the sophistication of current state-of-the-art, integration and interoperability between digital and even non-digital devices and equipment remains an existing and growing problem. The root cause is the paradigm that designs technology systems such that they are reliant upon highly centralised frameworks for integration and operability.

It is possible to formulate a strong argument for the effect of highly hierarchical and centralised order of human societies upon a correspondingly centralised sociological structure, feeding into the political and therefore mechanical, and eventually technological inventions and methodologies of the present. Challenges have risen against this strong reliance upon centralised structures from time to time. For example, ARPANET, the original framework of the internet, was essentially a peer-to-peer network [1]. However, in what could be ascribed to a combination of sociological and technological forces, this original system later had forms of centralisation overlaid upon it (e.g.., servers), forcing it to support an inherently different architecture.

Breaking long established paradigms, such as systems reliant upon centralised control, is a challenge in itself. The advent of modern electronics resulted in the commitment of great effort into the construction of highly centralised systems, such as web servers, file servers, centralised databases and master controllers in automation, to name a few. One of the consequent problems of this established paradigm manifest in the present state of affairs, is the design of electronic components and devices such as CPUs, printers and manufacturing equipment to work within the context of this centralised architecture. Thus, the development of a system operating on a fundamentally different paradigm, while using technology inherently centralised nature, requires one to make compromises, implement inefficient strategies and constrain architectural decisions.

This thesis investigates these issues, proposing an alternate approach using a Distributed Intelligence System (DIS) [70] that employs a fully distributed, peer-to-peer architecture. Through an implemented example of a DIS, it aims to show that such a system is a viable alternative that enables entities to collaborate, co-ordinate and co-operate with each other in an intelligent manner to achieve user-defined global and local system objectives. It also examines the modifications required to current technology to implement a DIS, and possible avenues of carrying them out.

1.2 Types of Systems Today

The development of contemporary cybernetic theory [2, 3, 4, 5] in the early 20^{tth} century found a major field of application in the implementation of systems for control of various equipment and devices. The progressive shift in the primary controlling and signalling mechanism from mechanical to electrical resulted in devices of greater complexity and capability. It consequently became possible to progress from stand-alone devices to the interconnection and integration of devices to form large systems with the purpose of fulfilling a particular objective. This affected systems developed and implemented for the purpose of oversight and fulfilment of the objective. It was required that they be able to provide increased functionality, and handle the increasing complexity of managing the underlying interactions of many devices working together. Systems architected for control and oversight purposes underwent significant changes in order to meet these imposed constraints.

Systems architecture began to assume a centralised form, relying upon an underlying framework that concentrated command, control, communications and intelligence within controller entities tasked with managing a large system of essentially ‘dumb’ devices. This is a ‘Master/Slave’ or ‘Client/Server’ [6 - 8] framework. Later attempts at proposing alternatives to highly centralised Client/Server systems involved the use of frameworks where entities were potentially functionally or geographically distributed, communicating with each other over a network using messages. These solutions supposed that such a ‘Distributed System’ [6, 7, 9-11] might overcome or mitigate some of the problems experienced when constructing large systems using centralised frameworks. Systems implementing such solutions saw some perceptible improvements, but they were not of the significant magnitude anticipated; their

fundamental architecture retained aspects of centralisation and consequently, the problems associated with the same.

1.2.1 Drawbacks of Centralised Systems

Along with growth in size and complexity, limitations emerge in the usefulness of designing systems using a centralised framework, shown as an example in Figure 1.1 [12] and outlined below.

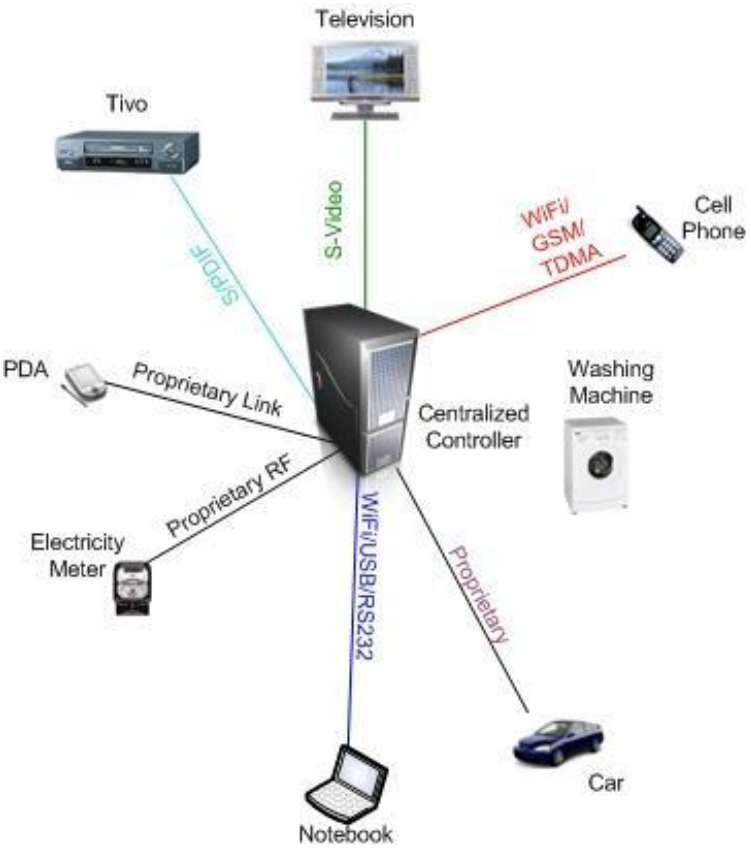

Figure 1.1: Example of centralised framework for controlling devices.

1. Controller Capability

Any upward scaling in addition of devices or device complexity requires a corresponding increase in sophistication of a controller's capabilities [13, 14]. Both scenarios demand that a controller entity become increasingly powerful in terms of

processing power; speed of response; detailed knowledge acquisition of devices under its jurisdiction; decision-making and communication. Addition of controller entities may mitigate some of the load experienced over the short term, but is still a temporary solution and does not address the underlying shortcomings. Controllers operating within a centralised framework cannot accommodate limitless expansion of such capabilities. Upward scaling degrades controller performance and negatively affects system operation as a whole.

2. Resource Management

Proper operation of a system is dependent on the availability of resources in sufficient quantities, such as storage space, bandwidth for communication and available energy, among others [15]. Controllers tasked with exercising control and oversight must manage finite resources in such a manner that allows for smooth operation, while provisioning appropriate reserves for unexpected situations. A centralised system, where new devices do not contribute to the pool of resources and depend on controller entities to manage and provide resources, is thus limited in its capability for growth.

3. Robustness

Increasing complexity and interconnection of devices to form large systems increased the possibility for failure during operation. As the paradigm of constructing systems moved towards the use of a centralised framework, sophistication of controller entities allowed them to take increasing responsibility for the robustness of the system. However, focusing increasing concentration of capability and responsibility within intelligent controllers supervising ‘dumb’ devices became a flaw. Failure of a controller left the devices under its jurisdiction stranded and degraded system operation [16]. Failure of a number of controllers severely impeded system operation or brought it to a halt.

4. Cost

Increasing reliance upon a sophisticated centralised framework also revealed incurred cost as a drawback during initial implementation as well as expansion phases [17, 18]. Implementation of the system in its initial form required the prior existence of a complete infrastructure (encompassing services such as controllers, communications, redundancy, storage, administration, maintenance and support among others) before devices could operate together. With increasing application to realising large systems,

the initial investment cost became a significantly unappealing factor. Furthermore, future expansion required additional investment in supporting infrastructure for new equipment before integration with the existing system could occur. Cost became prohibitive where regular maintenance and upgrading of several components of large systems had to be undertaken.

5. Security

It is required that systems for important applications be kept secure from unauthorised or hostile access, that intends to compromise part or all of system operation. Examples include, but are not restricted to, applications developed for energy and utilities, medical and healthcare, defence, manufacturing, business and commerce, finance, education, transportation, law and government. As large systems such as these became progressively reliant upon centralised frameworks, securing them became correspondingly difficult; the potential for inflicting damage if compromised by an intruder paralleled increasing sophistication of entities in the system.

6. Administration

System administrators require regular generation and presentation of relevant information pertaining to system events, device and controller status, resource management, overall system performance and other details. In addition, modification of the system configuration may be necessary from time to time. The trend in systems utilising centralised frameworks is to provide dedicated entities at which such services are concentrated. However, the difficulty of administration of a centralised system increases with scale [17]. Increasing flow of information to administrative entities burdens finite system resources and capabilities. Modifications that affect system operation make administration an increasingly complex decision-making process, especially in situations requiring numerous changes throughout the system. Administrative changes require propagation of instructions back to controller entities, placing further overhead on the system infrastructure. Cost extracts a significant penalty upon centralised administration, during initial implementation and future scaling of the same.

Table 1.1 summarises the discussion thus far.

Table 1.1: Drawbacks experienced in large centralised systems.

Centralised Feature	Drawback in a Centralised Framework	
Controller Capability	Increasing central sophistication needed	
Resource Management	Burden on contollers to manage limited resources	
Robustness	Failure of controllers leaves 'dumb' devices stranded	
Cost	Large initial investment in supporting infrastructure	
Security	**Compromised Feature**	**Effect**
	Controller Entity	Local disruption or halt of device operation
	Communications	Monitoring of system-wide communications
	Data	Access to system information and records
	Administration	Modification of system-wide operation
	Energy sources	Reduced performance or complete shutdown
Administration	Overhead on system infrastructure; increasing complexity of administrative changes	

1.2.2 Examples

The application of centralised frameworks to process automation systems illustrates the progression of centralised systems development and its implications. Direct Digital Control (DDC) [19, 20, 21] and more sophisticated microprocessor-based Programmable Logic Controllers (PLCs) [19, 22] became common in process automation. These used an industrial computer to realize in software the input, decision-making and output instructions that replaced centralised electrical controllers. Though a significant improvement, a PLC still functions as an intelligent 'master' entity in a centralised framework, controlling many essentially 'dumb' devices. The independent

introduction of the Distributed Control System (DCS) [23, 24, 25] increased sophistication of then existing automation system capabilities, and added new functionality. The structure of a DCS is such that one or more functionally or geographically distributed controllers control various sub-systems. However, it essentially retains the highly centralised framework of previous systems in terms of the features considered previously.

In order to support such functionality, communication infrastructure reflected a correspondingly centralised approach. This was reflected in the framework of industry-standard protocols such as Modbus [26]; Building Automation and Control Network (BACNet) [27, 28]; Process Field Bus (PROFIBUS) [29], Controller Area Network (CAN) [30, 31, 32] and DeviceNet [33, 34], among others. The distinctive feature was the incorporation of a main high capacity, high-speed bus or 'backbone' for routing all communications between controllers, operator stations, and other entities on the network. In addition, some communication frameworks incorporated 'master' entities that managed rights and privileges, sequenced the order of communications and so on. The main bus or a master entity represented points of failure, which if compromised, significantly damaged the operation and achievement of the system objective.

Products in the field of consumer electronics, appliances and mobile devices originally functioned as stand-alone entities. Later development by manufacturers and vendors for device networking and control gave rise to proprietary frameworks and standards, becoming a barrier to interoperability. One of the recent initiatives taken to address this challenge is Universal Plug and Play (UPnP) [35]. This is an "industry initiative designed to enable simple and robust connectivity among consumer electronics, intelligent appliances and mobile devices from many different vendors" [35]. Although the proposed framework is detailed and comprehensive [36-39], it relies upon some of the features of centralised systems discussed previously. One of these is the requirement for a working communications infrastructure providing services such as 'access points' that devices can connect to. Another feature is the necessity for controller entities known as 'control points' that devices must register with before being able to advertise their existence and services provided to other devices in the system. Although much of the framework takes on the form of a distributed system, these elements represent points of vulnerability, failure of which can severely degrade device interoperation or render it non-functional.

These examples show that the difficulties experienced when centralised systems become larger and increase in complexity are recognizable. These bottlenecks can lead to future reoccurrences of present-day problems as large systems are constructed. A fundamental re-consideration of the underlying framework is required.

1.2.3 Distributed Intelligence System

Attempts at overcoming the problems of centralised systems focused on modifying the framework in order to make it more distributed, both geographically and functionally. Although solutions were attempted, many proposed alternatives retained underlying vestiges of centralised frameworks. Thus, although some improvement was seen, these 'distributed systems' experienced the same problems that they were attempting to solve. It required the emergence of peer-to-peer systems employing a fundamentally different framework, as a clue to the beginnings of a potentially viable solution.

1.2.4 Peer-to-Peer Systems

Current efforts to provide new services and applications over various centralised communications networks, present requirements and challenges. Meeting growth in demand will require capabilities that the limitations of client/server frameworks will be unable to meet in future. The emergence of peer-to-peer (P2P) applications [40 - 43] on the Internet, and their demonstrated performance capability as highly scalable systems, presents a starting point for a new framework. A main design principle underlying successful P2P frameworks is their fully distributed and self-organizing [43] capability. Steinmetz and Wehrle provide a working definition of a peer-to-peer system as follows:

"…a self-organizing system of equal, autonomous entities (peers) [which] aims for the shared usage of distributed resources in a networked environment avoiding central services. In short, it is a system with completely decentralized self-organization and resource usage." [44]

Although P2P technology is not new, the large-scale demonstrated benefits of distributed resources [44] and distributed self-organisation [44] gained attention with hybrid P2P systems such as Napster [1]. Recent applications relate to file sharing and P2P communications, such as BitTorrent [1 - 46] and Skype [1, 42, 51]. Due to their success [48 - 55], there is consequently increasing consideration to harnessing the

framework for construction of other large-scale applications [56 - 69]. Current P2P systems provide potential advantages over client-server frameworks in some respects, but also present challenges in others [44]. Table 1.2 on page 11 outlines some of the challenges.

1.2.5 Extension from P2P to DIS

The existing framework of a P2P system provides a useful basis for the incorporation of additional functionality, necessary for the application under consideration. In addition to being fully distributed in terms of communications infrastructure, resources and self-organization, it will also require that intelligence capabilities be provided to all participating entities in the framework. The inclusion of this additional requirement is to allow all entities to work together in an intelligent fashion, to exert control within the system for achievement of its overall objective as defined by the user. Other challenges outlined in Table 1.2 will also require consideration and formulation of appropriate solutions. This enhanced framework is a Distributed Intelligence System (DIS) [70], which is a subset of the more general Distributed Systems Paradigm. Figure 1.2 [12] illustrates an example of a P2P framework extended for controlling devices.

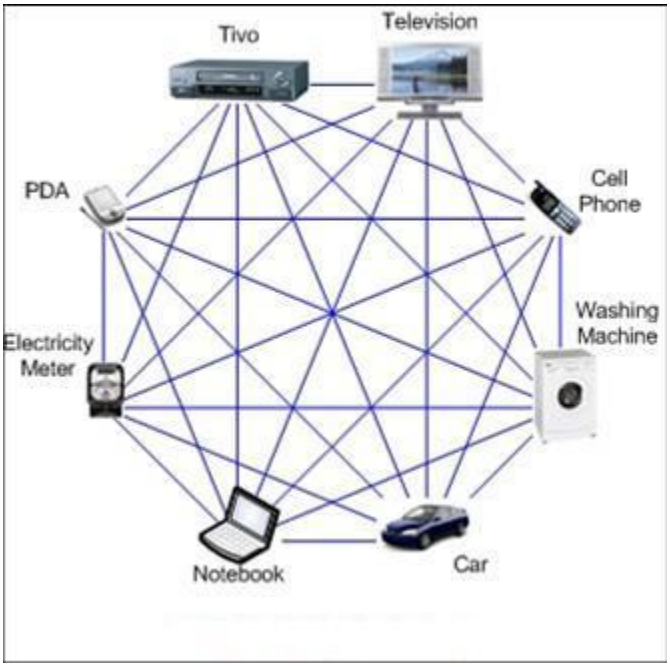

Figure 1.2: Example of P2P framework for controlling devices.

1.2.6 Examples

Interest in systems using distributed intelligence spans almost three decades. Prior proposals for such systems, for example in air fleet control [71], were eventually implemented upon centralised frameworks. Current interest occurs in areas such as Ubiquitous Computing and Ambient Intelligence (AmI) [72]. One recently proposed framework is Ambiance [73], intended for use on mobile devices, such as those in Wireless Sensor Networks (WSN). However, Ambiance centralises important functionality, relying on the use of a web client and server for proper operation. Although research into distributed intelligence is ongoing, constructing a DIS upon an underlying P2P framework is a relatively new topic.

Table 1.2: Benefits and challenges presented by the P2P framework.

Feature	Benefit Provided	Challenge Presented
Controller capability	Low intelligence capabilites	Intelligent functioning of entities
Resource availability	Entities contribute resources to pool	Identifying and making available free resources
Cost	Participating entities supply infrastructure capabilities	Open infrastructure variations as entities enter and leave system
Security	No server holding all system information	Providing uniform security to all entities
Robustness	No central point of failure to severely impede system function	Ensuring dynamic adaptation to changing environment
Administration	No central point of administration	Accurate reporting and configuration

1.3 Thesis Description

The work undertaken in this thesis attempts to demonstrate that a DIS using an underlying P2P framework provides a feasible avenue for engineering large systems that do not face the problems plaguing the large centralised systems of today. The following chapter presents a detailed consideration of the nature of a DIS framework, its requirements, and proposes an example application to a simple security system. Following this is an attempt to implement the proposed design as a working system, and a description of the process required to engineer various working components. An analysis stage examines the outcome of the implementation attempt, and the various practical challenges encountered during the process. The work concludes with a consideration of the lessons learned during the course of the endeavour, and application to building a better DIS in future.

CHAPTER 2: SYSTEM ARCHITECTURE AND DESIGN

2.1 Introduction

This chapter will discuss the system architecture and design of the intended system in some detail, with specific emphasis upon outlining the Distributed Intelligent Systems (DIS) architecture paradigm. Other possible architectures will be briefly discussed, along with the relative motivations that one architectural approach may have over another. Particular attention will be placed upon the interplay between local and global objectives of devices within a distributed control environment. This will be followed by a discussion of design methodology regarding intelligent devices existing within the DIS architecture. Specifically, it will be introduced, how using discrete mathematical structures like lattices can assist in the organizational requirements for a DIS. In recent years, Lattice Theory has been making inroads into many fields in the guise of Formal Concept Analysis (FCA) [74]. Briefly, FCA provides an ontological construct via lattices, which can be used to organize a domain of objects and their corresponding range of properties. In addition, examination of how these principles will be incorporated into the design of the Holonic Technology Platform [75] (HTP) will also be outlined. In order to show how the principles and paradigms developed during the course of discussion apply in practice, all of the above will be applied to a real-world application of a DIS system used for a simple security system.

2.2 Architecture Selection

The word ‘architecture’ in its most general sense pertains to style and form [76]. Applied to the context of systems engineering, the system architecture process deals with the abstract, high-level architecture of the system.. The current range of choices of architecture may be expressed via the two extremeum of the known spectrum of architectures::

- Fully centralised system architecture
- Fully distributed system architecture

The first of these pair, the fully centralised architecture, better known by its more common metonymy ‘Client/Server Architecture’, is how much of today’s software and hardware systems are implemented. The emphasis of the architecture is towards enabling concentration of various functions of the system at these central elements and dependence of peripheral elements upon central elements in order to function [77].

The emphasis on the other extrememum of software system styling (Fully distributed architecture) is towards the decomposition of system responsibilites and functions to the individual elements in the system.

Between these two extremums there are a plethora of architecuture types [40, 78]. As an example, one such system architecture is known as a ‘Decentralised System’. While it may be debated that such architecture qualifies towards the category of distributed systems, a legitimate case may also be put forward that a decentralised system may be composed of multiple server entities which may consitute areas of centralisation that work together as a co-ordinated group. The case may be put forward that this is essentially a multi-tier client/server system which falls into the spectrum bounded by the two formerly specified architecture paradigms, though a lesser form of client/server architecture. Figure 2.1 illustrates these various paradigms of architecture.

The discussion of architectural paradigms thus far lends itself to a consideration of the essential difference, between the more centralised system architectures that have been described and that proposed in this thesis, a Distributed Intelligence System (DIS). Specifically, this focuses on the difference between the tendency of centralised systems to group all devices in a system together as one class, and treat the entire goal space as one large indistinguishable set of requirements, as opposed to splitting this goal space into smaller goal sub-spaces and constructing some kind of set of operations that can organize the devices more effectively. A crude but perhaps applicable analogy is how a vector space can be split into equivalent classes, for instance, one of which is the kernel of the space, and that there exist linear operations connecting these sub-spaces together [79]. However in a systems scenario, instead of vectors we have the set of devices, where there arises a set of operations that linking these devices together. These operations can be partitioned on a broad scale into two juxaposed sets of goals, that is, local and global.

Another difference in architecture is intelligent management of the system while handling potentially conflicting local and global goals, unlike simple distributed systems

where the elements of the system may be endowed with very little understanding of how they relate and function with each other in their immediate vicinity as well as with a system-wide perspective in mind.

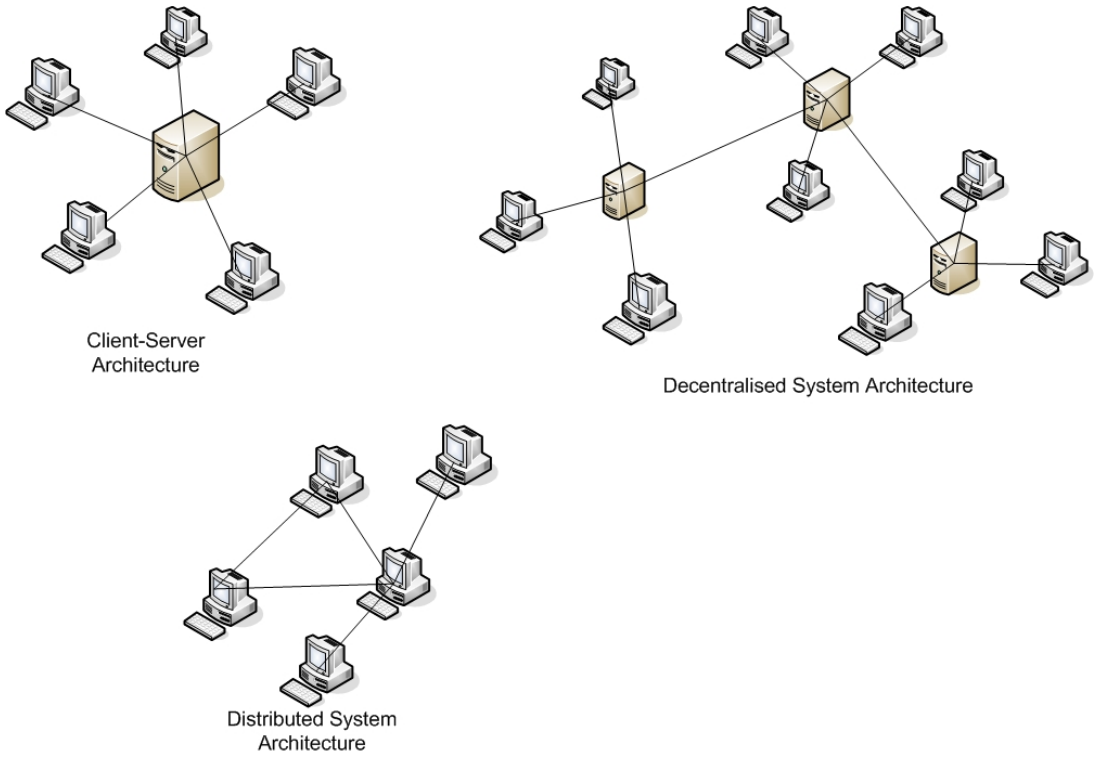

Figure 2.1: Centralised, Decentralised and Distributed Systems architecture.

These two criteria, that is, the partitioning of the goal space and the intelligence aspect, will be targeted for incorporation into the description of the system architecture, differentiating it from characterization as merely centralized or distributed system architecture. As an illlustration of these ideas, the application of a security system was used and developed as a working model to illustrate the points discussed thus far. Figure 2.2 shows a simple security system consisting of three devices – a motion sensor, network camera and Radio Frequency Identifier (RFID) tag reader, along with an optional monitoring device for a system administrator. The security system functions with a global user-specified objective of monitoring the movement of people or objects that have been labelled with RFID tags within a secured environment.

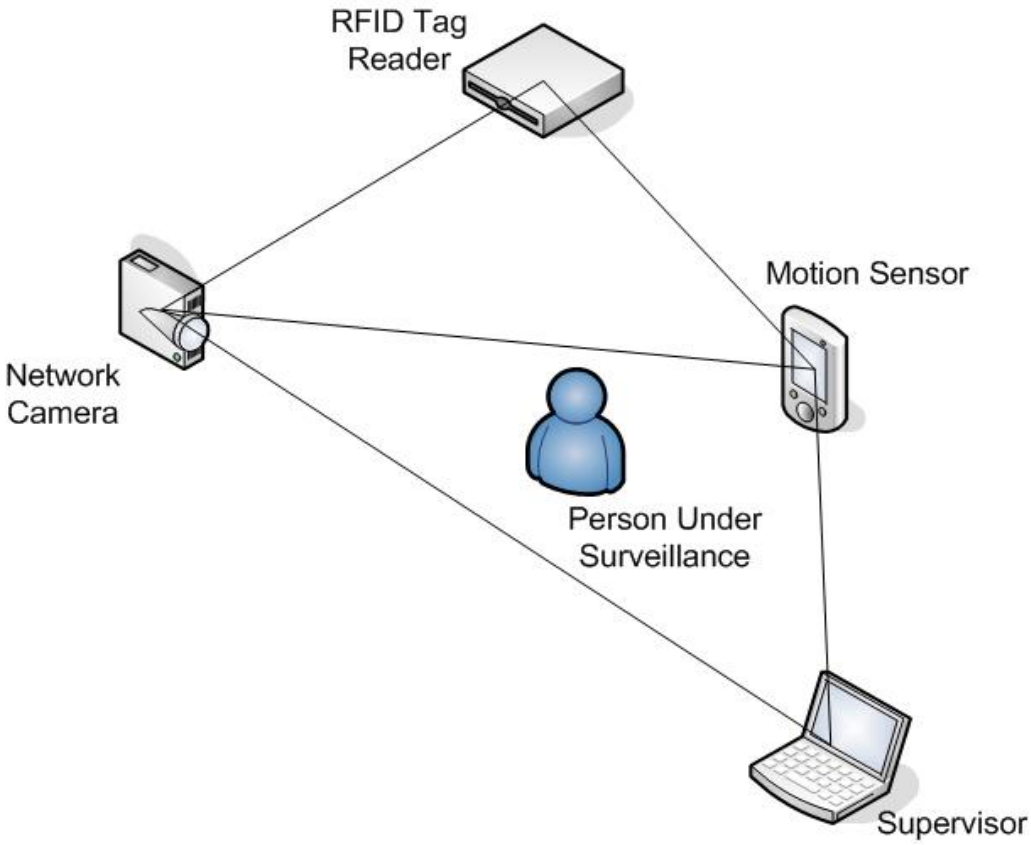

Figure 2.2: Distributed system architecture of a security system application.

The system's DIS architecture is demonstrated as devices work together to form the complete system and meet their user-specified objectives. Each device carries out its local security-related responsibilities, and also collaborates, co-ordinates and co-operates with other devices in its vicinity to meet the larger objective as a group – consistent monitoring of the movement of objects that have been labelled with an RFID tag.

Immediately, it can be noticed the goal partitioning aspects are split into local responsibilites, that is, those responsibilities that are required to maintain the device attached, and the system responsibilities, which in this case is the simple objective of surveillence. The intelligence aspect is manifested by how each device works with its adjacent neighboring devices.

For the purpose of discussion, a functional unit within the system architecture (e.g. a device within the security system that was described) will be referred to as a

‘node’ [80]. With respect to the interactions that must be conducted among them in order to fulfil their roles within the system, these nodes will be referred to as ‘peers’ [81].

2.3 Architecture Considerations and Requirements

A node in this DIS, illustrated in Figure 2.3, is comprised of the hardware and software that enable it to function. The hardware is abstracted as any generic device that accepts input for the purpose of control from a software entity residing on the same device, and supplies output regarding its current state of operation to the same. This software driving this device will be referred to as an ‘agent .’ The agent takes responsibility for the operation of the device within its local environment. It performs this intra-device control based on specified local objectives and rules of operation. The agent is in turn governed by another software entity also residing on the device, which will be referred to as a ‘holon’ [82]. The holon takes responsibility for the interaction of its node with other nodes, i.e. in the system. It performs this inter-device control within the context of specified ‘global’ or system-wide objectives. In this regard, it will also accept reporting from the agent regarding the state of the device and in turn supply the agent with any relevant information that it may require for its own operation.

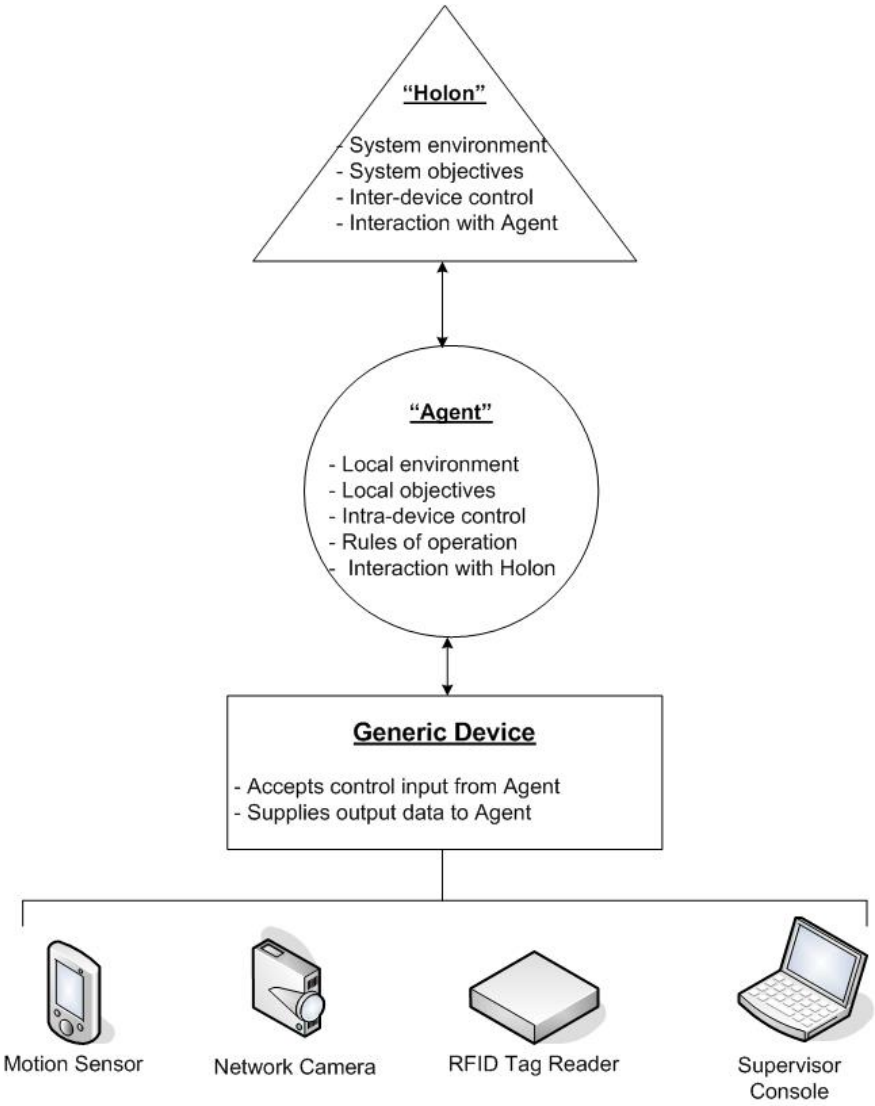

Figure 2.3: Abstract representation of a generic node.

A more detailed discussion of the architectural relationship between the holon and agent and the provisions that must be made within the proposed system references Figure 2.4.

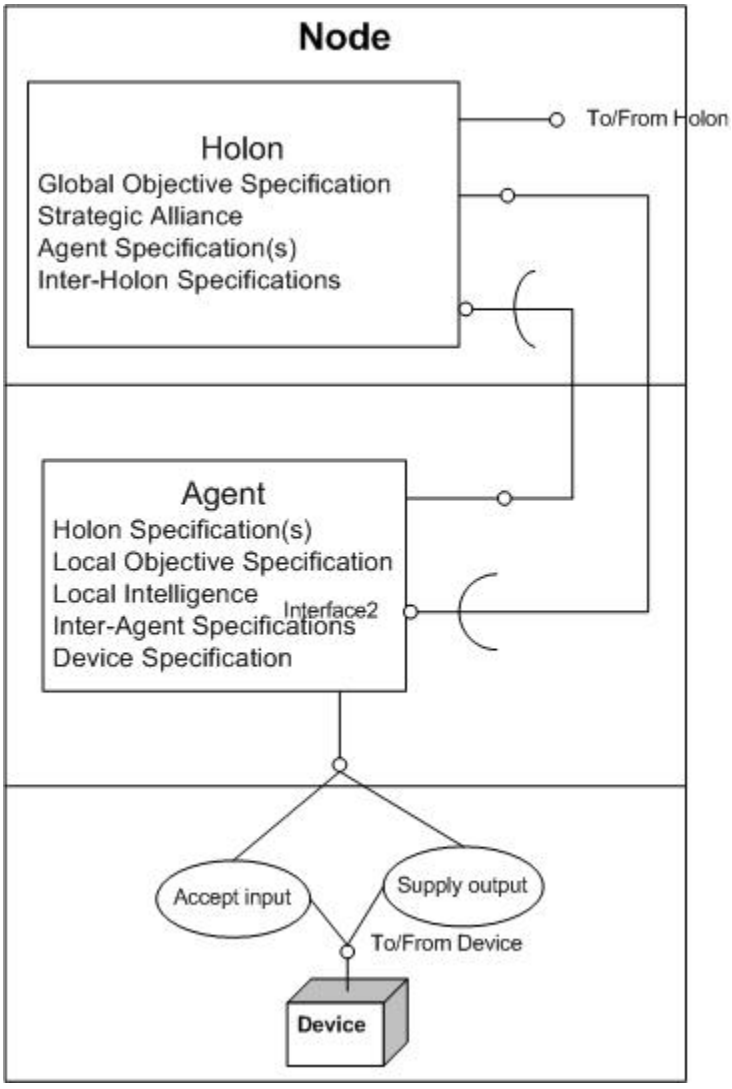

Figure 2.4: Holon-Agent architecture.

In order for the holon to perform its role within the distributed system architecture, it should primarily possess a reasonably comprehensive specification of the system objective. It will require additional intelligence on how to operate with holons on other nodes in order to achieve the system objective together. In its interaction with a single agent residing on the node, the holon must possess knowledge regarding: the identity of the agent; capabilities of the agent as specified via its Application Programming Interface (API); its priority or privilege levels, if any; when and how it should be notified to take action. In its interactions with other holons in on a system-wide level, it will need to possess similar knowledge on their specifications as well. In the situation where a holon may have jurisdiction over multiple agents residing on a single node, it will require the

necessary intelligence in comprehending, routing and managing communications between agents in intra-nodal and inter-nodal contexts such that the system objective is satisfactorily achieved. This is contingent upon the holon being able to keep a regularly updated record of the agent(s) under its jurisdiction and the holon will need to be endowed with that capability.

Similarly, an agent within the distributed architecture under the jurisdiction of a holon will require comprehensive specification necessary for control of the device that it is operating upon. This specification may include, among other things: the identity of the device; control commands that it responds to; response and output messages that it may send back; rules regarding action to be taken with respect to messages from the device. In addition to specification of the device itself, the agent will need specification of how to communicate with the holon via its own API. The agent will also require knowledge of the identities and specifications of other agents on the same node or different nodes, for communication purposes related to inter-device control.

As an example, consider the security system being developed. Figure 2.5 depicts the system architecture with the additional concepts that have been discussed thus far. In this case, there will be a holon and associated agent residing on the motion sensor, network camera, RFID tag reader and supervisor console. A node in this system will henceforth be referred to as a Holonic Unit (HU). Thus, the system architecture will be comprise of the following entities:

- Motion Sensor HU with Motion Sensor Holon and Agent
- Network Camera HU with Network Camera Holon and Agent
- RFID Tag Reader HU with RFID Tag Reader Holon and Agent
- Supervisor HU with Supervisor Holon and Agent

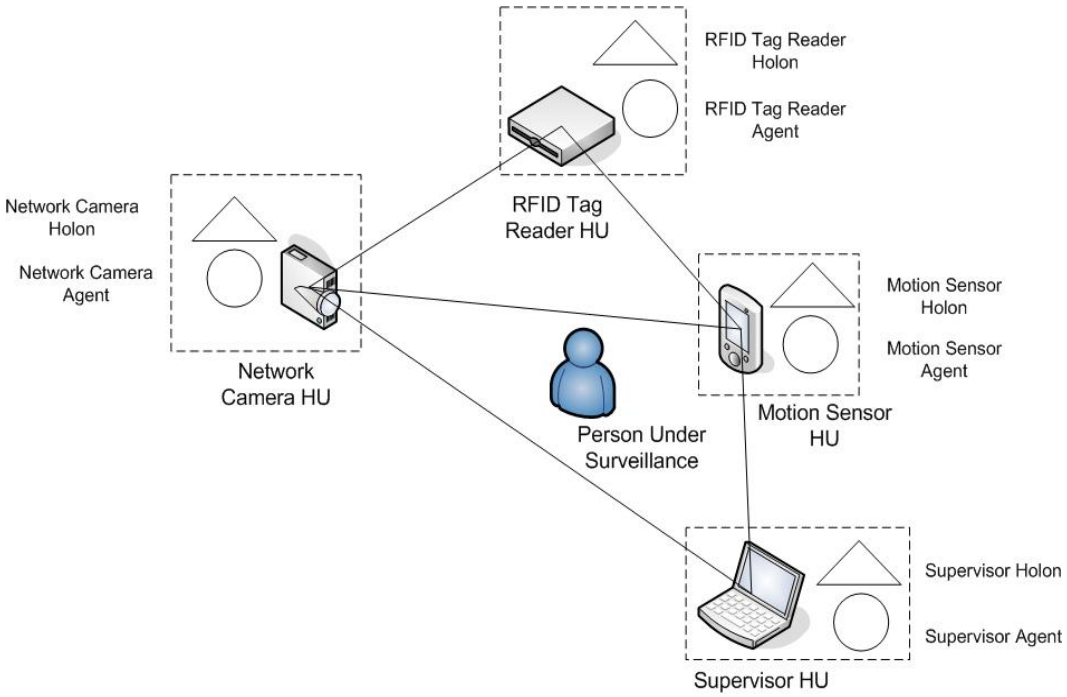

Figure 2.5: DIS security system architecture with holons and agents.

The holons residing on the various holonic units possess the specification of the overall 'strategic' system objective – in this case, to monitor the status of the secured environment and report pertinent information to the Supervisor HU. The parameter specifying 'pertinent information' may vary – registration of motion; snapshot of moving entity; identification of RFID tags moving into or out of the secured environment and so on. The Motion Sensor Holon possesses the knowledge that the Network Camera and RFID Tage Reader Holons must be notified in the event of motion being sensed and they must work together to inform the Supervisor Holon. As an example of possessing knowledge of their associated agents, the Network Camera Holon is aware that an agent identified as 'Network Camera Agent' resides under its jurisdiction, may be instructed to operate the camera via the commant "Take Snapshot" and has highest level priority in interaction when a snapshot has been taken by the camera. It also routes any incoming communication to the Network Camera Agent from holons on other HUs.

Similarly, the RFID Tag Reader Agent possesses the knowledge that: it controls a device identified as 'RFID Tag Reader'; the device responds to the command 'Commence Reading Tags'; it responds with messages listing the identification codes of RFID Tags within sensor range; when a message is received, the rules of operation specify that the RFID Tag Reader Holon must be notified with a message meant for the

Supervisor Agent. Finally, it knows that agents identified as 'Motion Sensor Agent', 'Network Camera Agent' and 'Supervisor Agent' exist within the system and how to process messages received from or meant for them.

The intelligence differentiating this architecture from a security system implemented via a distributed system architecture is exhibited as holonic units operate in an intelligent manner, modifying necessary operating parameters related to operating constraints (bandwidth, storage space, power consumption, etc.) and behaviour in consultation with each other such that the system objective is satisfactorily met.

In addition, a serverless environment such as this necessitates an alternative solution to a central location of control logic and data, if it is to be functional. Architecture considerations with regard to data manipulation and decision-making will be impacted. Alternatives are needed to processes that bring data back to one particular entity or node for processing and/or decision-making. Both code and control logic are distributed over the HUs that form the system. Each HU must be endowed with the capability to take responsibility for making the best processing decisions that it deems relevant, along with its global system responsibility [82]. The system in its entirety will be realized through its constituent nodes operating together in terms of program code and control logic.

In this respect, most currently available distributed sytems platforms [83, 84] making use of agents in their architecture, some examples of which are JADE [85], Cougaar [86, 87] and JXTA [88], are not feasible without extensive modification for demonstrating the proposed DIS system. For example, although JADE possesses many useful features, it introduces an element of centralisation with reliance on the existence of a 'Main Container' [89] with which all agents must first register in order to function, and without which proper operation is not possible. Similarly, although JXTA architecture [90] refers to 'peers', it may also employ elements of centralisation through the subdivision of peers into categories such as 'superpeers', 'rendezvous peers' and 'relay peers' in terms of the functions that they perform. In addition, a splitting of responsibilities between strategic and logistic objectives among entities is not inherent in these architectures.

It is for these reasons that the Intelligent/Distributed Enterprise Automation (iDEA) Laboratory [91] developed its own distributed agent environment, known as, the Holonic Technology Platform (HTP) [75] which provides the necessary requirements to

develop and implement a DIS in the security system application. Table 2.1 outlines some of the general differences between the inherent properties expected of a DIS architecture (which are adequately met by HTP) and other distributed architectures using agents, that were evaluated.

Having discussed how a DIS would work, the hitherto consideration raises the important aspect of system requirements that must be met by such a system. These practical constraints are required to be satisfied in a manner that does not adversely affect performance, both when realized for its initial requirements as well as for future changes that may occur. Some of these we outline as follows.

Table 2.1: Comparison of DIS and multi-agent architectures.

Evaluation of System Architectures	
Distributed Intelligence System	**Generic Multi-agent System**
Separation and independent addressing of global (strategic) and local (logistic) objectives	No clear separation of objectives; greater possibility of conflict in agent operation
Communications and other infrstructure does not determine design of environment	Architecture may be constrained by infrastructure dependency
No centralisation of elements such as communications, knowledge, data or control	Elements of centralisation may be present, requiring architectural modification, particularly for conflict resolution
System is realized as a whole when all constituents are organized and operate together	No necessity or inherent perspective of consituent nodes operating as a whole

2.3.1 Flexibility

Flexibility in the context of systems architecture refers to ease of modification or adaptation of the system to changes in its environment [92]. In a Client/Server environment, adaptation of the system requires repeated monitoring and configuration from a central point. The requirement for a DIS in terms of flexibility is the capability for

modification from anywhere in the system. Modification may occur in the context of system topology, load distribution, resources, etc. The DIS should be able to intelligently adapt to new scenarios whether upon specification from the user or a consensus agreed upon by all the entities in the system due to change in the system environment. System maintenance and modification should become easier, if nodes have to be taken offline or shut down, without having to impact system performance significantly.

In this regard, an advantageous feature that will differentiate a DIS from Client/Server or various multiagent systems hitherto discussed, is the ability to transfer concepts of software coupling from multiagent systems to the hardware domain. Weiss [93] initially discusses the idea of strong and loose coupling in the architecture of an intelligent agent [94], later extending it to interactions between agents [95]. The rationale behind this methodology was to propose an alternative to strong dependencies among software modules for proper operation, thus providing a solution to problems created by such technologies as the Remote Procedure Call (RPC) [96]. A DIS aims extend this principle of loose coupling to hardware. For example, by implementing some form of holon and agent into hardware, it can be endowed with some measure of intelligence. This enables the breaking of strong dependency between various hardware modules, and appropriating the advantages realised in software systems structured in such a manner, such as the capability to intelligently adapt to changes in the system environment.

2.3.2 Scalability

Scalability requirements concern the ability of the DIS to grow in terms of scale. This also refers to the impact, if any, on performance of the system in terms of locating and identifying entities, bandwidth, latency of system response and similar parameters. A DIS is required to handle these demands in an intelligent manner as the system grows. Processing and information handling should be spread throughout the system. In addition, a DIS should be able to incrementally grow intelligently. Unlike the case of a centralised framework, upward scaling of a DIS should not require extra controllers and associated hardware; the devices themselves, with onboard holons and agents providing intelligence for control, should provide this capability themselves.

Any system growth should contribute processing power and resources, instead of becoming burdens on a finite centralised architecture. In addition, changes in complexity

are required to be handled in a manner that does not adversely affect system performance. Complexity issues will arise in system operations that require different strategies to address them as opposed to those implemented for client/server architectures. Examples of such operations might be: location of entities; routing of communication; knowledge of the system and management of resources among others. Data-related issues such as storage and distributed processing become more complex and are required to be handled appropriately. As an example, in the security system application, complexity arises in processing data generated by various sensors, especially if the data generated by one device is required by another. One possible strategy will require making intelligent decisions with regard to distributed processing of information as it is generated at its source, reducing the complexity of moving data to other locations or nodes to be processed there. No doubt, the selection of a DIS architecture will extract a cost in terms of system complexity in contrast to a centralised system, where complexity is concentrated at the server entity. However, a good DIS architecture can mitigate this effect or even make it an asset.

2.3.3 Security

It is required that the DIS be secure in its operation. If one or more nodes are compromised by a foreign entity that is hostile in its intent, data may be corrupted and software modified to cause behaviour that was never intended to occur. The DIS should possess intelligence within its architecture to circumvent nodes that have been compromised. Nodes may continually monitor each other, work together to identify and contain potentially compromised nodes and manage security of data and control logic such that a hostile entity that manages to breach one or more nodes gains only partial understanding of the system, insufficient to cause a complete system breach. An intruding entity would have to compromise every single node in the system, instead of compromising a central server.

2.3.4 Robustness

Robustness or fault tolerance in a systems engineering context relates to the ability of the system to withstand failure in one or more of its components. This covers the aspects of hardware, software control and communications. With respect to hardware, each device ideally possesses all the hardware it requires for independent

operation; there is a higher risk of system failure if all devices depend on a common piece of hardware for normal functioning and whose malfunction can cause reliability issues. In terms of software, the risk of software failure is distributed over all the constituent holonic units, damage is mitigated and recovery can also be intelligently managed. If one or more units are rendered inoperable, the remaining can work together together to re-assign and perform the tasks assigned to the failed units and restore system capability to the extent possible. With a system employing centralised architecture, failure of a server entity can run the risk of failure of the entire system, even if redundant servers are incorporated into the architecture. Communications capability within the system should be distributed among the constituent devices; each device should be provided with the capability to independently communicate with other devices without a central point through which all communications must be routed.

In the case of the security system, the three essential devices that form the system do not depend upon a central holonic unit for instruction on how to operate or course of action to take.

2.3.5 Cost

Cost is related to other factors mentioned here. A DIS system is expected to provide an advantage over client/server in terms of incremental cost of deployment. An entire supporting infrastructure (communications, decision making, storage, services, processing capability) does not need to pre-exist for the system to be able to work. Each incremental change or alteration (or in the case of the security system, Holonic Unit), contributes to these aspects of system resources. Thus, the system is required to demonstrate the capability for incremental development and deployment, reducing the associated costs.

2.3.6 Administration

System administration capabilities for a DIS requires a different approach than usual practices employed in client/server scenarios. Due to the nature of the distributed architecture, it is unfeasible to manage the system from a central node. A requirement placed on the DIS in this regard is that information regarding current system status and operating conditions be accessible from anywhere in the system. In addition, it will also be required that an administrator be able to remotely access any node in the system, for

the purpose of configuration, maintenance and upgrading. Thus, administration capability and functionality is distributed over all the entities that comprise the system architecture.

Consider Figure 2.6, which illustrates how a hypothetical security system, larger in scale than the one under development, might exhibit these requirements. In this scenario, the security system is comprised of multiple HUs operating on a security policy similar to that outlined in the four-node system outlined in Figure 2.2. The system has grown in scale with respect to Figure 2.2, with the addition of HUs. Each node with its associated holon and agent makes resources available to the system, resulting in contribution to distributed processing, communications, knowledge, storage, and control. Also, incremental cost of development and deployment allows additional HUs to be added to the system and configured with relevant tasks without requiring major modifications to other HUs in the system. Although failure of a HU (such as one of the network cameras) will result in the loss of contribution to the system objective and resources provided by that HU, the system can intelligently work together to negotiate contribution and usage of resources so that performance is maintained within satisfactory limits. In addition, although a malicious node is potentially compromising one of the Desktop PC HUs, the intruder will only be able to acquire the piece of data stored on that HU, which is insufficient to construct a complete picture of knowledge or data related to the whole system. In order to compromise it, the attacker would have to compromise every HU present in the system

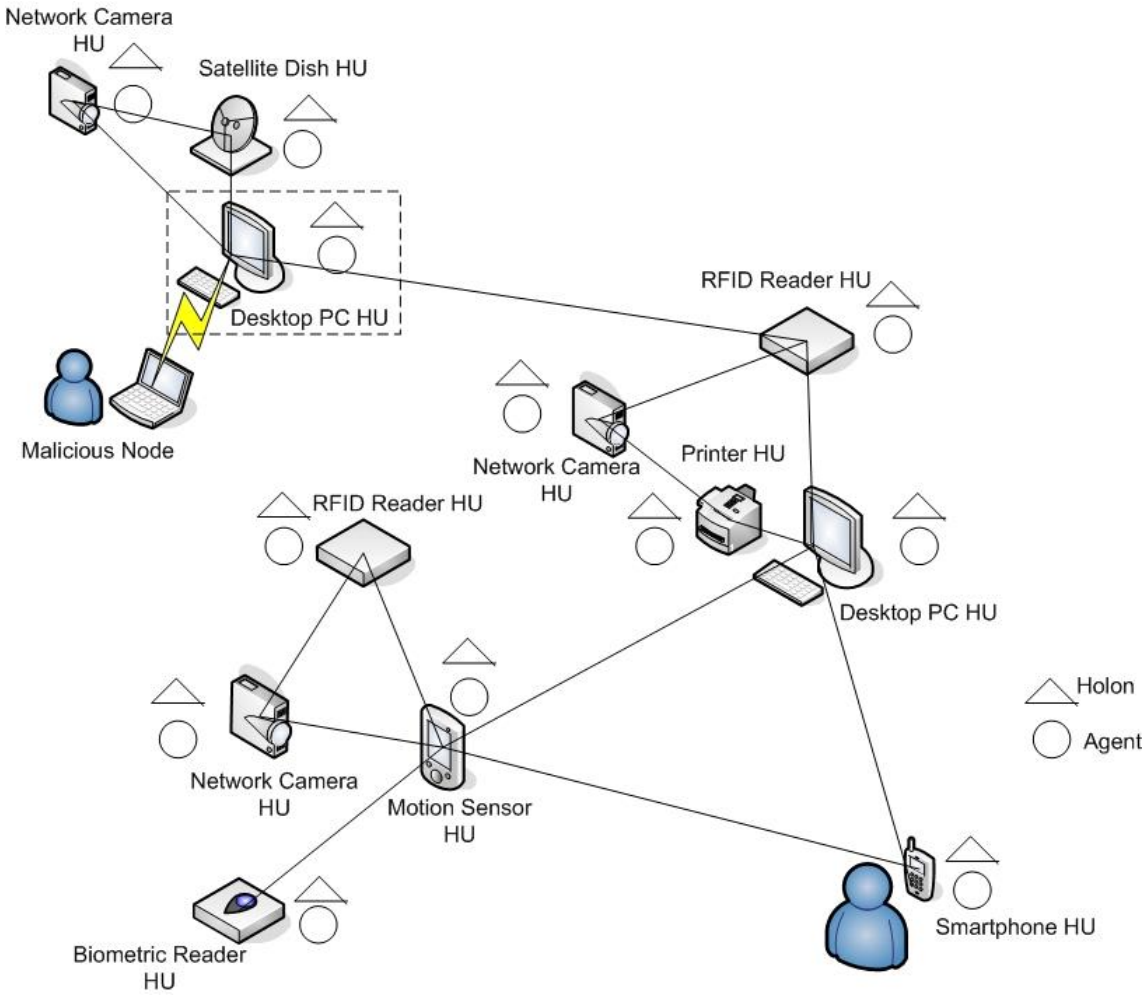

Figure 2.6: Requirements of a DIS architecture.

A supervisor using the Smartphone HU is able to interact with and configure any HU in the system, performing system administration and management. He might access any HU remotely and only concern himself with making local changes to HUs, without being burdened with the responsibility of the impact and modifications needed on a system-wide level due to local changes.

Table 2.2 provides a comparison between different system architectures on a five-point scale of comparison (Very Low, Low, Medium, High and Very High) with respect to different general parameters on which each could be rated.

Table 2.2: Comparison of different system architectures.

	System Architecture				
Property	**Client/Server**	**Decentralised**	**Distributed**	**DIS**	**Holonic**
Adaptability	V.L	L	M	H	V.H
Scalability	V.L	L	M	H	V.H
Robustness	V.L	L	M	H	V.H
Security	V.L	L	M	H	V.H
Cost	V.H	H	M	L	V.L
Management	V.H	H	M	L	V.L

2.4 Design Specifications and Considerations

In a similar fashion to the notion of architecture, the connotation of 'design' involves specification of the kind of components, the nature of their function within the system and the capabilities that will be provided in order for components (and the system as a whole) to function properly.

The development of a DIS will require the peers in the system to communicate with each others using an event-triggered process [96]. This may be accomplished by notifications that peers send themselves or each other upon an event that has transpired. Holons and agents residing on peers identify and register those on other peers in their neighbourhood, and register with them in turn. This 'handshaking' process facilitates peer discovery, listing of services provided by each peer, protocols understood and other vital information required for peers to establish communication. Figure 2.7 shows a UML sequence diagram illustrating a simple event-triggered process between two generic agents on peers within a DIS. In such fashion, other interactions within and

between peers are triggered based upon notification of events that have occurred among holons and agents. Interaction or behaviour of holons and agents on a peer may be specified or framed by a 'rule base' in the form of an 'event list' or table to be referenced upon notification of an event.

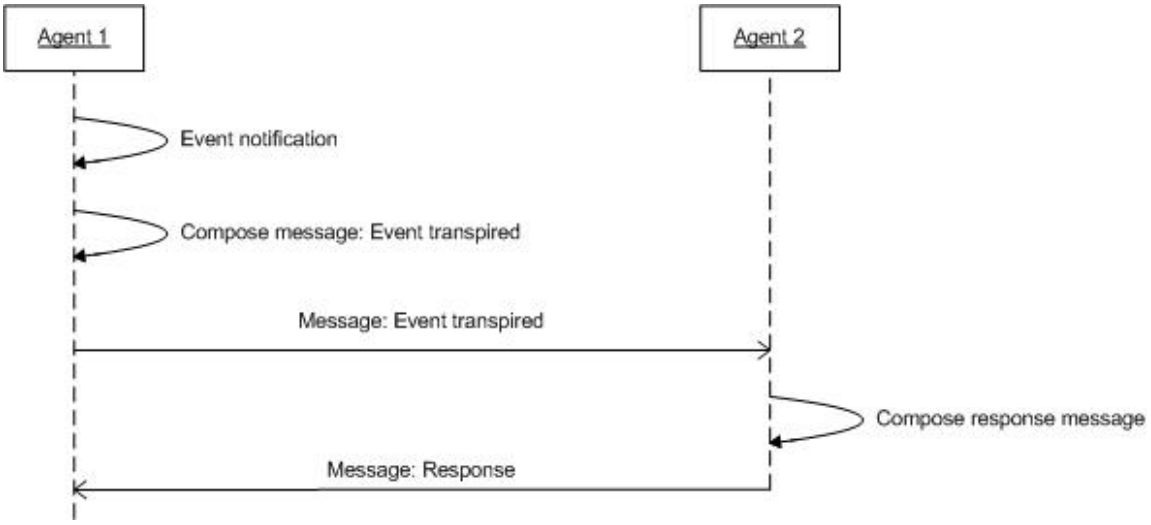

Figure 2.7: UML sequence diagram of generic event-triggered process.

A corresponding action is undertaken or decision made once a matching event description is found. In order to function effectively within their neigbourhood, peers must have a means of acquiring information about their environment, such as the topology of the architecture in which they find themselves. As an example, consider an abstract topology of a distributed architecture in Figure 2.8. In this scenario, the holon on node ‘A’ must be able to acquire knowledge of the ‘distance’ of another holon in its neighbourhood such as the holon on node ‘B’, or of one ‘further’ away, such as the one on node ‘F’. Distance specified in terms of the number of nodes that must be traversed to deliver a message to an intended recipient provides peers in the system with a means to construct a map of the known system topology. This and other relevant information is specified in messages exchanged in the handshaking process.

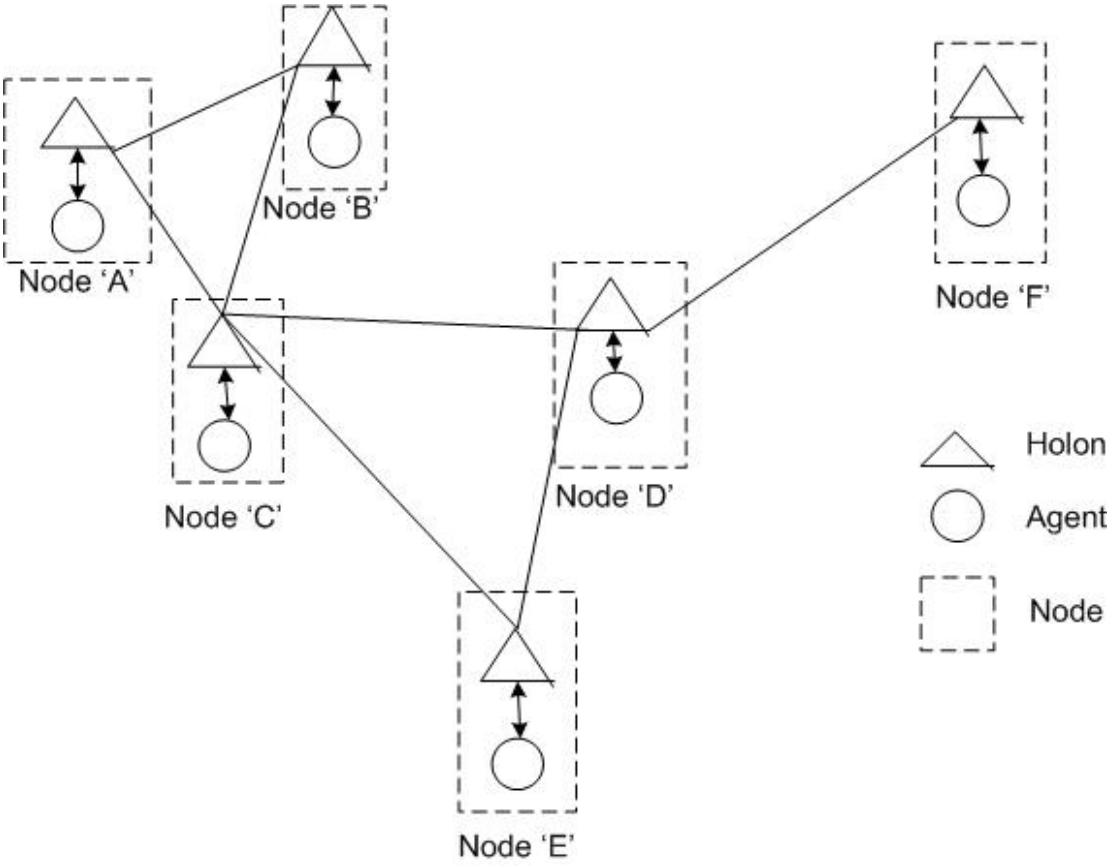

Figure 2.8: Hypothetical topology of peers within a distributed architecture.

The mechanism of interaction regarding local and global objectives of peers will be achieved using an agent-based architecture similar to those previously discussed. The agent paradigm is suitable for construction of a DIS, one of the reasons being its methodology of architecting multiple interacting agents in a non-monolithic framework, which can be easily modified, extended or re-built for different applications. Such architectures are typically built with the aim to endow agents with capacities such as: limited autonomy and intelligence; localised views of the system; negotiation of strategies for problem sovling and recovery from failure, among others. These useful features can be employed in enabling a DIS to meet its architecture requirements which were discussed in Section 2.4. One aspect of agent-based architecture requiring modification for the system under development arises when agents must deal with global as well as local objectives. An example of a global objective might be to maintain a system-wide resource (bandwidth, storage space, processor or memory usage) within a certain limit or use it to the maximum possible. Some architectures would require agents

on peers to deal with local and global objectives simultaneously. The consequence of such a design strategy is that agents may end up doing too much work in trying to meet both objectives, which may be difficult to meet if the objectives happen to conflict. Such a situation would essentially be a smaller scale of the problem posed by the client-server paradigm.

Although an agent-based system architecture is advantageous in many respects, such modifications are necessary to incorporate the division of responsibilities in the DIS between holons working to meet system-wide objectives and agents operating in a limited jurisdiction with local responsibilities. Although some agent-based paradigms characterise agents as operating to meet their own interests [97, 98], this will also require modification for use in a DIS. An agent operating on a peer operates with the intent of meeting a user-specified local objective by performing an assigned task. Multiple agents residing on each peer may operate under the jurisdiction of a holon. An agent will therefore have to work together with other agents residing on the peer in the interest of meeting the local objective, where attainment of its own individual goal or function is not completely attainable. For example, a HU in the four-peer security system will have different agents residing on it, assigned with accomplishing different tasks, examples of which may be: bandwidth usage, storage space and processing capability; message handling; encryption; statistics collection; routing; provision of services to other peers, and so on. This is illustrated as an example in Figure 2.9. The holons on each peer are provided with the knowledge of the global view of the system objective (monitoring the secured environment and reporting pertinent information to the supervisor). The depicted agents operating on each peer work intelligently with each other in handling their individual tasks to achieve their local objective, which is proper operation and control of the device and its resources. They may report information back to their respective holon, which will in turn make decisions about the importance and relevance of information that it receives.

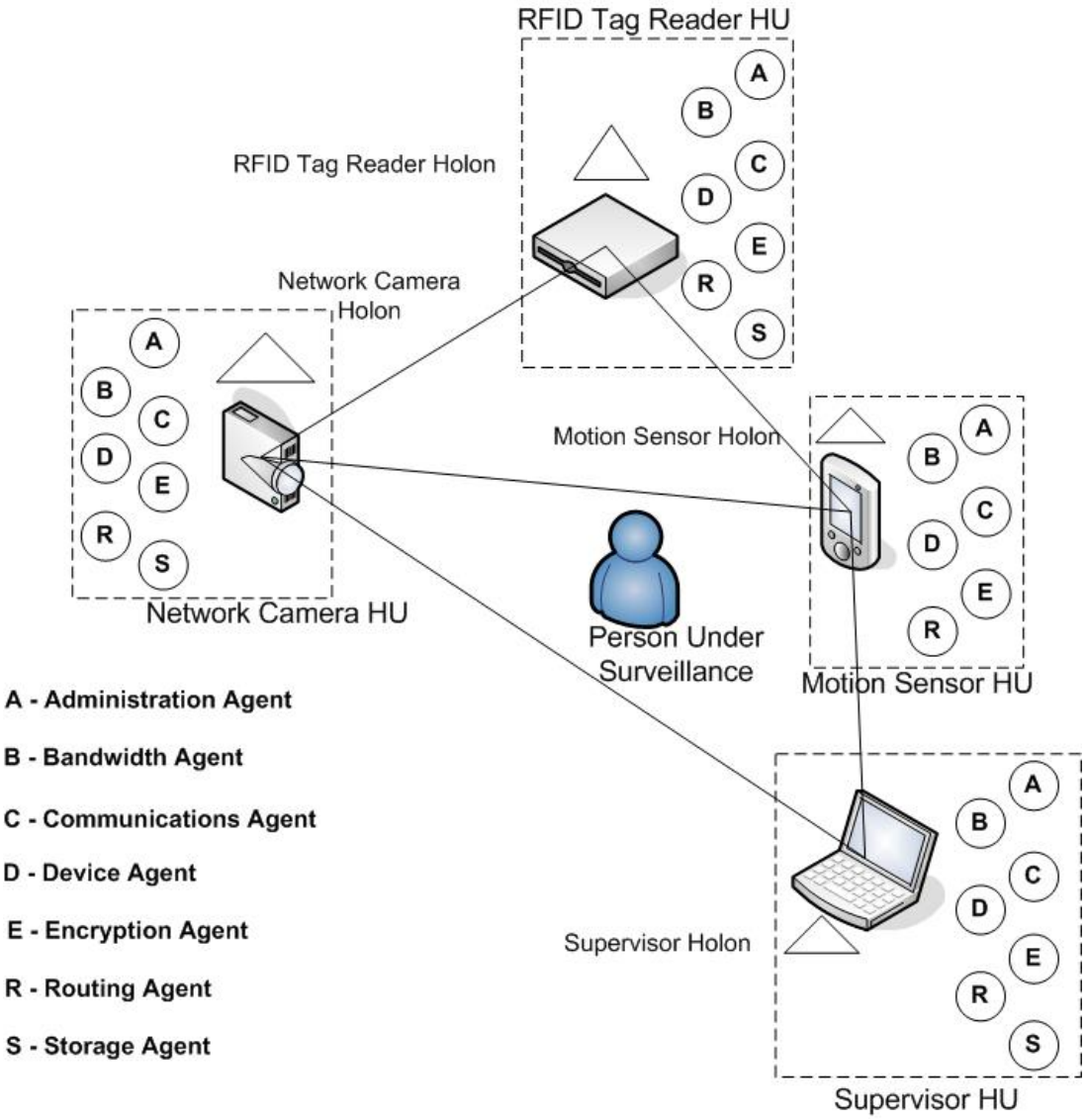

Figure 2.9: Agents operating within a DIS security system.

This splitting of responsibilities in this model, which will henceforth be referred to as the Holon-Agent model, is summarized in the table below.

Table 2.3: Division of responsibilities between holons and agents.

Responsibilities of the Holon	Responsibilities of the Agent
System-wide goals (e.g. Maintain available storage space)	Local objectives (e.g. Device power consumption)
Collaboration, Co-ordination, Co-operation	Facilitate services (e.g. Startup device, Shutdown, etc.)
	Logistics (e.g. Message routing: intra-holon and inter-holon)

With respect to the Holon-Agent Model described thus far and Table 2.3, collaboration [98] is the co-labouring of holons to accomplish the global objective(s) of the system; co-ordination [98] is the interaction between holons in order to properly sequence actions and events; co-operation [105] relates to groups of holons interacting in a neighbourhood within the system. Section 2.4 considered details of the possible knowledge required by holons and agents in their interactions with each other and carrying out their operations. With reference to Figure 2.9, the Motion Sensor Holon, Network Camera Holon, RFID Tag Reader Holon and Supervisor Holon will collaborate, co-ordinate and co-operate between each other to achieve the global objective of the security system. The Device Agent on each HU tasked with local management and control of a device will process and execute control messages received from other Device Agents in the system, as well as issuing control messages to other devices via its respective Holon. Figure 2.10 abstracts the interaction between a Device Agent and its associated generic device, along with a representation of its internal architecture.

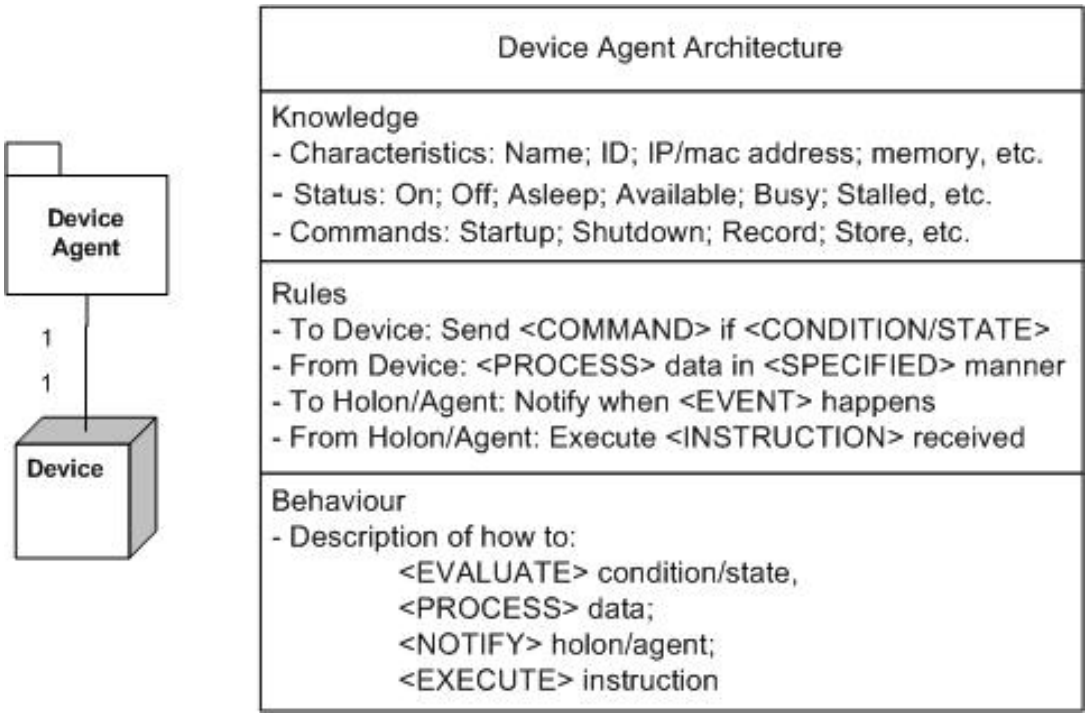

Figure 2.10: Abstract represent of Device Agent and generic device.

An agent operating under a Holon is essentially a package of software code and associated resources that may perform a particular task relevant to local objectives on the peer. Pertinent details relevant to the design and construction of the Device Agent emerge from an examination of the requirements needed to control a generic device. For the purposes of the DIS under development, the internal architecture of the Device Agent is comprised of three main sections – Knowledge, Rules and Behaviour (KRB) [94, 98], as described below.

2.4.1 Knowledge

The concept of 'knowledge' possessed by the Device Agent relates to data that the device generates regarding its own state or the state of its environment. This may occur through an internal signal notifying change in state or an external signal representing a change of state in the external environment or system of which the device is a part. Knowledge acquisition occurs via messages that a Device Agents receives from a device, sends to itself or to other agents in the DIS. For example, the motion sensor may register motion within its sensor range. Transcription occurs from the output voltage from the motion sensor into a message sent to the Device Agent

controlling the motion sensor. When the Device Agent receives this message, and processes it, it will have acquired the knowledge that there has been an internal change of state (motion sensed) within the device (motion sensor). In a similar fashion, as messages pass from one Device Agent to another requesting a service or advising taking of action, knowledge acquisition occurs of changes occurring at other points in the external system.

2.4.2 Rules

The concept of 'rules' deals with the protocol for behaviour that a Device Agent will follow once knowledge of internal or external states is acquired and processed. The rules proposed for the DIS formulate a conditional statement resembling an 'if-then' statement. That is, given a finite set of states (either internal or external), if a change in state occurs which will give rise to knowledge or awareness of the same by a Device Agent, rules specify that in each case, a certain protocol is to be followed in terms of the course of action to be taken. For example, in the DIS security system, an abstract rule regarding the internal change of state of the motion sensor might be:

"IF motion has been sensed, THEN proceed to notify the RFID holonic unit to begin monitoring of RFID tags within its sensor range."

2.4.3 Behaviour

'Behaviour' with respect to device control is carrying out the course of action(s) specified by the rules that govern how to handle changes in internal/external states. As an example, when the Device Agent controlling the motion sensor receives an internal message specifying sensing of motion, acquisition of knowledge occurs. The rules governing this change in internal state may specify notification of the Device Agent controlling the RFID tag reader, to begin monitoring the movement of tags within its sensor range. The corresponding behaviour carried out by the Device Agent controlling the motion sensor will then be to:

- Compose a message notifying the Device Agent controlling the RFID tag reader
- Send it to the HU responsible for the RFID tag reader

Figure 2.11 illustrates the ideas discussed in the form of a UML activity diagram of a Device Agent operating under a Holon and controlling a generic device.

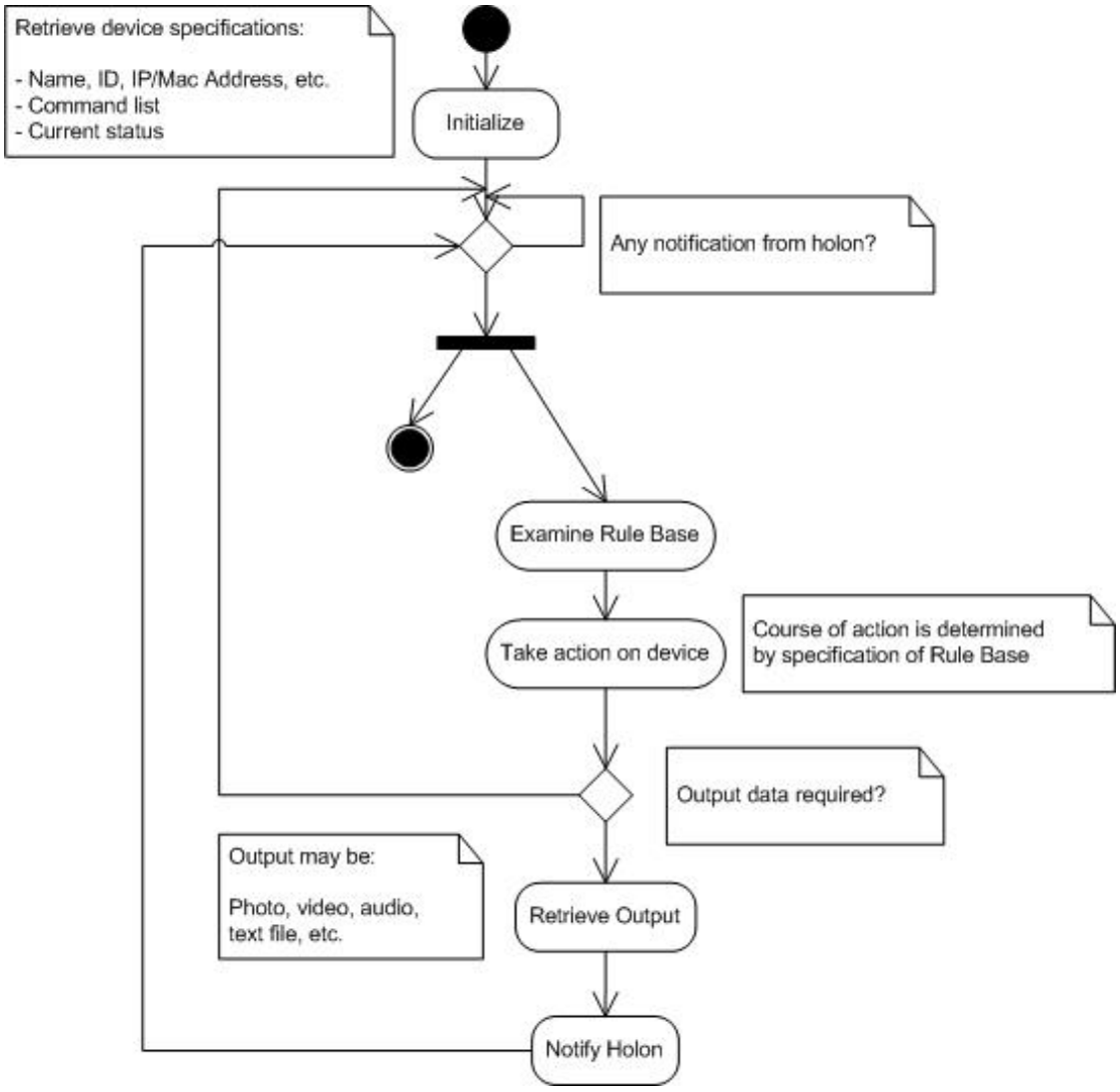

Figure 2.11: UML activity diagram of Device Agent.

Thus, the architecture of the Device Agent should mirror these three concepts in its implementation, composed of three sections dealing with each of these faculties - acquiring and processing of knowledge, specification of rules outlining actions taken after knowledge acquisition, and a section detailing the behaviour that implements those rules. The design of the Device Agent will ideally follow the template laid out in Figure 2.12.

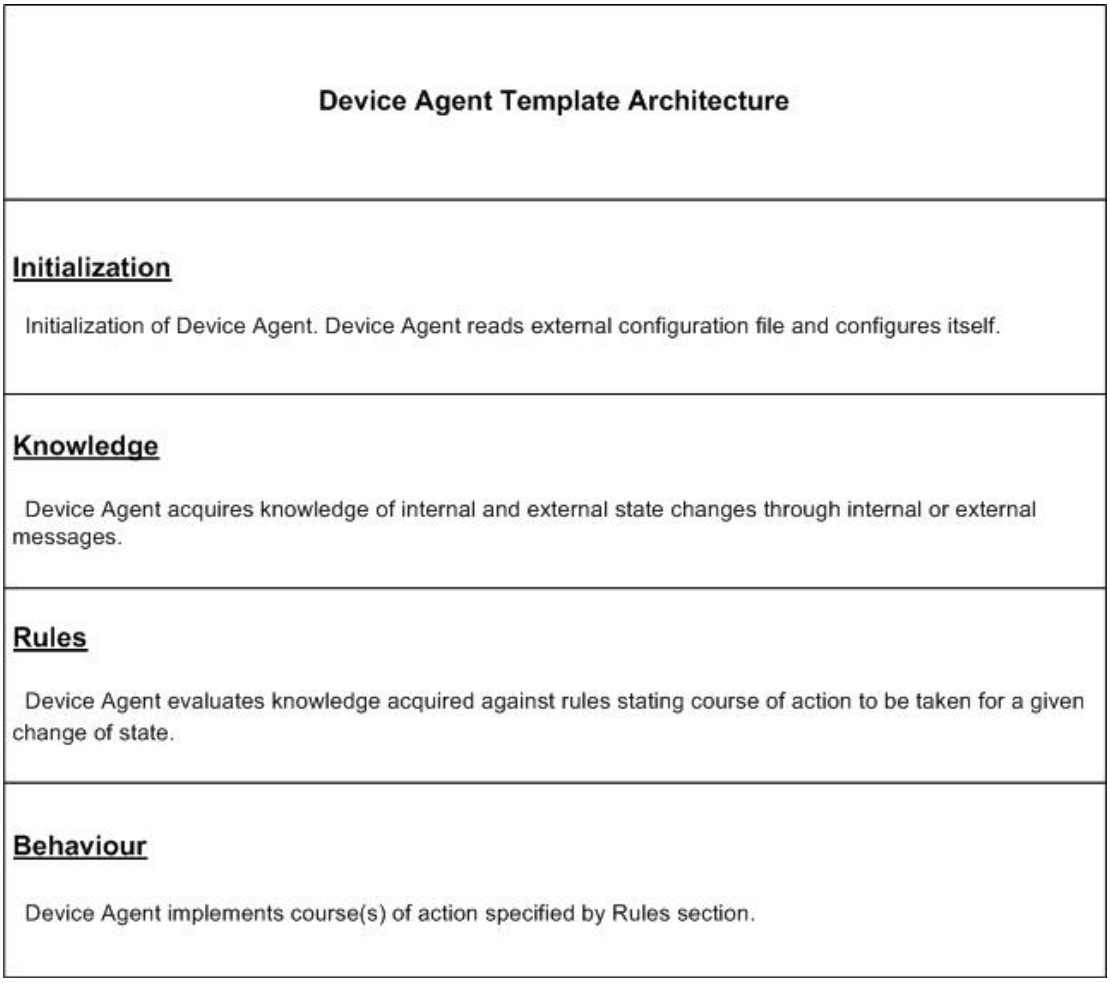

Figure 2.12: Template for Device Agent.

This abstraction of Agents implementing control methodology in terms of knowledge, rules and behaviour extends to the architecture of other agents providing internal and external services within the DIS. It will be possible to implement sophisticated methodologies of knowledge acquisition and interpretation of system state change, specifications of rules and actions taken as a result.

The interaction between Device Agent and device will take place through messages. Messages to the device are mapped from notifications that the agent may receive from the Holon, requesting execution of a particular control function. They may also be initiated by the agent itself, acting on local rules of behaviour regarding control of the assigned device. Using its rule base, the agent will possess a means to look up the corresponding message sent to the device for a given incoming notification. Having done

so, the agent will transmit the message over the hardware link to the device. As an example, consider a scenario in the security system depicted by Figure 2.13.

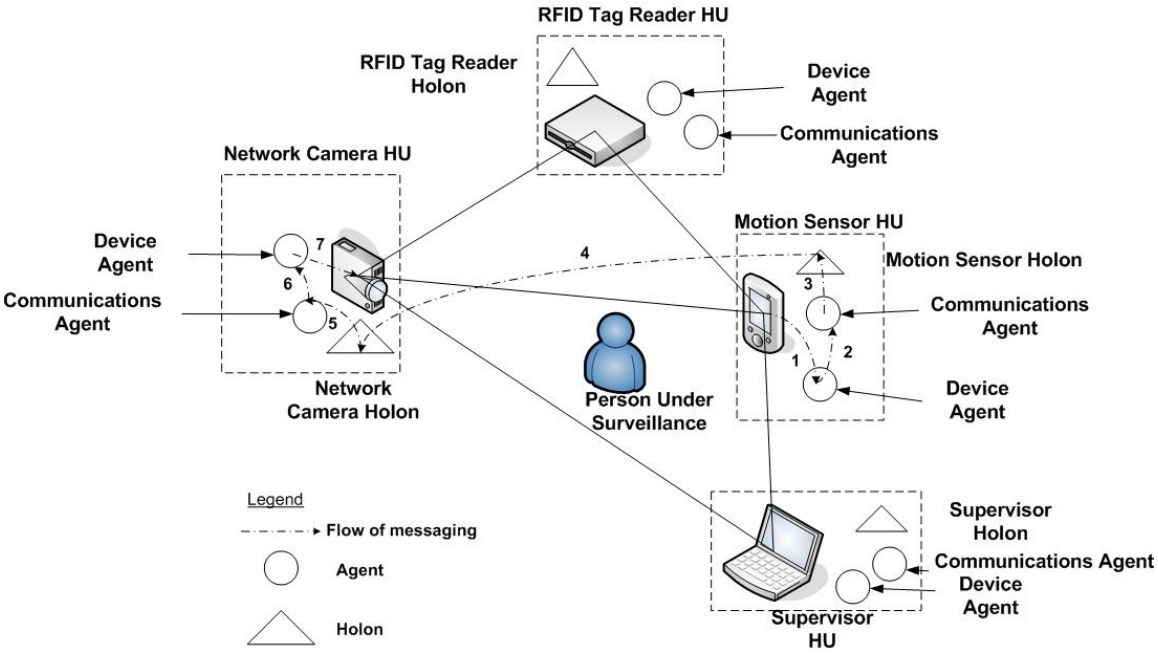

Figure 2.13: Interaction scenario within security system.

For the sake of clarity, other agents that are not relevant in the scenario under consideration have been removed from the figure. The Network Camera Holon will receive a message from the Motion Sensor HU for the Device Agent controlling the network camera stating “Wake Up Device”, and will route it to the Device Agent. Routing messages may involve a dedicated agent tasked with handling communications. After the Device Agent deciphers the message, it will map the instruction to the corresponding control message for the network camera, and then transmit it. When the network camera receives the message, it will exit its idle state and wait for further control messages.

The existing core architecture of HTP provides for much of the messaging capability required between agents [75]. However, HTP lacks the functionality required to handle the KRB capabilities planned for the Device Agent. Specifically, there is no mapping provision in HTP for a Device Agent to perform the necessary lookup, regarding the proper course of action corresponding to an event notification. This becomes a disadvantage when there may be a large number of event notifications that a Device Agent is capable of responding to, as well as numerous courses of action that it may choose from dependent upon the context of the situation. Attempting to provide this

capability using HTP's current provisions for messaging and response would require all of it to be concentrated within the Device Agent. This would be necessary, as the role, capabilities and requirements of the Device Agent are unique amongst the other Agents currently operating within the working HTP architecture [75]. Pursuing such a course of design will ultimately result in an unwieldy Device Agent architecture and program code, difficult to maintain and modify for future requirements. Thus, modification is necessary to the existing HTP architecture is in order, to provide this functionality.

The proposed mechanism of the interaction between agents residing on different holons as well as between holons themselves is illustrated in Figure 2.14.

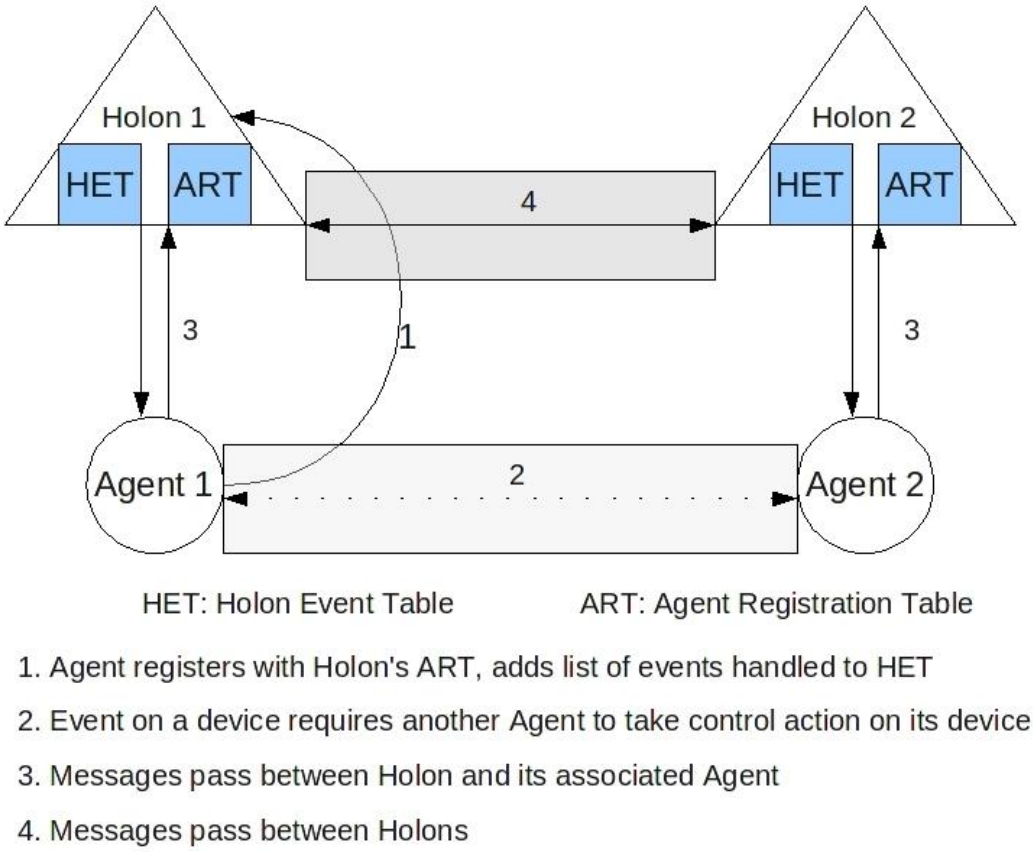

Figure 2.13: Holon-Holon and Agent-Agent interaction model.

All agents operating under a Holon's jurisdiction will register with its Agent Registration Table (ART), so that the Holon can keep track of them. This also enables proper routing of incoming messages for different agents to the correct recipient. Apart

from the ART, the Holon will also contain a Holon Event Table (HET), which will contain a list of the user-defined events and notifications to which a particular agent will respond to. Figure 2.15 illustrates possible architectures for the ART and HET.

Table 2.4: Agent Registration Table and Holon Event Table.

Agent Registration Table (ART)				
Agent Name	**Agent ID**	**Functional Description**	**Area of Responsibility**	**Services Provided**
Device Agent	H1.DA001	Operation of motion sensor	Device Control	Startup(); Shutdown(); ...
Encryption Agent	H1.EA001	Encryption/decryption of messages, files, ...	Security	Encrypt(); Decrypt(); ...

Holon Event Table (HET)				
Event Name	**Event ID**	**Description**	**Notify**	**Invoke**
MS	H1.DA001-MS	Motion Sensed	H2.DA002	Startup()
MS	H1.DA001-MS	Motion Sensed	H3.DA003	RecordTags()

The following example with reference to Figures 2.13 and 2.14 will illustrate the purpose of these components in the context of holon-holon, holon-agent and agent-agent interaction. Consider the scenario previously discussed in Figure 2.13 in the interaction between the Motion Sensor and Network Camera HU. Let Holon 1 and Holon 2 in Figure 2.14 represent the Motion Sensor Holon and Network Camera Holon

respectively. Similarly, let Agent 1 and Agent 2 represent the Device Agents on the Motion Sensor and Network Camera HUs. Let the rule base of Agent 1 on Holon 1 specify that it notify the Agent 2 on Holon 2 to start up its associated device upon a triggered event. Upon this event, Agent 1 will generate a message and pass it to Holon 1. Holon 1 will consult its HET, which specifies the correct course of action is to send the message to Agent 2 on Holon 2. Holon 1 will proceed to route the message to Holon 2, whose HET specifies Agent 2 as the proper recipient. Appropriate routing delivers the message to Agent 2, instructing it to start up its associated device. Agent 2 proceeds to carry out this action. In such manner, the Holon-Agent model facilitates interactions betweens holons and agents within the DIS.

2.5 Distributed Control

Referencing Table 2.2 and the earlier discussion related to the design details of the Holon-Agent model, an important design aspect requiring consideration is the ordering of entities within the architecture. An infrastructure is necessary that allows clear specification of how nodes in the system are organized and related to each other. In addition, this ordering mechanism must be such that it is suitable for use in a distributed architecture, accommodating all the requirements and implications of a DIS discussed thus far. Endowing this high-level functionality contributes to meeting such requirements of the DIS architecture as flexibility, scalability and administration, previously discussed in Section 2.3. For example, a means of specifying ordered structure allows ease of adaptation to changes within the system environment, as it might be possible for co-operating nodes to take intelligent decisions and make changes to structural specifications without resulting in major impact to system performance. It will be able to accommodate growth in the system without requiring substantial modification to the structural specifications already existing within the system as a whole. The mechanism of specification should have a positive impact on administration, allowing a supervisor to make changes such that the nodes in the system work together in an intelligent manner to implement the structural changes specified.

There are various ordering methodologies available to incorporate this design feature into the high-level functionality of the DIS architecture, some examples being linear ordering [99, 100], cyclic ordering [101] and partial ordering [102]. Of the various possibilities available, the partial ordering concept is most suitable in meeting the

requirements of this design feature. The lattice methodology of expressing partially ordered sets (i.e. posets), taken from Formal Concept Analysis [102 -104] is employed in designing this advanced functionality of the system. For a description of partial ordering, please consult the Appendices.

Building on the application of the security system considered thus far, the isomorphism of a lattice applies to a set of three nodes - the motion sensor, network camera and RFID tag reader HUs respectively. Consider Figure 2.16, which illustrates the isomorphism of the DIS security system as a Hasse diagram. All further explanation of organization within the system references this representation. Let ‘D’ be the entire set of devices in the DIS. system. The elements that form the set are represented in set notation as {MS, NC, RS}, representing the Motion Sensor HU, Network Camera HU and RFID Tag Reader HU, respectively. The fourth unconnected element in the diagram {SU} is representative of the system administrator, also known as the Supervisor.

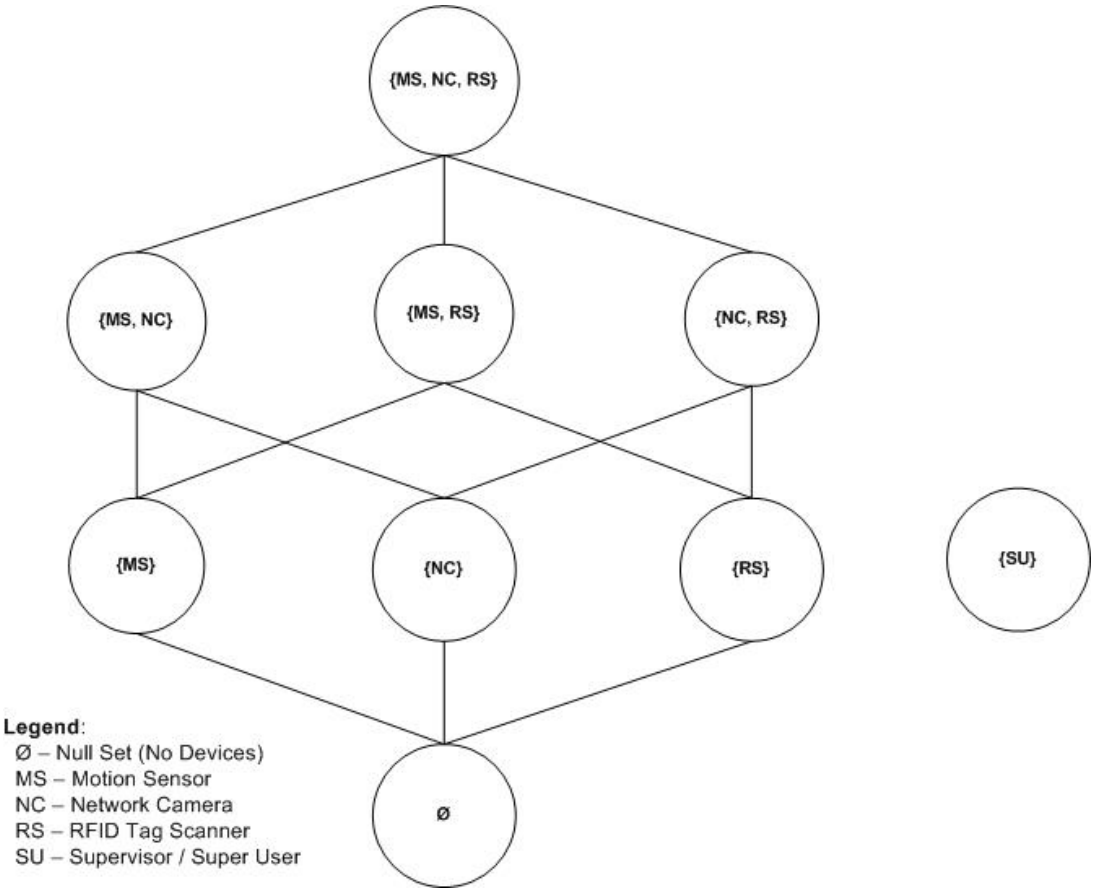

Figure 2.14: Hasse diagram for a DISC security system.

To illustrate how the lattice contributes to the ordering and control mechanism in the DIS, consider the events and changes that occur within the lattice framework when a person enters an area monitored by the security system. As {MS} senses motion, it may undertake one or more courses of action, depending upon the rules specified by the system administrator. In this case, the rules specify notification of {RS} to monitor RFID tag movement, then {NC} to take a snapshot of the monitored area. Figure 2.17 illustrates the partial ordering occurring amongst the elements of the lattice, when participating in this distributed control mechanism.

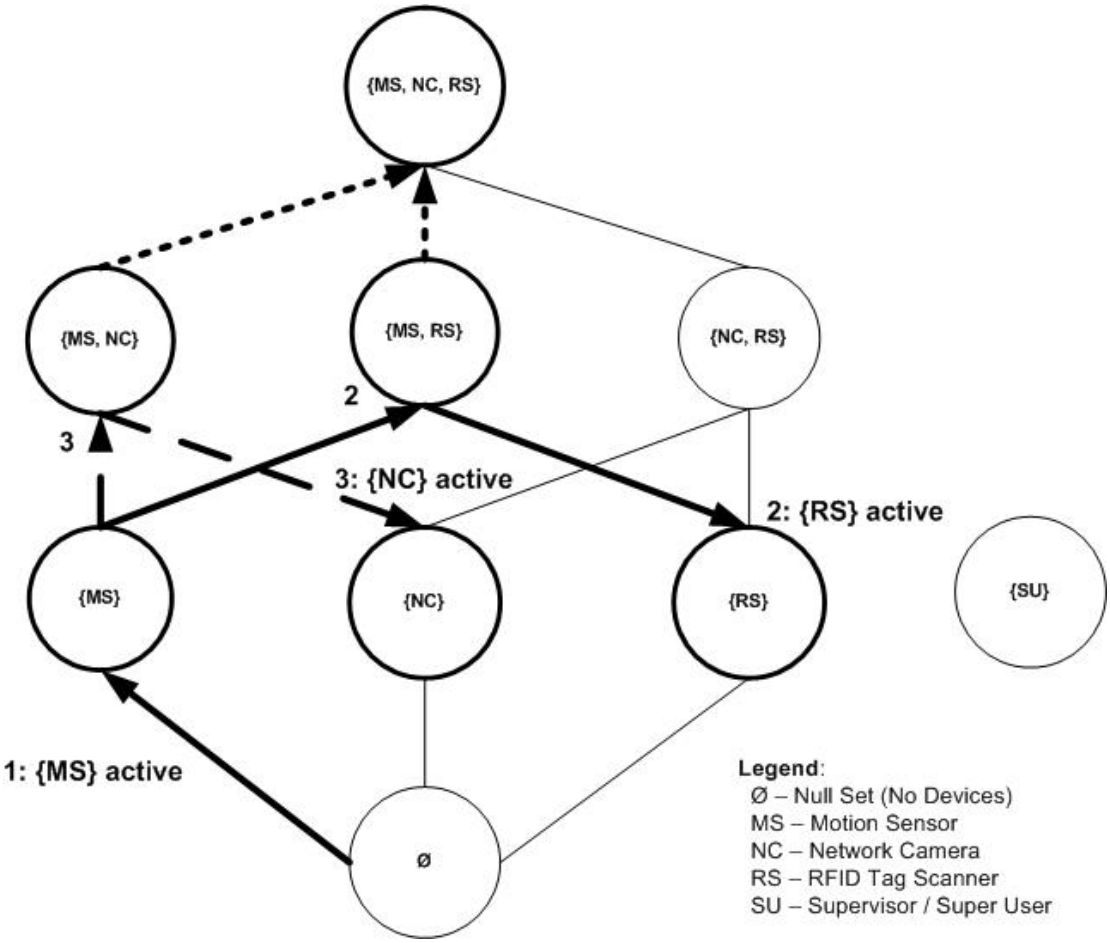

Figure 2.15: Changes in the lattice upon detecting movement.

Finally, the rules for all three HUs {MS, NC, RS} instruct them to individually notify {SU} with their acquired data of changes in the system environment. In this event, it is required to bring {SU} into the ordering structure of the lattice, forming a new partial ordering of four peers, shown in Figure 2.18.

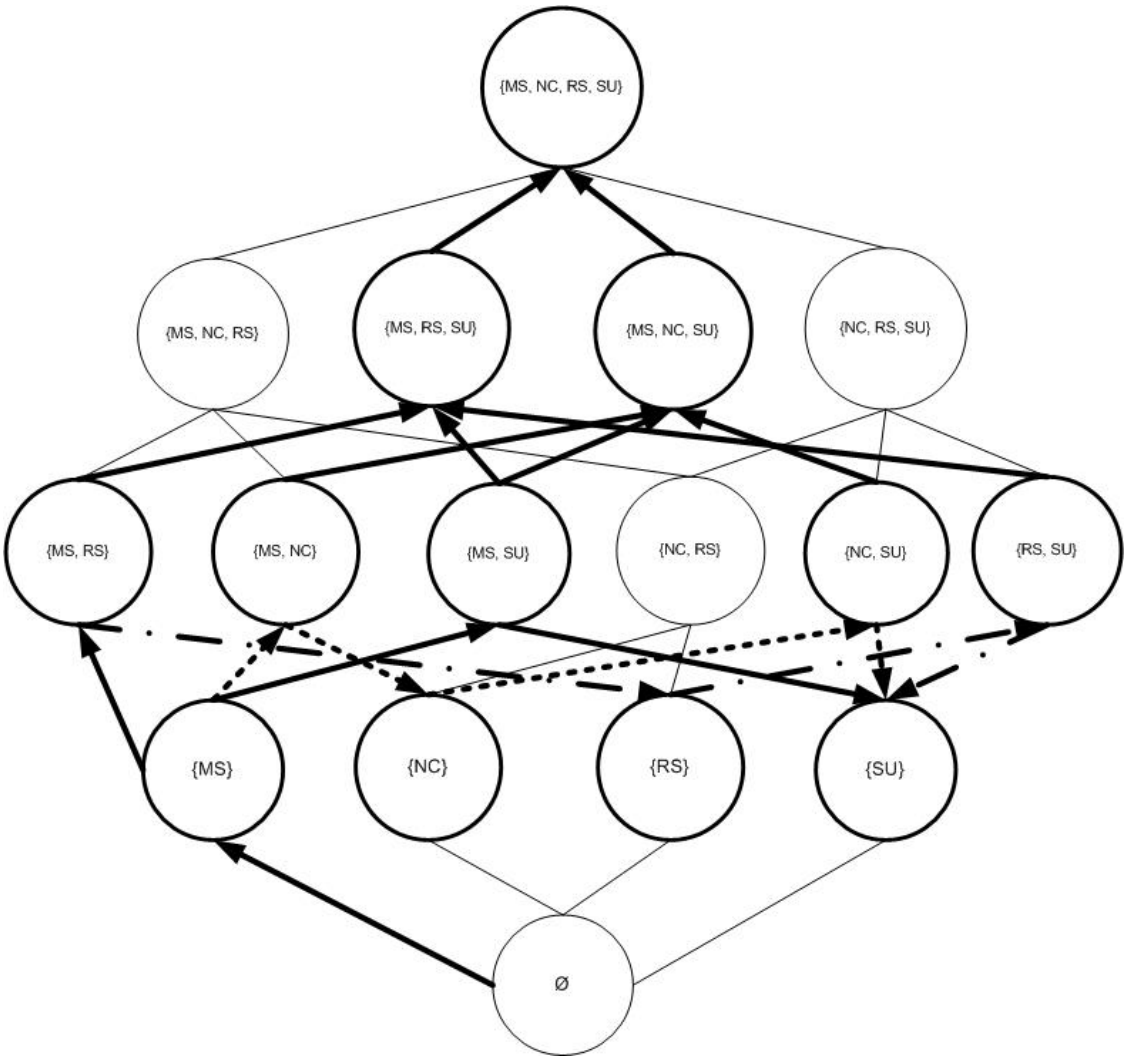

Figure 2.16: Partial ordering and distributed control with four-node lattice.

In this manner, it is seen how fully distributed peer control can be conducted in an intelligent fashion among the devices of a DIS, in order to achieve a user-specified global objective. In this application, {MS}, {NC}, {RS} and {SU} collaborate to achieve the monitoring of a secure environment and report changes occurring within it. They co-ordinate by taking decisions based on their user-specified rule bases, achieving a proper sequence of action within the ordering infrastructure. Co-operation occurs as devices can work together, modifying the sequence within the partial order and sacrificing their individual goals if necessary, to achieve the system objective.

The ordering that a lattice methodology and its associated tools bring to the simple architectures considered thus far favours its viability for use in a DIS. It

possesses a simple and elegant, yet extremely flexible and scalable means of dynamically creating and modifying the structure between nodes in the DIS. When all nodes within a DIS possess the capability to contain and comprehend an algebraic specification of the structural order, they will be able to maintain the structure within their neighbourhood in an intelligent manner. In addition, dynamic modification of the ordering specification during system runtime becomes much easier; nodes provided with new specifications in the form of a lattice equation can reconfigure the structure of their neighbourhood. All of this will occur in a completely distributed manner with no centralized control.

2.6 Conclusion of Architecture and Design Process

The motivation for an in-depth analysis and discussion in this chapter was to ensure the proper understanding and selection of architecture for a DIS system; the impact of architecture selection on system requirements; the resulting issues requiring solutions and the proper design of system entities with these factors in mind. The DIS security system application considered and developed in this chapter will allow the constituent nodes to implement a security policy without any centralised form of control. For example, one possible security policy might be as follows: upon detecting motion, the motion sensor is required to notify the network camera to take a snapshot of its field of view and inform the RFID tag reader to register any RFID-tagged objects moving within its sensor range. All three nodes will proceed to inform a terminal belonging to a human supervisor of the registration of motion, any photographs taken and RFID tags respectively. The supervisor may then proceed to take action as he or she sees fits. Figure 2.19 shows a sequence diagram of this security policy.

The considerations of this chapter regarding a DIS system in general and the security system application in particular explore the implications and issues that arise during the selection and design of a distributed architecture, with the end goal of demonstrating effective paradigms for development, implementation and deployment in real world applications. The next chapter will explore the implementation of the DIS security system in detail.

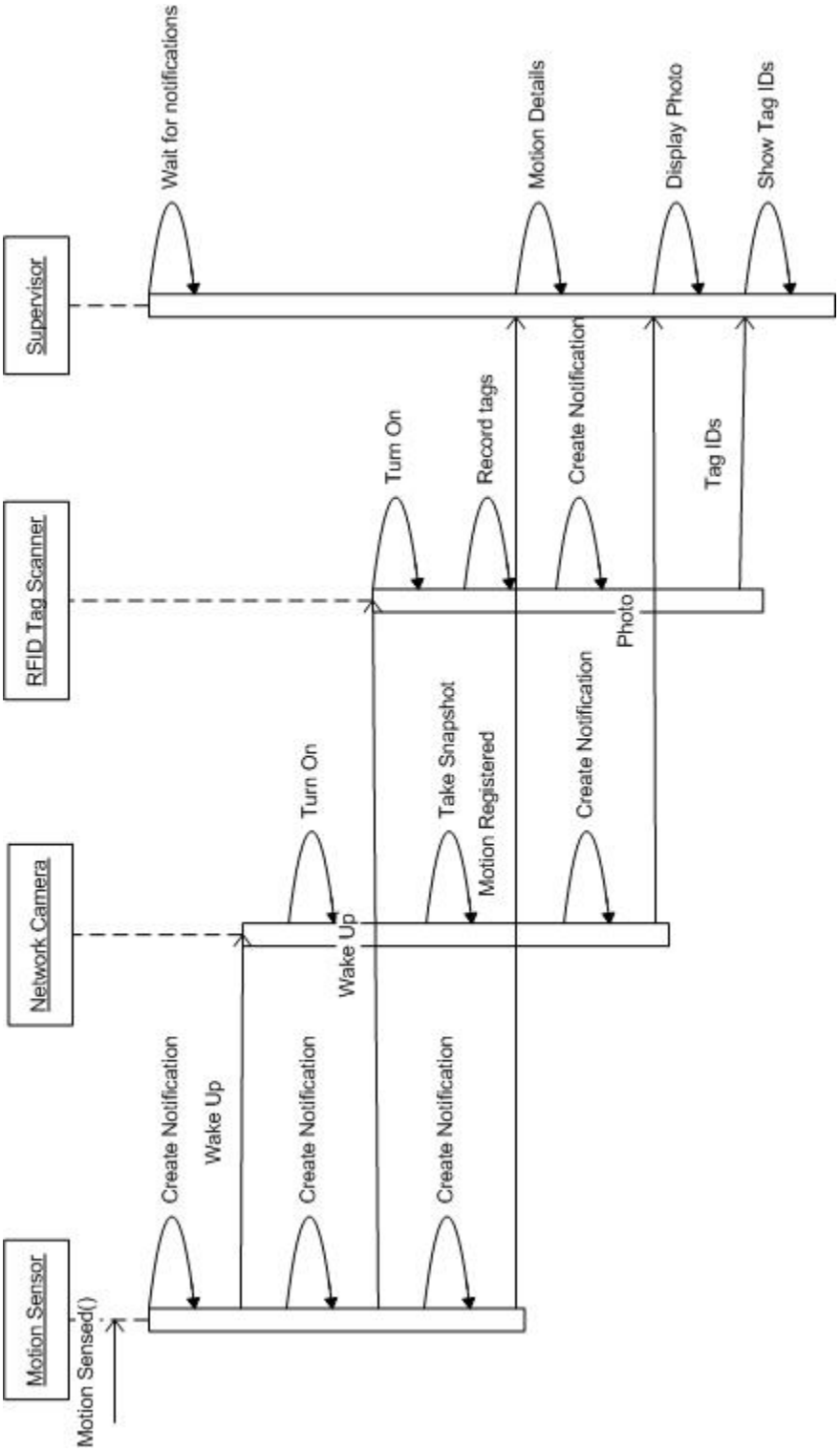

Figure 2.17: Diagram of possible security policy.

CHAPTER 3: SYSTEM IMPLEMENTATION

3.1 Introduction

This chapter focuses on the implementation of the system architecture and design considerations of Chapter 2. The implementation process concerns the translation of the designs and specifications for system architecture into working hardware and software.

Implementation required careful selection of the type of hardware and software capable of realizing the system objectives and requirements discussed in the previous chapter. Specifications of hardware and software components and relevant details regarding their working are provided. Discussion will cover the choices available for each case, the relative benefits and drawbacks posed by each, the metric of evaluation of a suitable component and the rationale for the final selection. Integration of components into the system as a whole is an important stage of implementation, and due consideration is given to stages of incorporating different modules. Individual components were built and tested in stages, first in a stand-alone setting and then integrated into the system and tested in that setting as well.

The testing of the completed system follows, in order to prove that the system demonstrates the design objectives and requirements outlined in Chapter 2. Consideration is given to compromises that were made in transition from design to realization, due to constraints of hardware and software that were encountered. This also examines issues and obstacles encountered during the implementation process and their resolution.

3.2 Hardware Selection

The DIS architecture considerations of the design process required the use of distributed hardware in order to realise specifications and requirements such as flexibility, scalability, robustness and cost. This also facilitated the realisation of distributed control of individual devices, with individual holons and agents residing on each HU.

Consideration was given to the nature of the hardware hosting the holon and agents and interfacing with the associated device. An initial option given consideration was the use of a single controlling device, with adquate resources of memory and processing capability. . To provide modularity in the design, the system was divided into two parts, one responsible for intelligence and decision making and the other tasked with interfacing with devices as well as handling intra-HU and inter-HU communications. A major advantage presented by this configuration is that a dedicated controller with multiple interfacing capabilities allows easy attachment and removal of devices from the HU. These two components will be henceforth referred to as the 'Holon Board' and 'Communications/Interface Board' respectively. Figure 3.1 shows the mapping of architecture specifications of Figure 2.3 and Figure 2.4 from Chapter 2, to the implemented hardware of a typical HU.

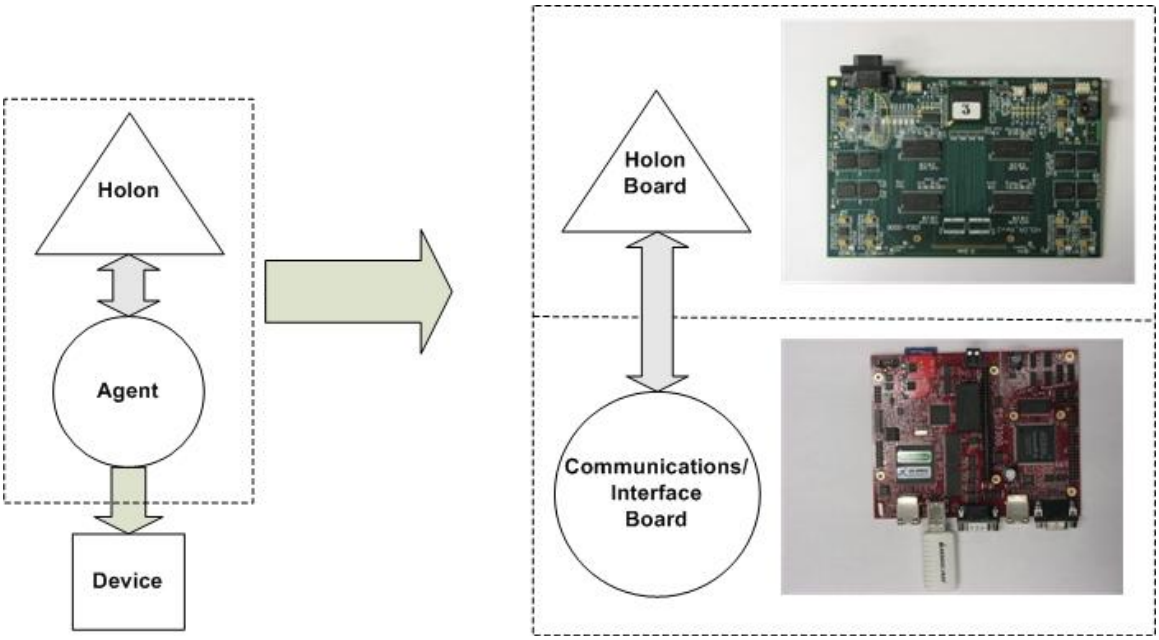

Figure 3.1: Mapping of architecture specifications to hardware.

3.3 Holon Board

The Holon Board provided the intelligence services of the HU. It executed almost all the control and intelligence logic, housed the Holon and agents, and processed incoming messages from other HUs. Hence, the processor used was required to be more powerful, as well as possessing more system memory than the Communication/Interface Boards. It emerged from the research work conducted by the

iDEA Laboratory [91], and was designed and fabricated by Ovcharenko [105]. Its initial purpose was to run all the software of the HU, and handle a lot of data throughput generated by operations related to wireless communication and video. Figure 3.2 shows the Holon Board.

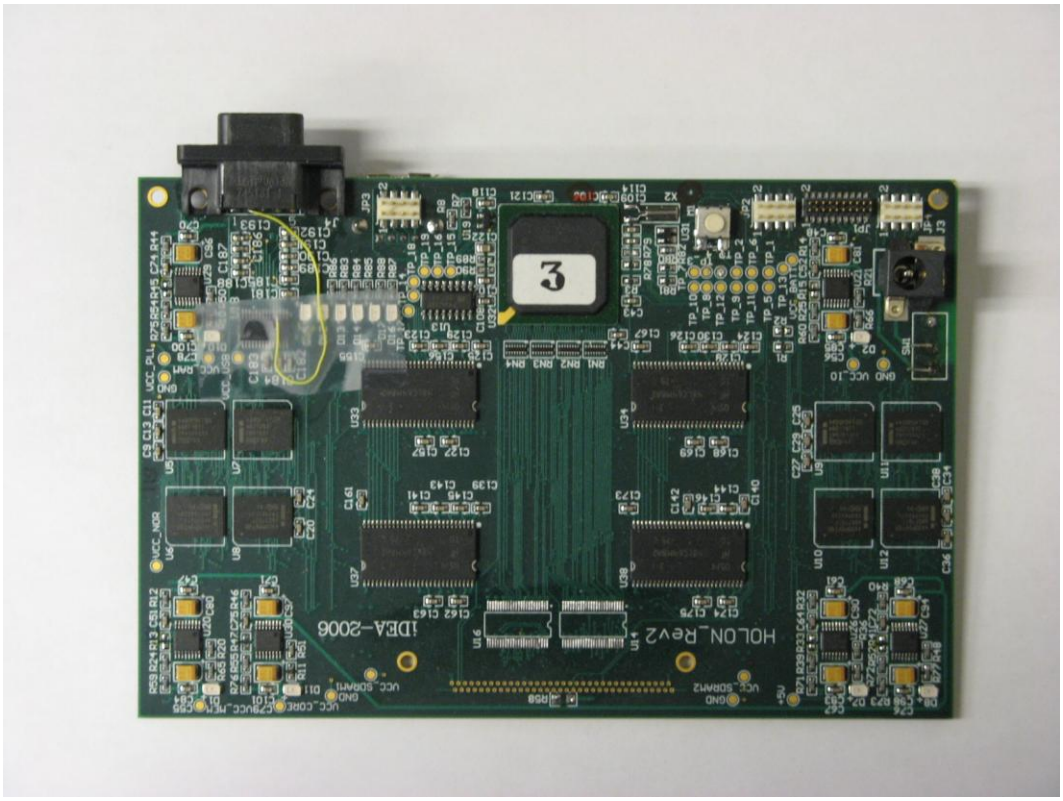

Figure 3.2: Holon board of the Holonic Unit.

Initially conceived as a platform meant to implement embedded systems communicating over wireless networks, the Holon Board is not a commercial off-the-shelf product. It does not support many peripheral interfaces for communication with devices, possessing only one standard serial interface communicating using RS-232 and one USB interface to connect devices. It possessed an IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture [106] hardware interface, also known as Joint Test Action Group or JTAG. It facilitated debugging and testing of the hardware of the Holon Board. The Holon Boards of the Motion Sensor, Network Camera and RFID Tag Reader HUs interfaced to their Communications/Interface Boards through their USB interfaces. The Holon Board was a blank device, and contained no on-board software; it therefore required an operating systems package before any DIS software components

could function on it. The working operating system implemented on the Holon Board was a Linux distribution specifically built for it using the OpenEmbedded software build tool [107]. OpenEmbedded helps build Linux distributions for embedded systems and help maintain them. Ångström [108], the distribution built for the Holon Board is another embedded version of the GNU/Linux operating system. Ångström provided the necessary core Linux kernel as well as a few essential tools and utilities necessary for limited development and execution of software programs. However, Ångström does not possess the full features of the Linux distribution provided with the Communications/Interface Board. It is also limited in terms of the number and features of the tools and utilities provided with it. One of the constraining factors that dictated the extent of the features built into Ångström was the limited storage space available on the Holon board.

In addition to the provision of an operating system, a major decision concerned the selection of a Java Virtual Machine (JVM) [109, 110] as part of the associated supporting software tools provided with Ångström. There were a number of options available in terms of currently available architectures of the JVM – proprietary virtual machine implementations provided by both commercial vendors as well as implementations provided by the open-source community. The main implementations evaluated for their capacity to execute the holon-agent architecture were:

- Java [111] from Sun Microsystems (Sun Java)
- Kaffe OpenVM [112]
- Cacao [113]
- JamVM [114]

Of these implementations, Sun Java is a commercial and proprietary product. The other three (Cacao, Kaffe, and JamVM) are implementations that attempt to be compatible with the official architecture specified by Sun Microsystems for the Java Virtual Machine. Development and implementation stage of the DIS system initially began on version 1.4 of the Java Development Kit (JDK) and Java Runtime Engine (JRE) from Sun Microsystems (Sun Java 1.4). This decision was made as at the time of development, Sun Java 1.4 was the most ubiquitous in terms of usage by industry, in addition to being well understood and supported by the community of developers making use of it.

As development work progressed, project requirements dictated that the source code of the JVM be available for inspection as well as modification, if necessary. As any JVM implementation provided by Sun Microsystems is proprietary by nature, the source code is not available for inspection and modification by the user. Hence, the decision to use one of the various OSS alternatives previously listed. While evaluating the options available, the criteria for selection was based on the constraints of memory and processor usage, as well as storage space of the Holon Board, which would execute the intelligence services as well as the JVM. Running on an embedded system, the JVM would need to consume as little storage space as possible, along with the other software that would also be stored on the Holon Board. It would also need to consume as little memory while the system is running, as memory available on the Holon Board is limited, and other software components required sufficient memory to be available in order not to compromise system performance.

Kaffe OpenVM is the oldest and most mature open-source offering of a JVM, being the original effort by the open-source community to provide a JVM for application development and implementation. Kaffe in its fundamental development is be lean and portable, but does suffer a drawback in terms of performance – it is significantly slower in terms of execution speed than offerings from commercial vendors. In spite of this, it possesses value for embedded systems development, because of its smaller size in comparison to a standard JVM from Sun Microsystems, as well as making use of Just In-Time (JIT) execution capability to run native code.

Cacao (also known as cacaovm) is a JVM that places emphasis on speed of execution, making use of the principle of JIT execution, like Kaffe. First released in 2004 under the GNU Public License, it has been under active development since then. While Cacao did seem to promise speed of execution, there were compatibility issues that precluded its usage as the JVM on the Holon Board.

JamVM was the final option considered for usage on the Holon Board. It is extremely compact as compared to other JVM implementations, while attempting to conform to the Java Virtual Machine Specification, published by Sun Microsystems. In addition, it also employs the Just-In-Time (JIT) execution technique in its compiler, allowing speed of execution of java code. In addition, it is tested and currently supported on a variety of different operating system and processor combinations. Given these features, it was finally decided to use JamVM as the working JVM on the Holon Board.

Table 1 shows the various options considered for the working JVM in terms of their advantages and disadvantages for use in this particular system. Once JamVM was selected as the working JVM, it was kept throughout the course of the project. There were no upgrades as newer versions became available. This avoided subtle errors that might have crept in because of changes made to the JamVM architecture from one version to the other. Such errors are difficult to track and deal with during development. In addition newer versions are less familiar (and hence less well supported) by the development community and documentation will not exist regarding issues encountered and solutions to the same.

Table 3.1: Comparison of various JVM implementations.

JVM Implementation	Advantages of Use	Disadvantages of Use
Sun Java	Commercially supported Well documented	Proprietary source code Requires more storage space
Kaffe OpenVM	Open source Oldest, mature OSS JVM Lean and portable	Insufficient documentation Smaller user base Slower than commercial JVMs
Cacao	Open source	Insufficient documentation Small user base Compatibility issues
JamVM	Open source Speed of execution Widely ported and tested	Insufficient documentation Small user base Relatively new

3.4 Communications/Interface Board

A Communications/Interface Board interfaced with a device on a low level, read output signals from it and supply input signals to control it, if needed. In addition, each Communications/Interface Board possessed wireless communication capability, allowing the formation of a wireless peer-to-peer network. It acted as the communications unit between the Holon Board and its associated device(s), as well as between HUs. A

Communications/Interface Board connected to a Holon Board, forming a two-element holon-agent unit for device control. Figure 3.3 is a UML Use-Case diagram of the Holon board acting on the Communications/Interface Board.

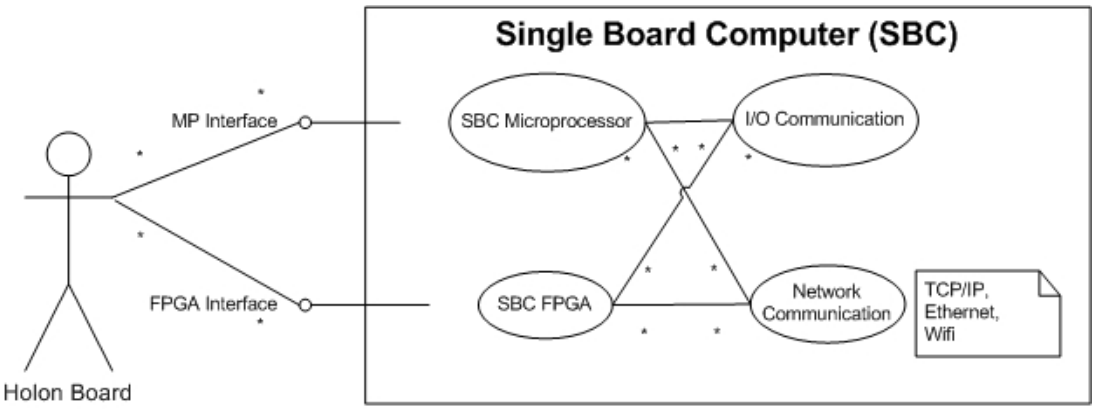

Figure 3.3: UML Use-Case of Holon Board acting on Communications Board.

One practical factor that necessitated this implementation strategy was the limitations of the Holon Board in terms of the hardware interfaces available for I/O as well as hardware resources such as memory, storage and processor speed. In addition, the USB interface available on the Holon Board could only work in client mode with other devices. Therefore, the implementation strategy dictated a device possessing multiple hardware interfaces and a USB interface capable of working as a host in order to initiate control of USB devices. The device chosen to realize this functionality was the TS-7300 Single Board Computer (SBC) provided as a commercial off-the-shelf product by Technologic Systems. Figure 3.4 shows an image of the TS-7300 SBC.

Of the various peripheral interfaces available, the following implemented I/O functionality between the TS-7300 and device or with the Holon Board:

- Secure Digital (SD) Card interface #1
- Ethernet interface #1
- Digital I/O interface #1 (DIO1)
- Serial (COM 1 DB9) interface
- USB interface

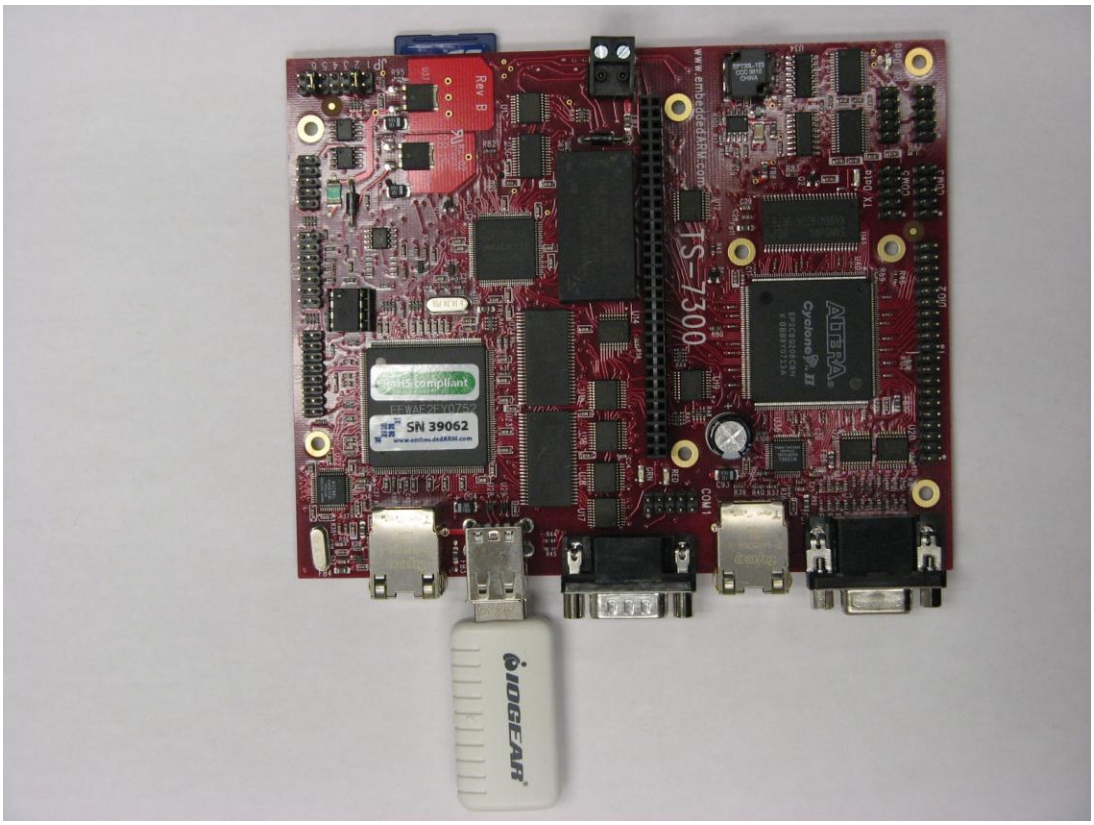

Figure 3.4: The TS-7300 SBC from Technologic Systems.

The SD Card interface stored the operating system and supporting software. The Ethernet interface communicated with the network camera. The motion sensor connected to two pins of the DIO1 [115] interface. The Serial (COM1 DB9) interface (which communicated using the RS-232 protocol) was used during development to load software onto the SD Card and test them. The USB interfaces supplied power to and operated an IOGEAR 54g Wi-fi USB Adapter [116]. This device operated using the IEEE 802.11 b/g [117, 118] wireless communications protocol and enabled wireless communication between the HUs of the DIS security system. Communication between each TS-7300 SBC and its respective Holon Board used the host USB interface on the TS-7300, as the USB interface on the Holon boards could only operate in client mode. The TS-7300 SBC possessed two slots capable of accommodating SD cards of 1 GB capacity or more. This provided enough storage space for a full-featured operating system and other software programs. The Holon Board lacked this type of hardware interface.

The tools and utilities supplied with the embedded operating system of the TS-7300 SBC enabled much of its functionality. Although there are many embedded operating systems feasible for use in this DIS, a major constraint that influenced decision-making was the usage of open-source software (OSS) [119] for development and implementation wherever possible. Since OSS makes the source code available to the end user, it becomes easier to compile and deploy the software on hardware platforms specially selected or developed for a particular application, such as the DIS security system. Extensive modification of the source code is possible to meet specialised requirements of the system and there are numerous choices available with regard to tools that implement a desired feature or functionality. In this regard, the TS-7300 SBC used an embedded version of the GNU/Linux [120] operating system. This was a suitable choice, due to the increasing use of GNU/Linux in real-world commercial applications of embedded systems. It is deployed on numerous hardware platforms and documentation is extensively available with respect to development and tools available for the same.

The TS-7300 used two different GNU/Linux distributions, selection of which depended upon the user's requirements. One is TS-Linux [121], a compact Linux distribution based on BusyBox [122, 123]. TS-Linux is suitable for implementing systems that must be compact in terms of storage space, and provides the most essentially used tools and utilities. However, in endeavouring for compactness of size and storage space occupied, these tools and utilities often lack all the features of the complete versions. In the event that a complete Linux distribution is required for development, the other operating system of choice was a full Debian Linux [124] distribution with full versions of associated tools and utilities. These were essential for system development and testing. The version of Debian Linux used on the TS-7300 was version 3 (also known as 'Sarge'). Of the two, implementation used Debian Linux as the working operating system.

Due to the software requirements of the DIS, Debian Linux functioned as the operating system for the scope of my project. A JVM was not suitable for use on the TS-7300, due to the relatively demanding memory and processor requirements. The TS-7300 possessed limited memory capacity in comparison to the Holon Board, and a JVM would adversely affect the communications and interfacing operations executed.

3.5 Equipment

The four interacting devices of the DIS security application – motion sensor, network camera, RFID tag reader and supervisor, interfaced with a TS-7300 and together with their respective Holon Boards, formed a HU. All four HUs communicated with each other over a wireless network. Figure 3.5 shows the various modules of the system hardware. The Supervisor HU in Figure 3.5 did not interface with a Holon-Communications/Interface unit; it possesses the capacity to support the functionality of intelligence, communications, interfacing, and control. Implementation details follow.

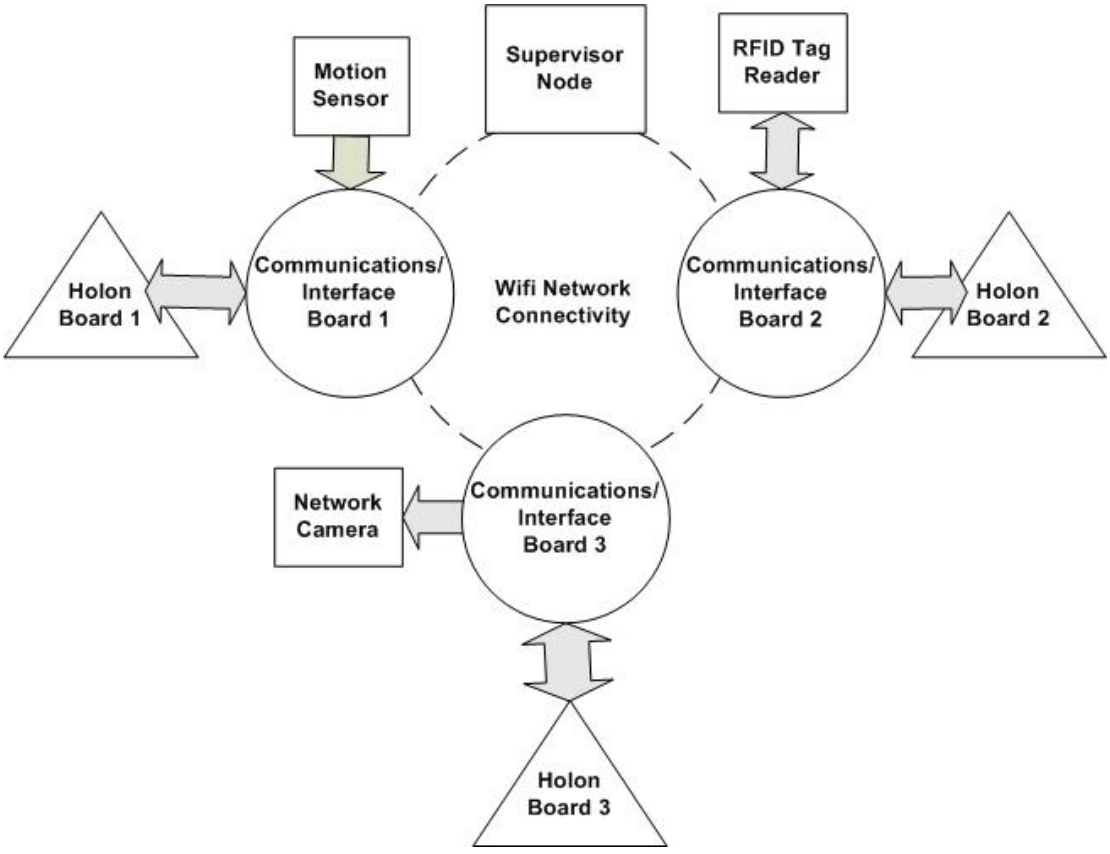

Figure 3.5: Block diagram of DIS security system hardware.

The motion sensor was a commercial off-the-shelf device. It made use of an infrared sensor to detect motion, and an LM324N [125] operational amplifier (op-amp) to process the signal generated by the infrared sensor. The LM324N delivered an output signal that went to +5V when the infrared sensor detected movement and a 0V signal otherwise. Figure 3.6 shows the electronic board inside the motion sensor, containing

the infrared sensor and LM324 op-amp. Figure 3.7 shows the output signal from the LM324N as displayed on an oscilloscope when a series of motions occurred in front of the motion sensor. The TS-7300 SBC received the signal from the output of the LM324N through its Digital I/O interface. Figure 3.8 shows the completed interfacing of the two components.

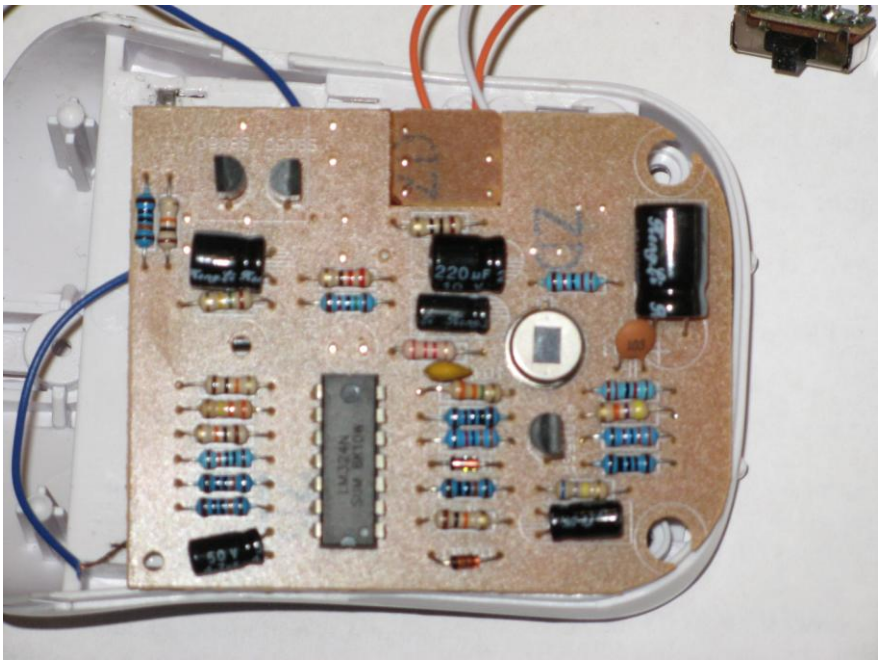

Figure 3.6: Motion sensor with infrared sensor and LM324N.

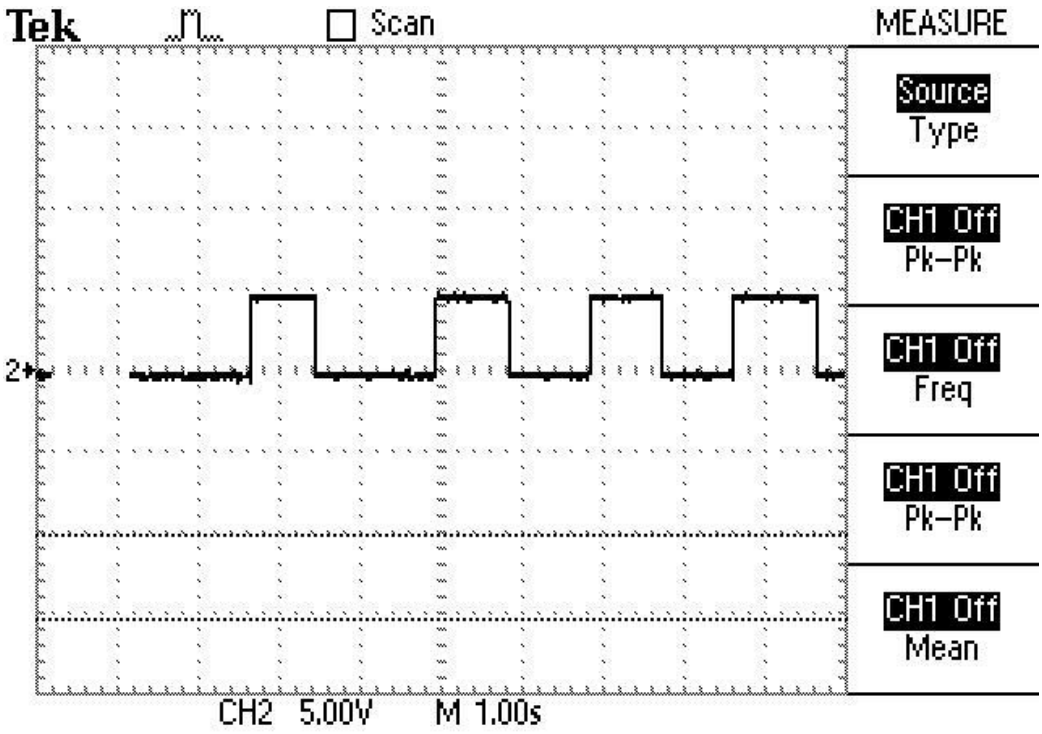

Figure 3.7: Output signal from LM324N of motion sensor.

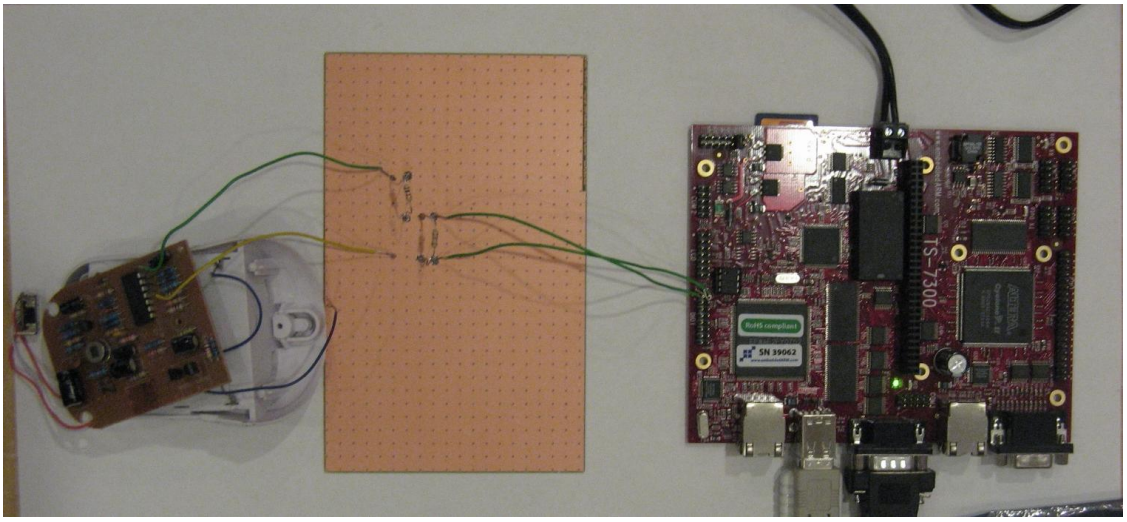

Figure 3.8: Completed interfacing of motion sensor and TS-7300.

The RFID tag reader used was the M5e development system from ThingMagic Inc. [126, 127], operating in the UHF spectrum and making use of an Application Specific Integrated Circuit (ASIC) for its RFID reader chip and an Atmel ARM7 microcontroller for control of the device. Figure 3.9 shows the M5e with its associated antenna and a Gen 2 RFID tag.

Figure 3.9: RFID reader.

The M5e also possessed a serial interface that communicated using the RS-232 protocol. This connected to the serial interface on the TS-7300, and controlled using an RFID subsystem software component. This was necessary, as the RFID software component required a JVM in order to operate.

The network camera selected for use in the system was the Axis 210 [128] from Axis Communications. The Axis 210 is a professional grade surveillance camera that communicates using the Internet Protocol (IP), supporting IPv6 as well as IPv4. It is able to support advanced functionality due as it contains an embedded BusyBox Linux distribution. This enables it to provide useful services such as an embedded web server, a built in scripting tool for executing scripts, secure File Transfer Protocol (FTP), telnet, etc. The Axis 210 interfaced to the Ethernet port on a TS-7300 SBC using a crossover

cable. Figure 3.10 shows the Axis 210, TS-7300 and Holon Board forming the Network Camera HU.

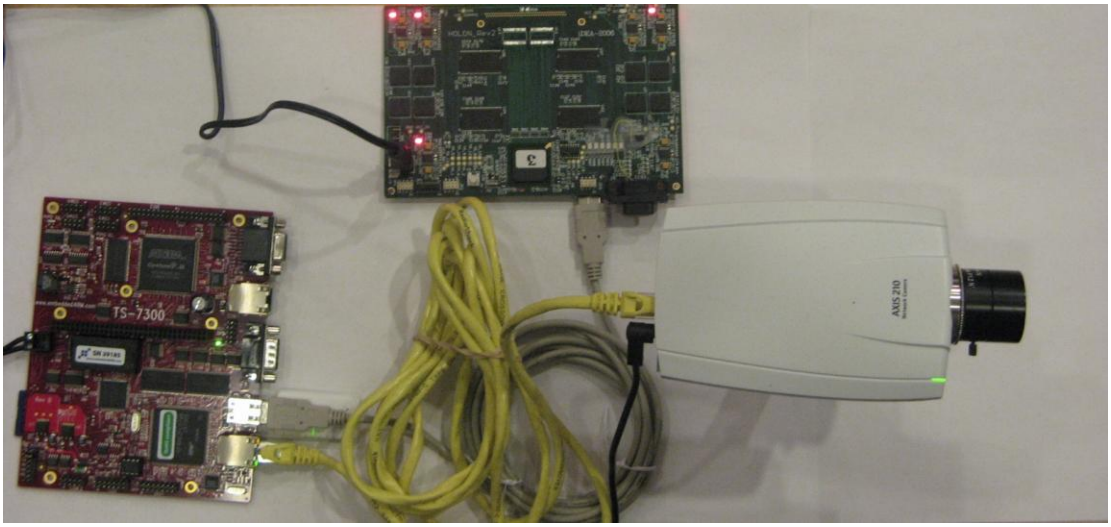

Figure 3.10: Interfacing of Axis 210, TS-7300 SBC and Holon Board.

The Axis 210 interfaced with the TS-7300 because it used an Ethernet interface to communicate, and this was not available on the Holon Board. In addition, the Angstrom OS did not have the necessary supporting tools to automate the login procedure to BusyBox on the Axis 210.

3.6 Software Selection

Having established some of the specifics of the system hardware, this Section will investigate the implementation of the software components helping to meet the architectural requirements of the DIS as outlined in Chapter 2. Whenever possible, I attempted to use already existing software tools that implemented necessary functionality within the system, in order to reduce development and implementation time.

DIS components on the Holon Board implemented the architecture of the holon-agent model. Among the options available, the Java programming language was determined to meet these requirements. As Java makes use of a Java Virtual Machine, object code implementing the holon-agent model should theoretically be hardware-independent, i.e. the same object code should be able to run on HUs making use of different hardware platforms with little or no modification. While essentially a high-level

language, Java possesses the capability to interface with low-level program code written in C through the Java Native Interface (JNI). Finally, one important factor that influenced the selection of Java for implementation of necessary DIS components was the already existing Holonic Technology Platform (HTP), which had previously implemented many of the foundational components of the holon-agent model and did not require extensive modification for the scope of this work.

3.7 Holonic Technology Platform (HTP)

Chapter 2 provided a brief mention of HTP as an alternative, agent-based framework for developing a DIS, based upon an underlying peer-to-peer architecture. HTP's stated system objectives [75] at the time of development were:

1. To create an environment where each individual node on a network is able to work with other network nodes, in accomplishing a global task. In such an environment, the individual must also accomplish its own task.
2. To allow each individual node on a network to have the capability to learn new responsibilities and new sets of functionality, and to have the capability to adapt to a changing network environment.
3. To create a system architecture where other system developers are able to expand the functionality of the system without having to make changes to the core of the environment.

The initial implementation of HTP met these objectives. The architecture was organised such that it comprised of twelve agents organised under the Holonic Logistics System (HLS) [75,], as shown in Figure 3.12

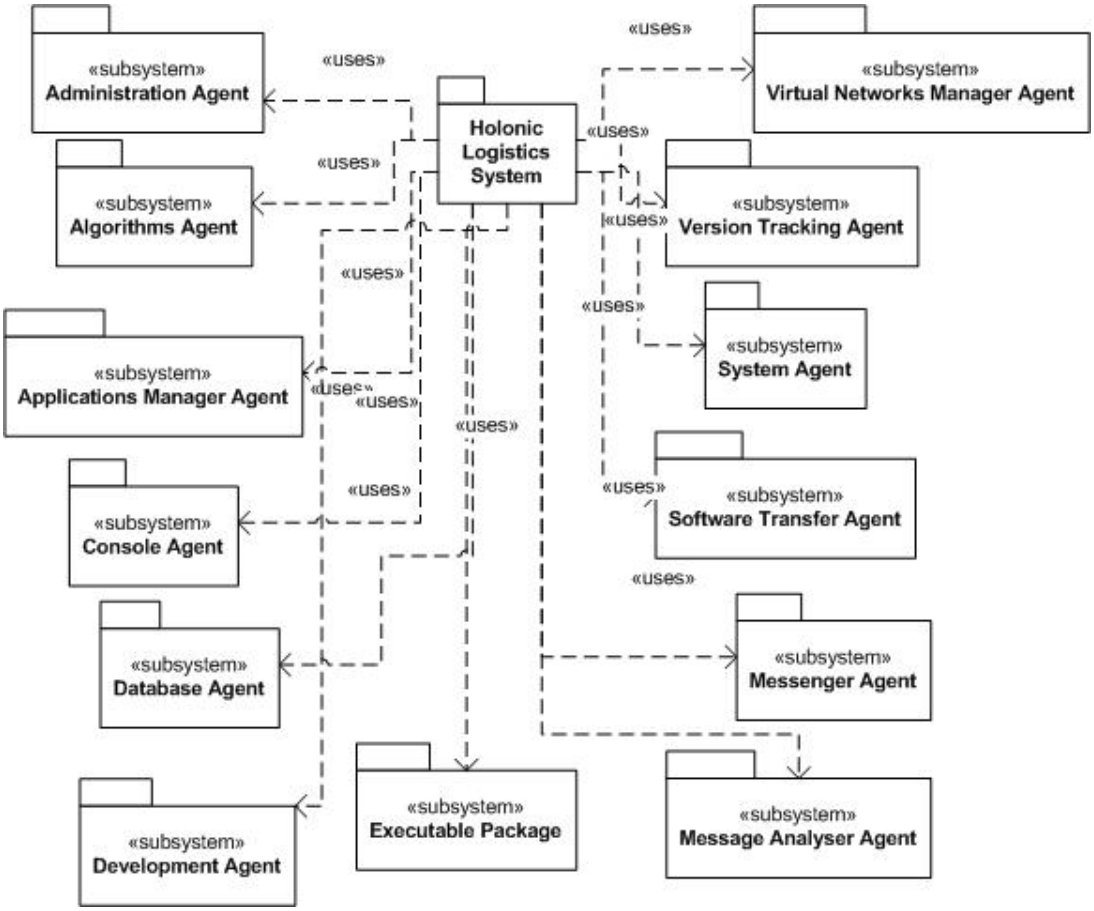

Figure 3.11: Architecture of Holonic Logistics System under HTP.

The initial HTP framework laid the foundation using the holon-agent model, upon which various holonic applications were possible. However, it lacked the necessary functionality to meet the requirements of an application performing distributed control of devices and handling interactions between them. Specifically, it did not possess components such as the Device Agent constructed upon the KRB model, the Agent Registration Table or Holon Event Table outlined in Chapter 2. Hence, modifications were necessary to the HLS that incorporated these features. Figure 3.13 illustrates these changes.

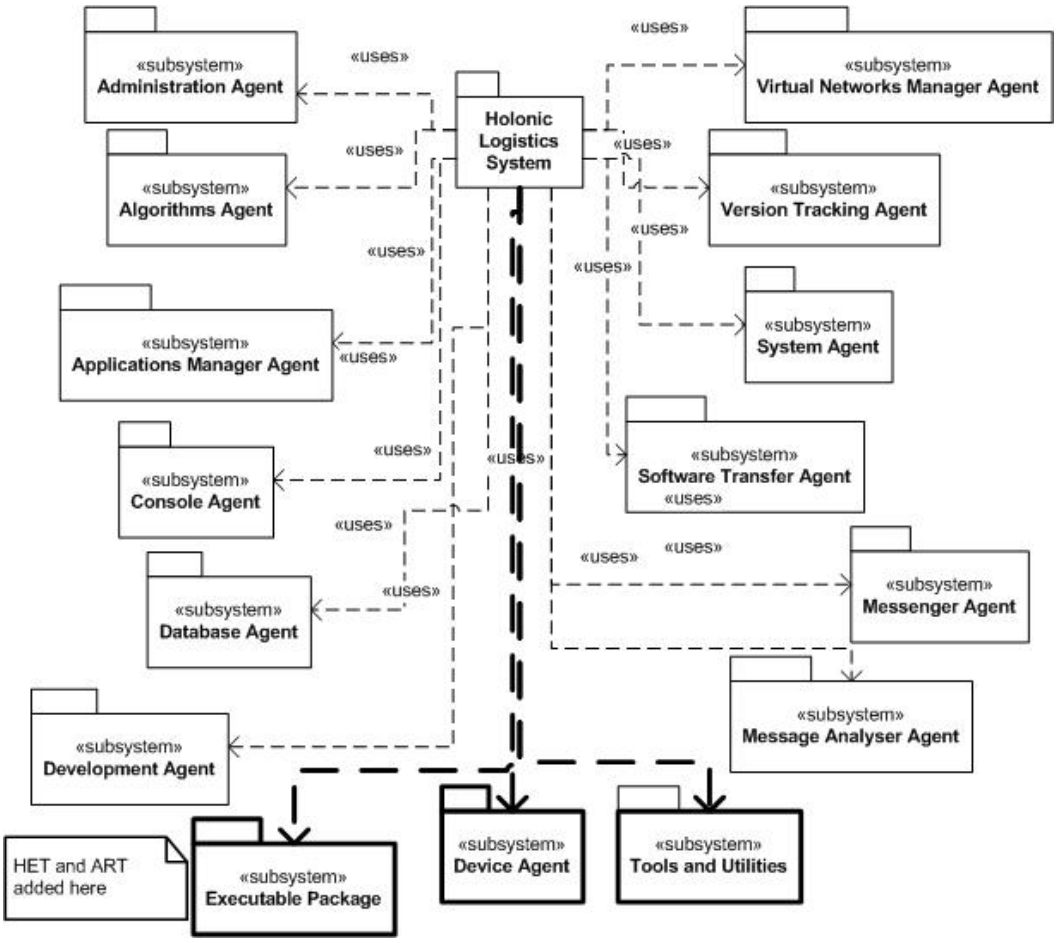

Figure 3.12: Modifications made to HLS architecture.

The Device Agent was the major modification made to the HLS, and implemented in Java. Its implementation was such to allow it to operate within the HTP framework; this required minor architectural modifications to the HTP source code. Figure 3.14 illustrates the architecture of the Device Agent.

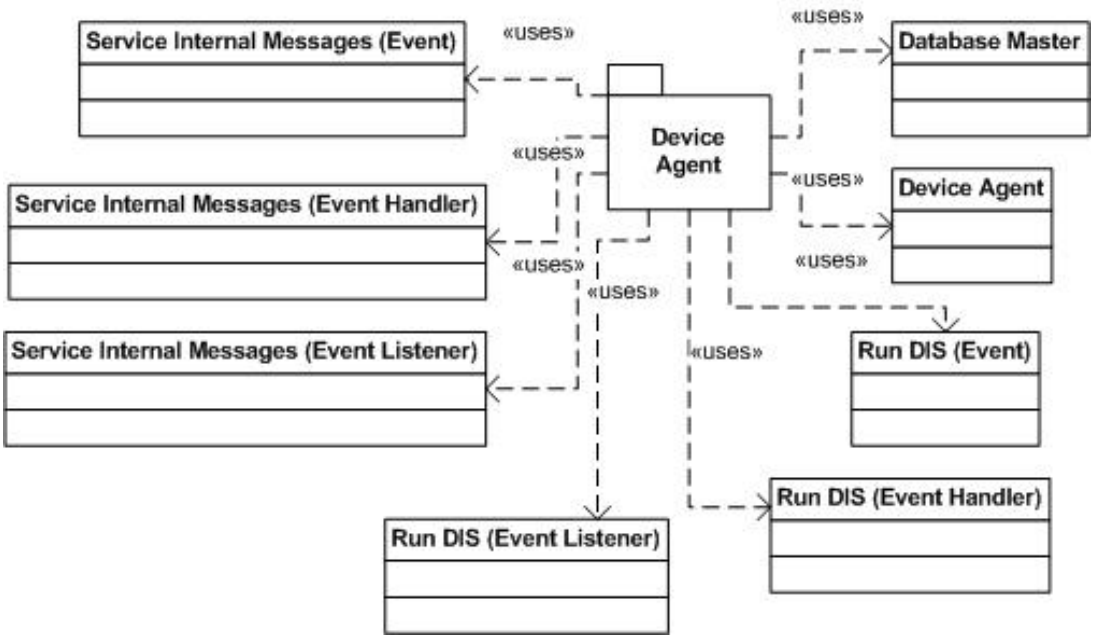

Figure 3.13: Architecture of the Device Agent.

The objective of the Device Agent was to anticipate incoming event-triggered messages from devices regarding device state or data generated by the same; determine the event that generated the message by consulting the HET; process the message and respond with the appropriate messages instructing control action, in accordance with a user-specified rule base.

The subsystem implementing the Device Agent made use of some of the tools developed within the 'Tools and Utilities' package. This was useful in manipulating the configuration file of the Device Agent. The 'DatabaseMaster' and 'DeviceAgent' files are compulsorily required for registration with HTP. Event description files such as 'Run DIS (Event)' specified the name of the event and registered it with HTP. These needed to operate in concert with event listener files, tasked with listening for notifications that the event specified had occurred. The associated event handler files contained detailed instructions of appropriate course(s) of action upon notification of the event.

Implementation of program code for the Device Agent followed the KRB model outlined in the section on its design in Chapter 2. In order for a particular Device Agent to initialize and configure to the specific HU and device that it monitored, it opened and read a configuration file containing all the necessary parameters and their corresponding values needed for the Device Agent to configure itself. This configuration file was stored

in the simple and popular text file (.txt) format. However, in order to implement more complex and demanding applications requiring more sophisticated and powerful schema for information representation, other formats such as XML are possible implementation options. Listing 3.1 shows a section of code that was part of the initialisation and configuration process. For the full source code listing, please see the Appendices.

```
public void AssignConfigurationParameters()
  {
retrievalOfAgentConfigurationParameters = new
Package_ToolsAndUtilities.PackageToolsAndUtilities_FileParamete
rRetrieval();


 //Retrieve and set Agent Parameters
System.out.println("Retrieving Device Agent Identity
Parameters...");
NAME_OF_PARENT_HOLONIC_UNIT =
retrievalOfAgentConfigurationParameters.ExtractRequestedParamet
erAsString(parserOfAgentConfigurationFile.InitializationFilePar
ameters, "PARENT_HOLONIC_UNIT");
DEVICE_AGENT_ID =
retrievalOfAgentConfigurationParameters.ExtractRequestedParamet
erAsString(parserOfAgentConfigurationFile.InitializationFilePar
ameters, "DEVICE_AGENT_NAME");
DEVICE_AGENT_IDENTIFICATION_NUMBER =
retrievalOfAgentConfigurationParameters.ExtractRequestedParamet
erAsInteger(parserOfAgentConfigurationFile.InitializationFilePa
rameters, "DEVICE_AGENT_IDENTIFICATION_NUMBER");
System.out.println("Retrieving Message Location
Parameters...");
LOCATION_OF_MOTION_SENSED_MESSAGE =
retrievalOfAgentConfigurationParameters.ExtractRequestedParamet
erAsString(parserOfAgentConfigurationFile.InitializationFilePar
ameters,"PATH_TO_MOTION_SENSED_MESSAGE");
…
} //This is the end of the method definition
```

Listing 3.1 – Example of Device Agent configuration process.

Typical information that was stored in the configuration file included: Device Agent name and identification number; connectivity information, such as IP addresses of

other HUs in the neighbourhood; location of incoming and outgoing message files; operating system specifications; rules of action; behaviour specifying action to be implemented when conditions meeting those rules. More complex forms of representation could use XML or other methods to specify the configuration and rule base for each agent, implemented as the system architect and developer sees fit. Listing 3.2 shows the contents of a sample configuration file. The full source code is provided in the Appendices.

```
//Main section Deliminators = ::
//Subsection Deliminators = :
<<DEVICE_AGENT.MAIN_SECTION>>::
<SUBSECTION.IDENTITY>:
PARENT_HOLONIC_UNIT=MotionSensorHolonicUnit
DEVICE_AGENT_NAME=MotionSensorDeviceAgent
<SUBSECTION.CONNECTIVITY>:
MOTION_SENSOR_HOLONIC_UNIT=192.168.0.50
<SUBSECTION.RULES>:
PRIMARY_INCOMING_MESSAGE_TO_MONITOR=
/DISC/Messages/MotionSensed.txt
<SUBSECTION.BEHAVIOUR>:
FIRST_COURSE_OF_ACTION=./ftpToRFIDReaderHolonicUnit.sh
SECOND_COURSE_OF_ACTION=./ftpToSupervisor.sh
```

Listing 3.2 – Sample Device Agent Configuration File

Once reading of the configuration file was completed and the Device Agent's internal variables assigned the corresponding values specified in the configuration file, the agent would wait for an internal message from HTP notifying it of some event that required its attention. The listening, notification and handling of the event made use of Java's internal event service. Through the messaging system made available by HTP [75], Device Agents were able to send internal messages to themselves, as well as external messages to other Device Agents residing on other HUs.

When either an internal or external arrived for the Device Agent, it would fire an event handling mechanism written for that particular event. The Agent would then execute the course of action specified in the event handling mechanism procedure. Listing 3.3 shows a segment of the code related to event handling where the Device

Agent receives an instruction to start working. As another example, an external message from the Device Agent on the Motion Sensor HU would arrive for the Device Agent on the RFID Reader HU, informing it to commence reading the ID's of tags within its sensor range. HTP internally routed this message to the Device Agent, triggering an event handling mechanism to take the appropriate action of beginning scanning for RFID tags within its sensor range. Figure 3.14 illustrates this as a UML sequence diagram.

```
public void
RunDISEventReceived(PackageDeviceAgent_RunDISEvent Event){
System.out.println("This is Device Agent, confirmed
working.");
System.out.println("Starting up DISC now.");
ManipulateAgentConfigurationFile();
AssignConfigurationParameters();
CheckIncomingMessageFolderForNewMessages = new
Thread(RunThreadCheckingIncomingMessageFolder, "DISC Incoming
Message Monitoring Thread");
CheckIncomingMessageFolderForNewMessages.start();
while(true){
try{
CheckIncomingMessageFolderForNewMessages.sleep(100);}
catch(InterruptedException
exceptionThrownWhileMonitoringIncomingMessages){
  CheckIncomingMessageFolderForNewMessages.run();}}}
```

Listing 3.3 – Example of event handling

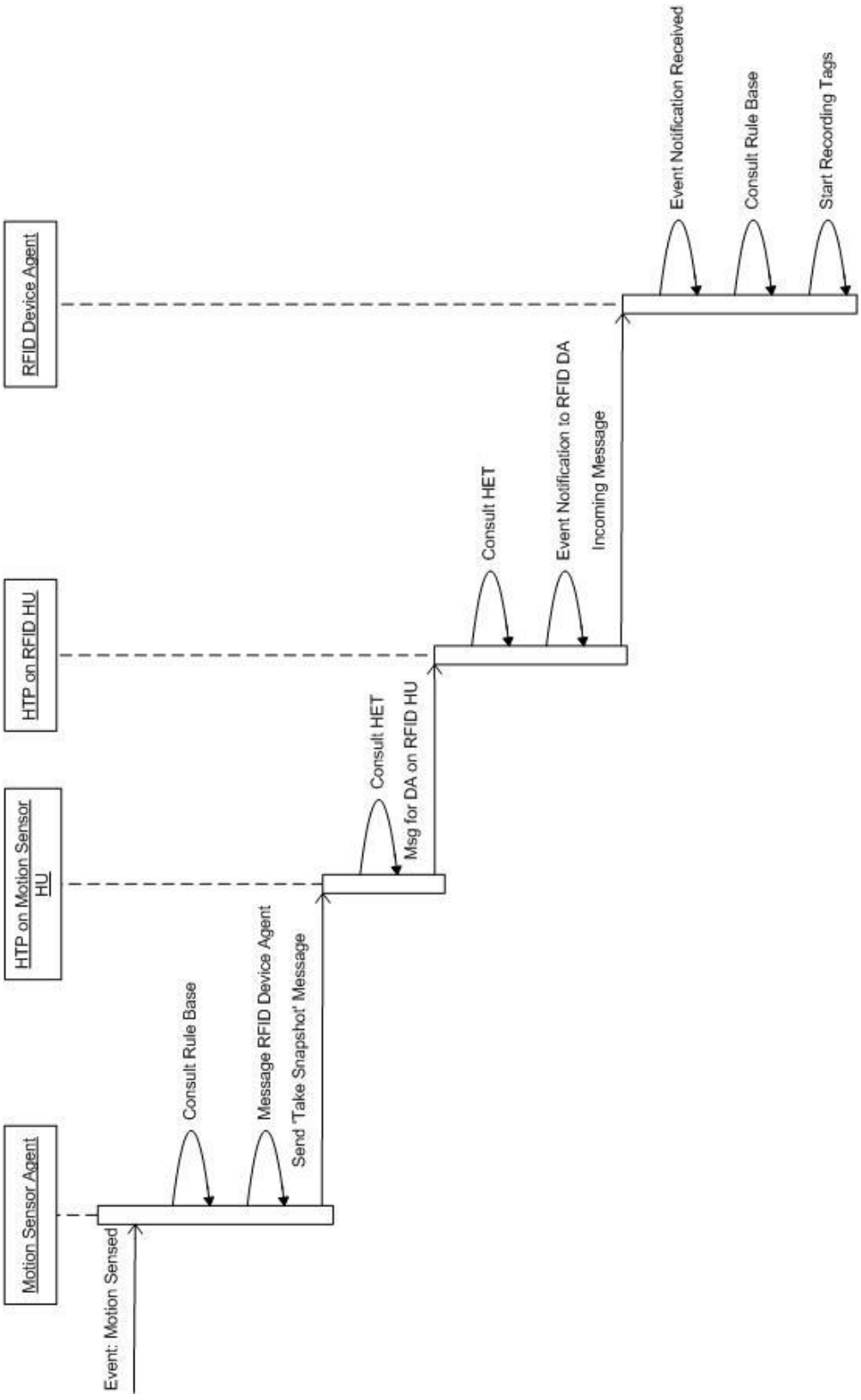

Figure 3.14: Event Handling by the Device Agent.

Thus, there was successful implementation of the abstract concepts of knowledge, rules and behaviour specified in the design and architecture section for the Device Agent in Chapter 2.

The system software running on the Holon Board also used software components facilitating communication with the TS-7300 SBC to which it interfaced. This was addressed by implementing the DIS components handling communications and interfacing for the TS-7300 SBC as general-purpose tools, so that they could be run on the Holon Board to provide the same services with little or no modification.

An instance where scripts written in the Bourne Again Shell (BASH) [129 - 131] automated subsystems of the DIS on the Holon Board was configuration of JamVM to execute java code. Once JamVM was implemented as the working JVM, some parameters had to be configured in order for DIS subsystems implemented in Java to work properly. These include parameters such as the 'PATH' variable (specifying the location of the JVM home directory and important subdirectories) within the operating system on the Holon Board, the setting of the 'CLASSPATH' variable (specifying the location of important JVM classes) of JamVM. Once the proper parameters were set for the JamVM, the proper instructions ran the java program code implementing the various DIS subsystems. A BASH script file contained these important instructions and repeatedly executed as many times as necessary. Listing 3.4 shows a segment of the source code for the script file used to start HTP. For a full listing, please consult the Appendices.

```
#!/bin/bash
#Bash script to run DISC files using JamVM
#Author: Kevin Thomas
#Date of last modification: 2009.06.10

#Java JDK & JRE settings for JamVM
JAVA_HOME=/usr/bin/jamvm:/usr/share/jamvm/
PATH=$PATH:/usr/bin/jamvm:/usr/share/jamvm/

JAVA_FILES_TO_COMPILE="HTPComponentTest.java"
JAVA_CLASS_FILES_TO_RUN="HTPComponentTest"

printf "\nThe current setting for the java PATH variable is:"
echo $PATH

printf "\nThe current setting for the java JAVA_HOME variable is:"
echo $JAVA_HOME

printf "\nThe current setting for the java CLASSPATH variable is:"
echo $CLASSPATH

cd $MAIN_WORKING_DIRECTORY

#Use JamVM to run the .class file(s)
jamvm -Xbootclasspath:/usr/share/jamvm/classes.zip:/usr/share/classpath/gnu/java/locale/:/usr/share/classpath/glibj.zip -verbose:class $JAVA_CLASS_FILES_TO_RUN

#This is the end of the script file
```

Listing 3.4 – Bash script automating configuration and start up of HTP

Apart from the selection of the JVM, the other components implemented in Java were the subsystem that read RFID tag data from the RFID Tag Reader and the Device Agent. Figure 3.15 shows the architecture of the implemented RFID subsystem. All discussion of the implementation will reference this figure.

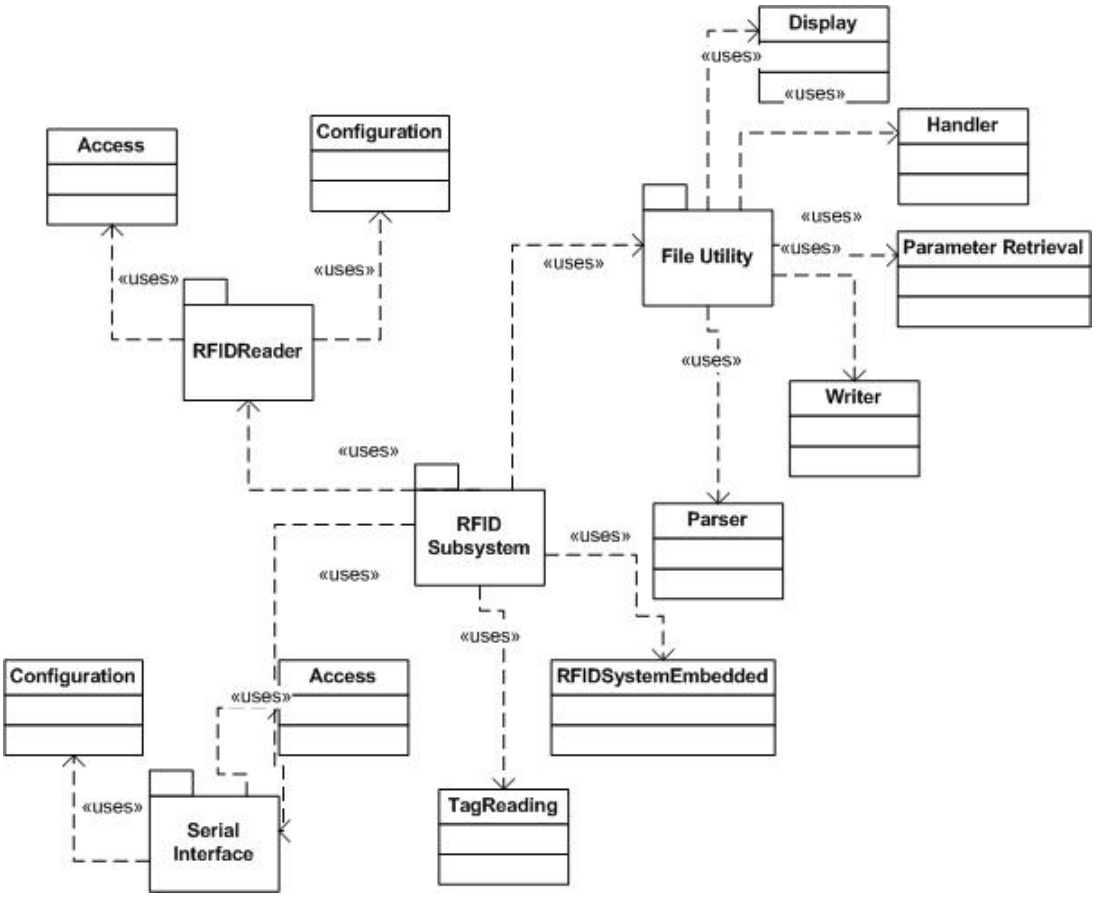

Figure 3.15: Architecture of implemented RFID subsystem.

The purpose of the RFID subsystem is as follows: “To initialize the RFID tag reader, instruct it to read and retrieve the ID’s of any RFID tags that enter its sensor range and store the RFID tag ID into a text file for processing by the Device Agent.”

When implementing components of the DIS, a goal of the development process was to make the implementation as generic as possible, so that tools developed for RFID application could build other applications as well. Thus, the ‘File Utility’ package is a set of generic file manipulation tools to handle text files; the ‘RFID Reader’ package is a set of generic tools to access and configure an RFID tag-reading device; the ‘Serial Interface’ package allows access and configuration of a generic serial interface. The file

‘RFIDSystemEmbedded.class’ is a java class file that invokes these various tools, making use of them to accomplish the purpose of the RFID subsystem. Similarly, the file ‘TagReading.class’ invokes the tools necessary to instruct the RFID tag reader to read the IDs of RFID tags that come within the field of its sensor.

The ‘File Utility’ package consists of a set of java class files to access, parse, retrieve parameters from and write data to a file. The parameter file ‘RFIDPackageOperatingParameters.txt’ configures various software components of the RFID subsystem, such as the serial interface, RFID tag reader, tag recording system and so on. This allows the user to configure the operating parameters of the RFID subsystem without having to modify the source code in order to effect those changes. The ‘Handler’ class file is invoked to accesses the file, open it and close it once it is processed. The ‘Parser’ class file is a tool to parse the configuration file by sections, and extract the various parameter values for processing. The ‘Parameter Retrieval’ class converts the various parameter values extracted by the Parser class into their appropriate data types (string, integer, etc.) for use by the RFID subsystem program code. Finally, the ‘Display’ and ‘Writer’ classes display the various parameters contained in the configuration file, and write data to files, respectively.

The ‘RFID Reader’ package contains two java class files. The ‘Access’ class file facilitates access to an RFID tag reading device, in order to send control signals to it. The ‘Configuration class file enables the tag reader to be configured using device-specific codes before certain functions can be performed. The tools within the RFID Reader package are made use of after the configuration file has been processed and the parameter values extracted.

Similarly, the ‘Serial Interface’ package contains ‘Access’ and ‘Configuration’ java class files that facilitate access to and configuration of a serial interface that communicates using the RS-232 protocol. These tools are used after the parameters are extracted from the configuration file within the File Utility package.

When the RFID subsystem is invoked, the ‘RFIDSystemEmbedded.class’ file uses the ‘Handler’ class file within the ‘File Utility’ package to access the configuration file containing the various operating parameters and their values. The configuration file is parsed into lines. The ‘Parser’ class is used to further parse the extracted information by section (Serial Interface, RFID Reader, etc.), subsection (Serial Port, Baud Rate, etc) and individual parameters and store them in corresponding data structures. The tools

within the ‘Serial Interface’ and ‘RFID Reader’ are used to access and configure the serial interface and RFID tag reader respectively. The ‘TagReading.class’ file then carries out the operation of reading RFID tag IDs. When tags are sensed, the ‘Writer’ class within the ‘File Utility’ package is invoked to write the tag ID to a text file. Figure 3.16 illustrates this in a UML diagram.

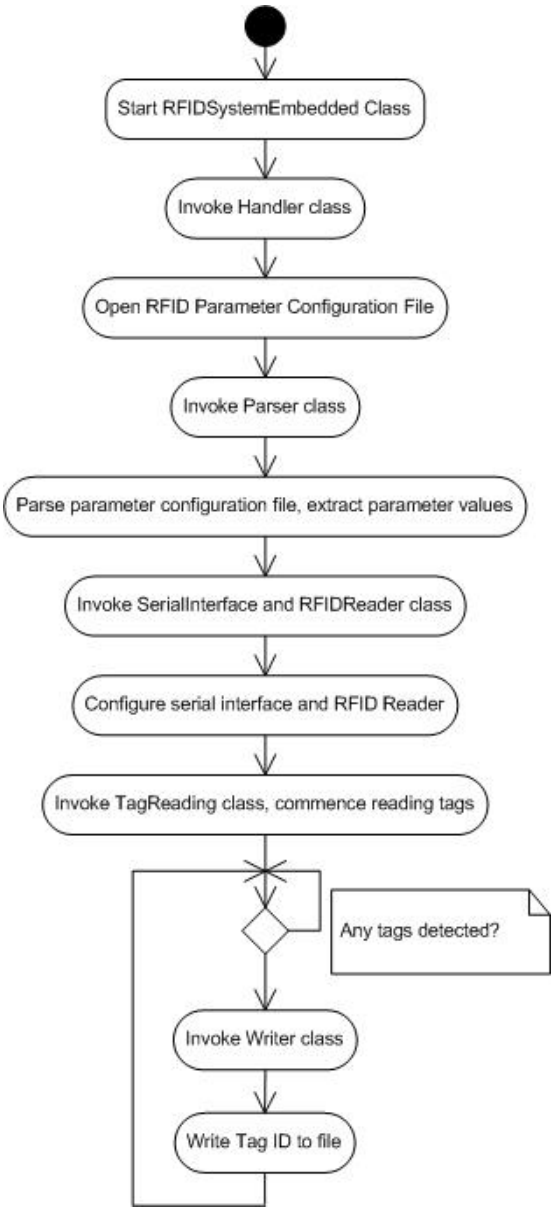

Figure 3.16: UML activity diagram of RFID subsystem operation.

The workstation to which the Holon Board was interfaced for programming purposes made use of version 7.04 of Ubuntu [80] as the working operating system, which is another Linux distribution provided by the open source community. This version of the working operating system on the workstation remained unchanged throughout the entire development and implementation cycle. The reason for this was to avoid subtle problems and conflicts in the working of tools and programs, which would be difficult to detect during development when versions of the operating system and its associated tools were changed or upgraded.

3.8 Communications/Interface Board

One of the operating requirements of the Communications/Interface Board was the capability to handle I/O in the case of devices or a Holon Board. In terms of input from devices, examples of such software functionality were reading signals generated by a device through an interface; encoding signals into a binary representation; packaging data from devices into a message format and transmitting them to the Device Agent residing on the Holon Board. From the perspective of output to devices in order to issue them instructions, software functionality needed convert control commands from a Device Agent on the Holon Board into signals understood by the individual device before transmitting them. In both cases, buffering capability was required for data moving to or from the device attached to the Communications/Interface Board. In addition, functionality was required that would make use of wireless capability, enabling HUs to communicate with one another. All of these capabilities required implementation on a low-level in order for communications and control between the constituent nodes of the system.

Some of the DIS software components implemented on the TS-7300 SBC used the BASH scripting language, an OSS shell program provided with some Unix-like operating systems. A scripting language allows some external control of various application programs or operating system services. A frequent use of scripting languages is to automate repetitive tasks or services, an advantage being modification of programs without having to recompile the source code. In addition, applications and operating system services usually provide interfaces for scripts to use.

For example, a BASH script performed remote connection to the Axis 210 network camera using the Ethernet interface and the telnet utility to execute the BASH

script embedded on it. This script controlled the camera's snapshot-taking capability and transfer of the photo back to the associated TS-7300 SBC. The script made use of an embedded web server to take a snapshot and write it to a local directory on the camera. It also employed an embedded file transfer service to transfer the snapshot to a specified directory on the TS-7300 SBC. Listing 3.4 shows the source code for this script. Another script enabled connection to the Holon Board and conducted the transfer of files between them using the File Transfer Protocol (FTP) [133] implemented by an appropriate tool. Setup of various interfaces for communication with devices and HUs (such as the Ethernet, Wireless and Serial Interfaces) used BASH scripts executed automatically upon start-up of the TS-7300 SBC.

```
#!/bin/bash
#Bash script to log in to Axis 210 Network Camera and request snapshot JPEG image
#Author: Kevin Thomas
#Last modified: 2009.10.19

#Temporarily deactivate wireless interface wlan0
ifconfig wlan0 down
#Temporarily reactivate ethernet interface to Network Camera (eth0)
ifconfig eth0 up

#Perform automatic telnet login to Network Camera
./loginToNetworkCamera.exp

#Transfer file from Network Camera using standard FTP
./ftpFromNetworkCamera.sh

#Temporarily deactivate ethernet interface to Network Camera (eth0)
ifconfig eth0 down

#Temporarily reactivate wireless interface (wlan0)
rmmod zd_b
./setupAdhocNetwork.sh

#This is the end of the script file
```

Listing 3.4 – Bash script automating operation of Axis 210 network camera.

The software component that read signals from the motion sensor interfaced to two pins of the Digital I/O interface on its respective TS-7300 SBC was implemented in the C [134] programming language. C possesses the capability to communicate with hardware on a low level, while being written in the syntax of a high-level language that is easier to understand, maintain and modify. In addition, system operation anticipated that the software component performing I/O to the motion sensor would remain in a single

location (i.e. on the TS-7300 SBC) for most of its operating life. Listing 3.5 shows a section of the source code. Please refer to the Appendices for a full listing.

```
startingLocationForPortBDataRegister = mmap(0, getpagesize(),
PROT_READ|PROT_WRITE, MAP_SHARED,
fileDescriptorHandlingMemoryLocation, 0x80840000);
PORT_B_DATA_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x04);
PORT_B_DATA_DIRECTION_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x14);
PORT_E_DATA_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x20);
PORT_E_DATA_DIRECTION_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x24);
GPIO_PORT_B_DATA_BIT = (unsigned int
*)(startingLocationForPortBDataRegister + 0xC4);
*PORT_B_DATA_DIRECTION_REGISTER = 0xf0;
*PORT_E_DATA_DIRECTION_REGISTER = 0xff;
*GPIO_PORT_B_DATA_BIT = 0x01;
stateOfPortBDataRegister = *PORT_B_DATA_REGISTER;
while(stateOfPortBDataRegister & 0x01){
  stateOfPortBDataRegister = *PORT_B_DATA_REGISTER;
}
WriteDataToFile(MOTION_SENSED);

for (counterToBlinkLED = 0; counterToBlinkLED < 5; counterToBlinkLED++){
  *PORT_E_DATA_REGISTER = 0xff;
   sleep(1);
   *PORT_E_DATA_REGISTER = 0x00;
   sleep(1); }
   close(fileDescriptorHandlingMemoryLocation);
  return 0;}
```

Listing 3.5 - Segment of C program reading signals from motion sensor.

One pin functioned as a ‘ground’ or reference signal, the other as a ‘live’ or ‘data’ signal. The comparison of the signal level on the data pin to that on the reference pin would be used to determine whether motion had been sensed or not. The microprocessor controlled these pins, and read the signal levels on the same into two data registers dedicated solely for this purpose. The C program reading the data off the

two pins first had to specify the direction of data flow on the same. This was necessary as the pins could function as output, to send data to the outside world, or as input, reading data in from some interfaced device.

The above segment performed proper configuration of the data registers in the EP9302 ARM9 processor of the TS-7300 SBC. Once this was completed, it read the contents of the data register monitoring the 'live' signal line and designated it as the 'initial' state of the incoming signal from the motion sensor. It proceeded to wait until the detection of a change in signal level on the pins interfaced to the motion sensor. When this occurred, it extracted the data stored in the data register and wrote it to a text file stored in the embedded operating system. As a visual indication to the user that the program was running correctly, it blinked two indicator LEDs on the TS-7300 board three times in succession, upon detecting a change in signal level. Scripts running on the TS-7300 SBC automatically transferred this file to the associated Holon Board on which resided the Device Agent for the motion sensor. Figure 3.17 illustrates these steps in a UML activity diagram.

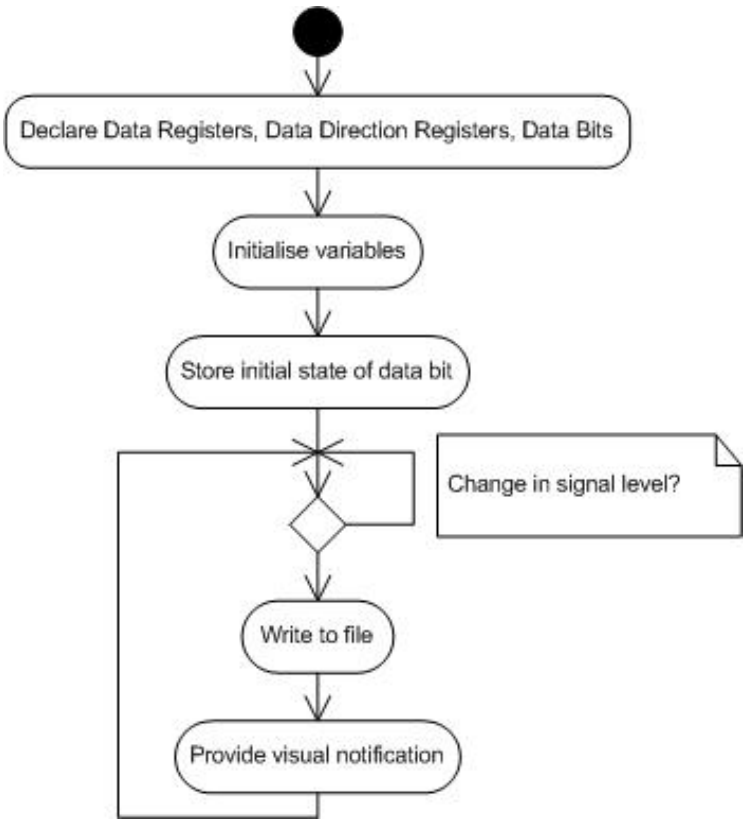

Figure 3.17: UML activity diagram of embedded C program.

3.9 Holonic Unit Integration

Aside from the implementation of the individual software and hardware components, was the integration process where they were all brought together to form their individual HUs and the complete DIS security system. Figure 3.18 shows the combined hardware and software component listing of a typical HU on the DIS. A discussion of how the integrated components worked together follows.

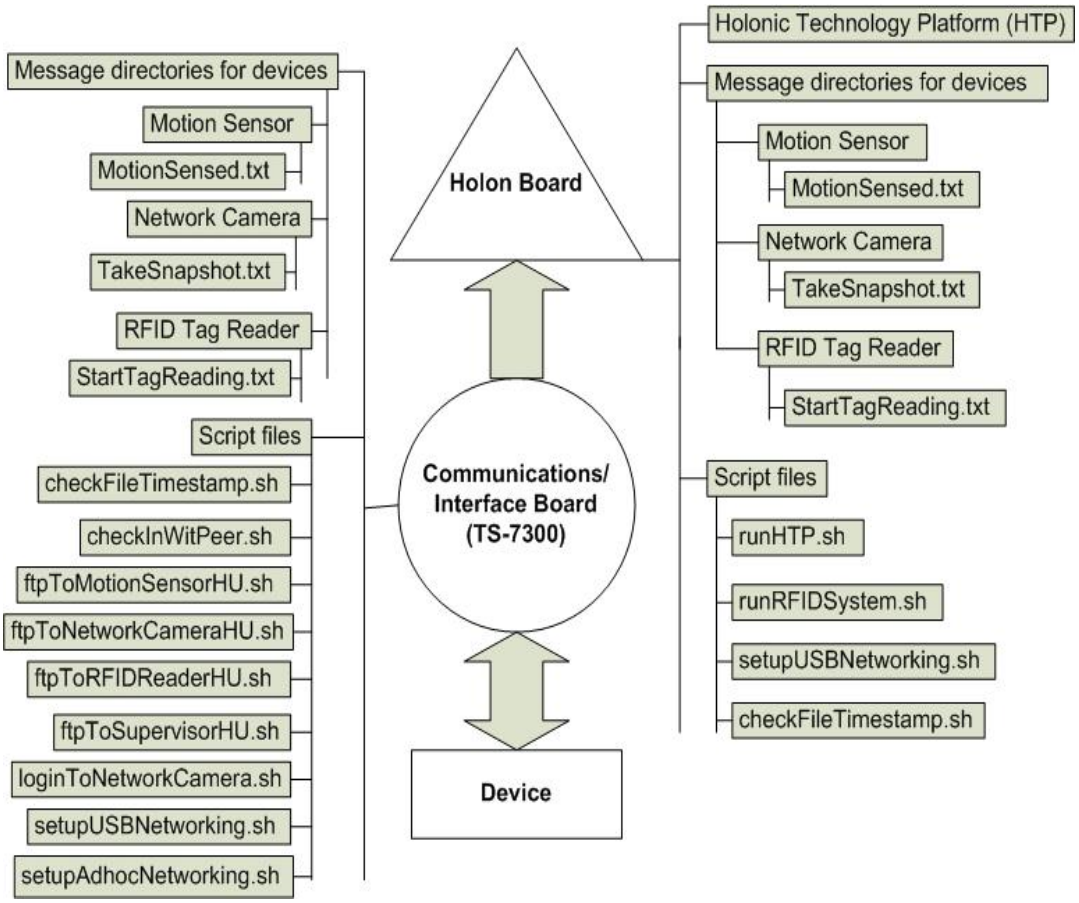

Figure 3.18: Combined component listing of Holonic Unit.

The communication process between a devices and its associated Device Agent on the Holon Board occurred via messages placed in the appropriate directories on the Communications/Interface Board. When a device had generated data from an operation (sensing motion, taking a snapshot, reading an RFID tag), it placed a file containing the data into the appropriate directory. A script running on the Communications/Interface Board registered the presence of a new message, and proceeded to send it to the corresponding message directory for that device on the Holon Board. A similar script on the Holon Board would notify the Device Agent, which proceeded to take action according to the rules specified for it in its configuration file. This may have involved the generation of a message meant for a Device Agent on another HU. In this case, the communication process worked in reverse. A script on the Communications/Interface Board detected the presence of a new message from the Holon Board, and invoked the appropriate script for a file transfer over the wireless network to the Communications/Interface Board on the destination HU. As an example, Figure 3.19

shows the process involved when the Network Camera receives a message from the Motion Sensor HU, instructing it to take a snapshot.

A similar process was involved when holons and agents on different HUs needed to communicate with each other. By generating the appropriate message files and placing them in the proper message directories, script files on the Communications/Interface Board watching for new outgoing messages could transfer them to the destination HUs, and similarly relay incoming messages.

Although this process worked for the implemented system, it was not the ideal solution. It suffered from drawbacks, due to constraints of both hardware and software. The coming chapter undertakes detailed discussion and consideration of this limitation, among others.

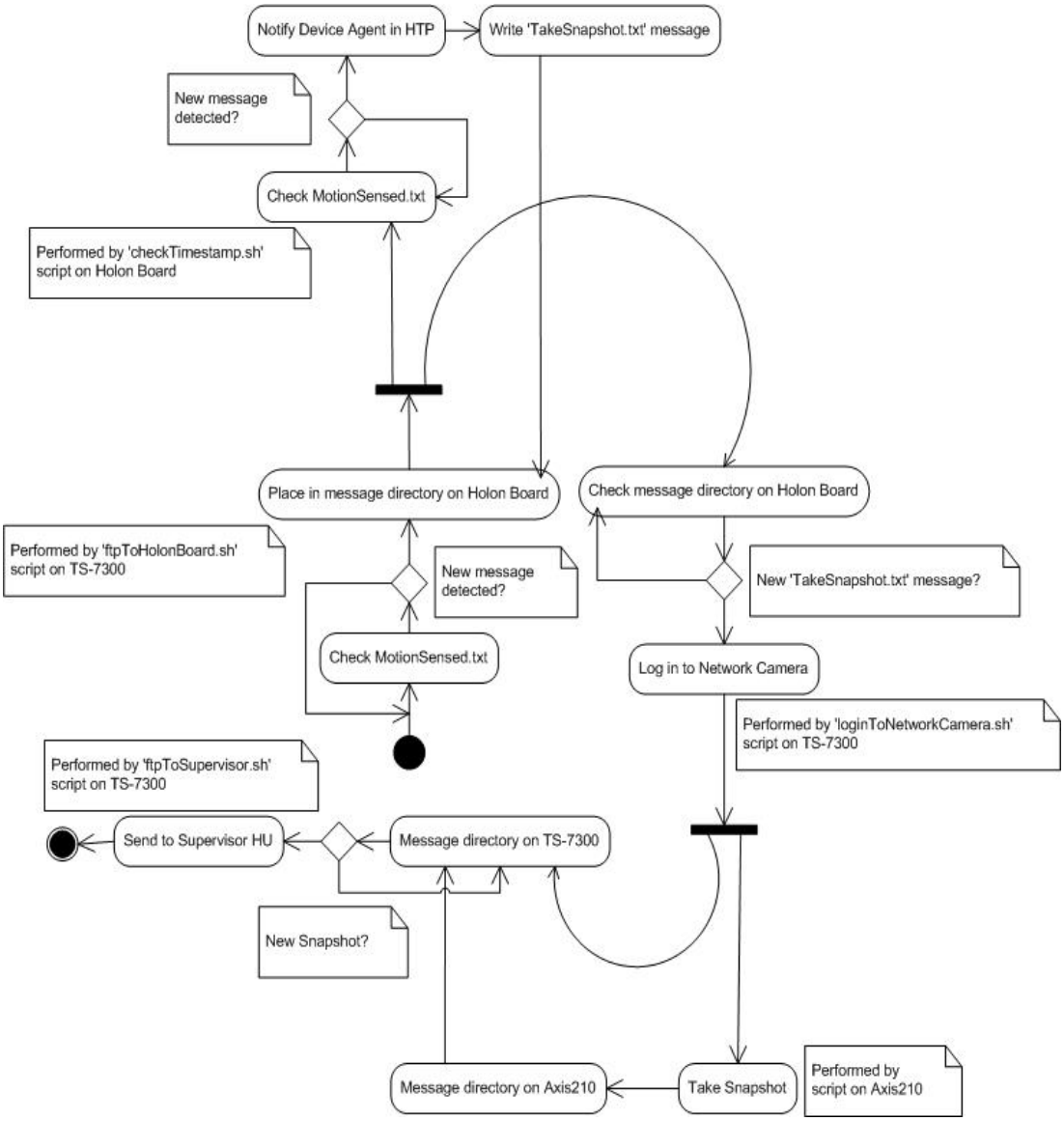

Figure 3.19: Snapshot operation on Network Camera HU.

3.10 Summary of Implementation Process

This chapter described the stages of implementation of the DIS security system designed in Chapter 2. Discussion pertained to relevant details regarding implementation of both hardware and software components. In each case, there was consideration of design factors that influenced implementation strategy. There was examination of various options available, as well as the relative merits and demerits of each. The option that was finally implemented was given attention, with due attention to the reason for selecting that over the others available.

Obstacles experienced during the implementation process necessitated the selection of certain solutions for implementation. Compromise was required in some aspects of translation of architecture specifications into working components, due to various limitations or compatibility issues with hardware and software. The next chapter delineates and discusses this, through an analysis and critique of the accomplishments of the system implementation process.

CHAPTER 4: ANALYSIS

4.1 Introduction

Translation of the initial high-level system specifications into working software and hardware components faced challenges at different stages of the implementation process. Constraints and difficulties posed occurred due to limitations of software tools and capabilities; hardware configuration and performance; design flaws and unforeseen situations, among others. Solutions were required when encountering obstacles, and required different strategies in addressing them.

In some cases, design specifications of components required reconsideration and extensive modification work before proceeding forward. Other challenges called for selection of different software tools from those initially decided upon, due to limitations in capability, ease of use or other factors. Some design flaws revealed in hardware and software required construction of a completely new component for optimum functioning. Such solutions were unfeasible due to constraints of time and other factors; alternate means were necessary to achieve adequate operation within reasonable performance limits.

This chapter will give due consideration to the resulting system, observations made during the course of system implementation, and lessons learned from both.

4.2 Implementation Difficulties

4.2.1 Hardware

Obstacles in terms of system hardware were most significant with respect to the Holon Board and Communications/Interface Board (TS-7300). Although selected to fulfil appropriate roles within the HU, their individual limitations raised challenges in operating with each other.

One of the observations in this regard related to the hardware interfaces on both components. The Holon Board was quite restricted in the number of interfaces available; it possessed only one serial (RS-232) and one USB interface. In addition, the USB

interface could only operate in the configuration of a ‘client’; it was unable to initiate communication with attached devices. This imposed a further constraint in terms of the number and nature of solutions available for communication. Since the Holon Board accommodated intelligence and decision-making capabilities of the HU, it became a drawback when it could not initiate communication with the TS-7300 after an agent or holon had taken a decision or created a message that required sending. In contrast, the TS-7300 possessed numerous hardware interfaces, including two USB interfaces – one operating as a ‘host’, the other as a ‘client’. The strategy undertaken as a working solution interfaced the USB interface on the Holon Board and the host USB interface on the TS-7300. A proper implementation would require mandatory bidirectional communication capability between the two devices. However, if using USB for communication, this would require a host and client interface on each board. The ideal would be a single USB interface that can function as host or client as required, or the use of a different communications interface (such as Ethernet), with appropriate inclusion into future versions of the Holon Board.

Hardware resources available on the TS-7300 [134] became a constraining factor in terms of the software components residing on it. Its relatively small memory capacity (32 MB of SDRAM) raised concern regarding the impact on its performance by memory intensive software components, such as those written in Java. Such restrictions limited the options of programming languages for development and implementation on the TS-7300. Software components were implemented using languages such as C, C++ and BASH, and were less sophisticated in their functionality in comparison to those executed on the Holon Board (such as the HTP platform with holons and agents).

The software driver of the Wi-Fi USB adapter used on the TS-7300 exhibited limitations when operating in ad-hoc mode. One of these limitations was manifest during start up of the system, as HUs attempted to form the wireless network for the first time and form links with other HUs in their vicinity. A minimum of one HU (such as the Supervisor) needed to be operational and broadcasting the right identification number of the DIS WiFi network, prior to start up of other HUs. Deviation from this procedure, such as independent and simultaneous start up of HUs, resulted in the self-assignment of each with different Wi-Fi network identification numbers. The outcome was the formation of essentially two or more different Wi-Fi networks, and the inability of HUs to locate each other. This represents an aspect of centralisation in the Wi-Fi communications

architecture. Similarly, temporary disabling of the wireless interface during operation, followed by reactivation created problems; a significant amount of time was required for the device to re-acquire signals from other HUs and join the network. As an example, this became a particular obstacle in the operation of the Network Camera HU. Two network interfaces on the associated TS-7300 were in operation – the wireless interface used by the Wi-Fi adapter, and the Ethernet interface used by the Axis 210. Temporary disabling of the wireless interface occurred when required to login to the Axis210 and take a snapshot, in order for the script automating the telnet utility to use the Ethernet interface. After retrieving the snapshot image, the Ethernet interface was disabled and the wireless reactivated, in order to transfer the snapshot to the Supervisor HU. However, this operation took a significant amount of time, as the wireless device driver attempted to find the signal from the Supervisor HU and register with it in turn.

Flaws in hardware design discovered on the Holon Board hindered proper execution of software components, and required addressing. For example, initial attempts to execute HTP using JamVM in the Angstrom environment failed on all three devices. Detailed examination of the root cause revealed the omission of providing an on-board hardware clock during design and fabrication. HTP in its standard operation checked the system time and made use of it in some aspects of its operation. A proper implementation of the system would have required re-design and fabrication of new Holon Boards incorporating on-board clocks. As this was unfeasible, appropriate modifications were necessary to the HTP source code to remove the usage of system time in any of its operations. The observations from this lesson will influence proper implementation procedures of a DIS in future. Careful checking regarding various requirements of the software platform for its proper operation, against the functionality supplied by the hardware that it must reside on, will help to avoid such difficulties in future.

Flaws introduced during the fabrication process of some Holon Boards presented another difficulty during implementation. Software executed on these boards sometimes functioned erratically, or not at all. Investigation revealed fluctuations of signals due to imperfections in wiring contacts, or shorting of the same. These required repair work; Figure 4.1 documents an example.

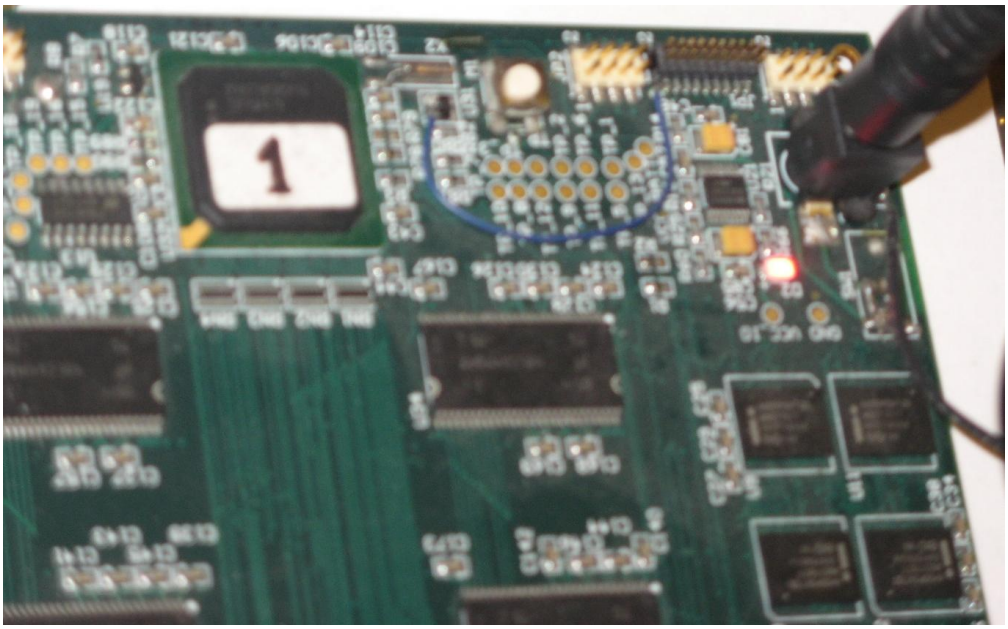

Figure 4.1: Repair work required on a Holon Board.

4.2.2 Software

Implementing software components faced similar challenges and difficulties. Some of these were due to the tools available for development and implementation. Other problems arose because of the architecture of the DIS system itself.

Frequent problems arose with developing and implementing software for the Holon Board A significant obstacle was the lack of a fully featured operating system, and associated tools that would assist in development. A contributing factor to this drawback was the dated nature of the board itself. Designed and fabricated with the hardware technology available four years ago, its specifications did not accommodate for the rapid development and sophistication of embedded Linux systems since its manufacture. Hence, compromise was necessary due to hardware limitations, trading operating system functionality for compactness, speed of loading and execution. Among the major omissions from the toolset were on-board compilers, particularly for programs written in Java and C. Other necessary omissions of tools and utilities included those for remote login (e.g. telnet); searching and locating files; updating software with new versions and native Java libraries for the serial port, among others. The result was an extremely limited development environment, requiring much improvisation in order to overcome the challenges faced.

The inability to update the operating system with ease meant that it remained static throughout the development and implementation process. The difficulty involved and consumption of time outweighed the benefit gained by addition of helpful new tools or functionality. The lack of an on-board compiler required editing and compiling of Java and C programs elsewhere, such as on a development workstation or on the TS-7300, before moving them onto the Holon Board. This resulted in wasted development time, particularly when errors required editing, recompiling and transfer of the modified code onto the board.

A major obstacle was the lack of native libraries for Java that supported serial port communication. These were crucial for the proper functioning of the java code written for the RFID subsystem. A solution was required that did not require developing these components from scratch. This necessitated the use of the native library and supporting Java API provided by the RXTX [135] project. Developed as an alternative to the proprietary CommAPI [136] available from Sun Microsystems, RXTX is an OSS solution supporting serial and parallel communication for the Java Development Kit (JDK). Although RXTX is an extremely useful tool for development involving communication over serial ports, there were numerous issues related to usage on the Holon Board. Problems arose due to improper configuration and malfunction during attempted operation. Many of these problems were due to limited availability and improper organisation of documentation on the usage of RXTX with JamVM, on operating systems such as Angstrom with a very small user base. In addition, although both JamVM and RXTX were ported to embedded platforms using the ARM processor architecture, there was very limited documentation of the issues encountered in doing so. Thus, understanding, solving and documenting the numerous problems encountered in getting RXTX to operate in the development environment occupied significant amounts of development time.

Similarly, issues arose when attempting to execute DIS components developed in Java using JamVM. Some were due to configuration problems related to the operating system environment. Others arose due to compatibility issues with some of the Java library components used in HTP or the RFID subsystem. As with RXTX, extremely limited availability and improper organisation of documentation became the root cause of many difficulties experienced when using JamVM. Necessary modifications to the HTP

and RFID subsystem source code in order for them to function properly required significant time and effort.

Although HTP is a demonstration of a how to construct a viable framework for developing a DIS, shortcomings in its architecture presented difficulties that needed circumventing. The current implementation of HTP presents a drawback when modification is required to the HLS that forms its core. Development and addition of a new Agent to the core architecture necessitates addition of source code to the 'HolonicTechnologyPlatformMaster' file in the Executable Package. An Agent package contains multiple files, each of which requires registration with this master file for proper functioning of the Agent within the HLS. Modifications to this file become increasingly cumbersome with new additions to the core. This feature presents a bottleneck to development, and requires addressing. In addition, HTP did not contain functionality that allowed it to interact with an operating system through the command line of a shell program, such as the Bash shell used in GNU/Linux. The implementation strategy compulsorily required this capability. After taking a decision specified by its rule base, there needed to be a means for the Device Agent to take action. Modification was required to the HTP architecture, allowing it to use the shell to execute scripts automating various courses of action. Another drawback lay in the lack of an automated ability to probe its operating environment and configure or adapt itself for proper operation within the same. Finally, HTP also lacked simplicity in the process that automated its initialisation and activation, requiring the writing of a correspondingly detailed script for this purpose. This was also partly due to its implementation in Java, and its corresponding dependence upon proper specification and configuration of JVM parameters, such as those required by JamVM.

4.3 System Performance and Limitations

Simple tests conducted on the working DIS aimed to obtain some idea of basic performance capabilities. The tests involved measuring the time taken to perform simple system functions, such as transmission of a small message of a size of 3 bytes from one node to another. The results are summarised in the table below.

Table 4.1: Results of timing tests conducted on DIS.

Timed tests		
Test Conducted	**Time taken in unoptimised system (seconds)**	**Time taken in optimised system (seconds)**
Single-hop FTP to Supervisor	< 1	< 1
Two-hop FTP to Supervisor	91	4
Three-hop FTP to Supervisor	170	7
Login to Network Camera and take snapshot	54	54

The results of the tests provide some basic insights. A direct transmission of a simple message with minimal payload from one of the HUs associated with a device to the Supervisor was quite fast. However, in an unoptimised DIS, the various FTP operations that transmitted a single message over multiple Holonic Units to the Supervisor required a significant amount of time. For example, a transmission of the message 'Take Snapshot' from the Motion Sensor HU to the Network Camera HU, followed by the transmission of the message from the Network Camera HU to the Supervisor was an instance of a two-hop FTP process. The major impeding factor was the time taken for the Motion Sensor HU to obtain a login prompt from the Network Camera HU (an average of 90 seconds) while performing an automated FTP process to transmit the message file. If the time taken for the Communications/Interface Board to perform an automated snapshot-taking procedure (54 seconds) is included, it requires approximately 145 seconds for completion of the procedure. With the inclusion of an extra node in the messaging chain covering three HUs before reaching the Supervisor, a

time of almost 170 seconds was required for a message to reach the Supervisor. This is an unacceptable response time for a real-world DIS in a security system application. Optimisation of the system primarily involved configuration of the FTP services supplied with the Debian Linux on the TS-7300, with the objective of reducing the waiting time for one HU to retrieve a login prompt from another. Proper configuration resulted in a drop by an order of magnitude in the values generated from the timing tests, as seen in the second column of Table 4.1.

A drawback to the interpretation of these results was the lack of a similar security system using a centralised architecture, as a reference for comparison with similar tests. Similarly, the small scale of the system obviated a study of performance with increase in the population of participating HUs and comparison with a corresponding centralised architecture. Future study will investigate system performance of an improved and optimised system, under situations involving high bandwidth usage and transmission large messages. Furthermore, DIS architectures rely extensively on reliable communications and messaging capabilities that facilitate negotiation among entities. In-depth study, to investigate the effects of degraded communications capabilities upon the operation of HUs towards the global system objective was not undertaken. However, the possibility exists that with further development effort and optimisation, a high-performance DIS for a security system application, with real-time response times can be achieved.

CHAPTER 5: CONCLUSION

5.1 Introduction

This thesis has attempted to demonstrate the viability of a Distributed Intelligent System as an alternative framework to centralised architectures for control of devices. In the course of this endeavour, we have experienced some aspects of the systems development cycle.

The development cycle in systems engineering begins with an understanding of an existing problem related to a system, and the formulation of an idea for a viable solution to the same. A planning process follows, involving consideration of details regarding the proposed solution, and its implementation. Subsequent implementation of the solution follows, covering all aspects that translate the abstract system specifications into a working system, in terms of hardware, software or both. Analysis performed upon the results of implementation examines the outcome and lessons learned in the process, which may be valuable for future undertakings. These become the basis for engineering a better system in future, than the one currently designed and implemented.

This chapter concludes the process by considering the outcome of the undertaking, and the direction of future work in building upon the progress made thus far.

5.2 DIS Development Cycle

The concept of a DIS emerged from a consideration of the history of systems development and engineering, focused on control of devices. Examination of increasing reliance upon the centralised framework that progressively became the norm revealed its drawbacks, particularly in the context of increasing scale in both the number of devices comprising the system, as well as their inherent complexity. An inquiry into attempts at solving these significant problems, through various forms of decentralised or distributed systems, revealed their shortcomings in addressing the root cause of the problem. The end of the inquiry process showed that an alternative solution based on a different paradigm was necessary, in order to meet the requirements and challenges of

large-scale systems development for the future. Attention was drawn to the significant success demonstrated in this regard by the recent emergence of Internet applications based upon the peer-to-peer (P2P) framework. Examination of the inherent benefits of their underlying architecture suggested the viability of using a P2P system as a starting point for constructing an alternative framework for device control applications. Recognition was given to the modifications and added capabilities necessary to the existing P2P architecture, in order to achieve this objective. A simple DIS security system would demonstrate how the proposed architecture could be used in a real-world application.

Detailed consideration was given to the nature of the DIS architecture, based upon the formulation of the initial concept. Evaluation of the fundamental requirements that the proposed alternative would have to meet, led to a specification of the entities that would comprise the architecture. Particular emphasis on architectural specifications related to the concept of a holon and associated agents residing on a device, each entrusted with user-specified global and local objectives respectively. Consideration of existing platforms using agent-based architectures dealt with their drawbacks that made them unsuitable for the foundation of the DIS. A potential candidate identified was the Holonic Technology Platform (HTP), expressly constructed as a framework for developing systems operating in a P2P environment. Further delineation through the design process resulted in the introduction of the holon-agent model and associated support structures necessary for proper functioning of the same. The general application to control of devices and specific application in the DIS security system, led to formulation of the architecture and design specifications of a generic Device Agent based on the Knowledge, Rules and Behaviour (KRB) model. With an understanding of the structure of entities in the system in mind, architecture and design considerations shifted to the problem of bringing order to the entire system, for the purposes of control. Inherent value was noted in tools from Formal Concept Analysis (specifically partial ordering and lattices), leading to invocation of the same as viable mechanisms. A depiction applying the partial ordering methodology within the DIS concluded the architectural and design process.

Implementation of the DIS architecture followed, translating the specifications into working hardware and software components of the security system. Hardware selection related to components hosting the Holon and associated agents, and

equipment such as the motion sensor, network camera and RFID tag reader. Explanation provided dealt with specifications of the components, the rationale behind their selection, and interconnection of hardware modules to form the individual entities within the DIS. Similar attention covered rationale for selection, specification and implementation of software components implementing the intelligence aspect of the DIS. This specifically related to HTP, the Device Agent and supporting components facilitating operations within a Holonic Unit.

Following the implementation process, assessment of the results provided an initial understanding of the difficulties and challenges involved in implementing a proposed DIS, using existing tools for development and implementation. Consideration of difficulties covered problems experienced in both the hardware and software domains. Numerical results obtained demonstrated some of the aspects of system operation, providing a basis for further insight into system performance and limitations.

The final stage involves contributing to improvements in future efforts at engineering DIS, and follows in the next section.

5.3 Improvements and Future Work

The basis for engineering a better DIS draws upon the observations of the implementation process, described in Section 4.2 of Chapter 4. This may provide scope for future work, using the lessons learnt in this endeavour and past efforts in DIS research to advance the state-of-the-art in the field.

5.3.1 Hardware

Efforts at improving DIS hardware capabilities will require re-consideration of some hardware design paradigms. Applying the principle of loose coupling in multiagent systems mentioned in Section 2.3, would contribute to realisation of ‘smart’ hardware.

One such domain where application is required relates to hardware interfaces such as USB, which currently operate only as host or client. It would be extremely advantageous to develop a USB interface incorporating the intelligence required to function as both, or switch between one and the other as required. Bidirectional communications and control capability over USB is a valuable feature. Although the counterargument might suggest using a different interface such as Ethernet to achieve

such capability, it is also less compact and involves more electronics than USB, which is currently widely used in very compact forms.

Another area for improvement pertains to flexibility of on-board storage. The dated design of the Holon Board makes for difficulty in maintenance and modification - the surface-mounted NOR and NAND flash memory chips cannot be easily removed and replaced when necessary. The hardware capabilities are quite restricted, consequently constraining the degree of sophistication of an operating software environment. The TS-7300 provides inspiration for future design improvements, with its two on-board slots for SD-Cards hosting the operating system and development tools. This allows variable and expandable options in terms of on-board storage and support for more sophisticated software capabilities. During the course of the project, a single slot on the TS-7300 accommodated an SD-Card of 2 GB capacity. Continuous lowering in cost of such storage technology makes them viable for standard inclusion in the design of future DIS hardware.

Chipsets and software drivers developed for wireless hardware require incorporation of better intelligence capability, in order to function effectively in a peer-to-peer environment. As mentioned in Section 4.2.1, the Wi-Fi chipset and associated software driver operated in ‘ad-hoc’ mode for the purposes of the project, but this was not true peer-to-peer capability. For instance, formation of the wireless DIS network required at least one HU to be already operating in ad-hoc mode, for the others to be able to find and join it. A direction for future development and optimisation is the capability to intelligently network with other devices and manage the wireless infrastructure without this conditional requirement.

The proliferation of GNU/Linux, and its subsequent advancement in sophistication in the embedded space, provides opportunities for increasing the sophistication of devices and their integration of an on-board DIS system. The Axis 210 network camera used in the DIS security system application is an example of how future attempts towards ease of integration may progress through the provision of on-board embedded operating systems such as Angstrom, Debian, TS-Linux and others. The ability to accommodate a full-featured Debian distribution on a SD Card as small as 512 MB proved to be extremely valuable during this project; advancements in hardware and compact storage devices present future opportunities for incorporation into a DIS through the ability to store and load full-featured operating systems on them.

5.3.2 Software

Constructing a better DIS in future requires improvements in engineering of software components on many different fronts. Some of the areas of difficulty experienced during the course of this project, and requiring significant effort at improvement are the operating system environment; development toolset and utilities; architecture of intelligence platforms such as HTP and documentation.

Operating systems based on GNU/Linux for the embedded space have rapidly increased in sophistication in recent years. However, improvements will facilitate ease of support and use by a DIS. For example, difficulties encountered during this project suggest incorporation of better intelligence in automatically detecting and configuring on-board hardware. Similarly, the intelligence to probe a DIS for any required libraries or supporting platforms (such as a JVM), determine the optimum one suited for the operating environment, retrieve and install it would be advantageous. Such capability opens opportunities for incorporation into future DIS systems.

Development tools and utilities require significant effort directed towards increased sophistication. Despite attempts at advancement, they are still relatively unsophisticated with respect to advancements in hardware. The limitations of the toolset with respect to availability, ease of configuration, and ease of use consumed much development time during this project. As an example, future effort to develop intelligence in tools for automatic probing of and configuration to the environment will significantly shift the balance in favour of productivity. This applies to development of operating systems, programming languages, and supporting tools.

In terms of operating systems, cognizance of the need for greater architecture independence by the OSS community has influenced the development of GNU/Linux, particularly with respect to its kernel. Nonetheless, configuring, compiling and installing a GNU/Linux-based operating system with supporting packages for an embedded target is a tedious task. Many opportunities exist for improvement of development environments. For example, sophisticated capabilities such as easy configuration and swapping of kernels for different targets, will greatly contribute towards engineering better DIS systems in future.

Similarly, efforts are also necessary to better the independence of programming languages from underlying hardware or operating systems. For example, although Java

was supposed to be independent of the underlying operating system, this is not the case. Much room for improvement remains in attaining this goal, for both proprietary offerings and the alternatives from the OSS community. For example, a useful feature for JVMs such as JamVM might be an intelligence component that retrieves both the DIS and operating system specifications, using the attained information to modify its own configuration for proper operation. Many of the concepts employed in the DIS architecture, such as loose coupling and the holon-agent model, are viable for application on many fronts in the improvement of both operating systems and development tools. Using the previous discussion of issues with JVMs as an example, an agent-based architecture for JamVM might assist in intelligent configuration to the requirements of an operating system or DIS.

Thus, there is much opportunity available on the hardware and software fronts that will help in engineering a better DIS for future applications.

5.4 Possible Specifications of a Future HU

Although the realisation of the previously discussed future improvements may require significant time and development effort, a suggestion of the specifications of the next HU prototype is possible.

The current hardware made use of a mixture of Commercial Off-The-Shelf (COTS) and custom-built embedded computers, employing the ARM microprocessor architecture. A possible venture for the next prototype in this regard might be employing an embedded computer using the Intel Atom [137] microprocessor. Used in the current generation of ultra-portable notebook computers, also known as ‘netbooks’, an additional incentive is the existence of fully featured and commercial grade GNU/Linux distributions such as Ubuntu, which have been specifically compiled for this platform. This should allow easier development, with the existence of a wider user base and the increased availability of technical support and documentation.

If necessary, fabrication of custom-built embedded hardware employing an Atom™ processor complemented with a necessary amount of on-board memory and I/O is being investigated in the iDEA Laboratory. If a custom-built solution is required, advantageous features of the current individual embedded boards will be selected and integrated into a single platform. Given the current state-of-the-art in miniaturisation of embedded devices, it is viable to design a hardware package with the physical

dimensions of a small mobile phone. This will result in the advantageous reduction of the current physical implementation of a two-element controller to a single-element device as the basis of the hardware package.

5.5 Final Evaluation and Summary

This thesis set out to show the viability of an alternative framework for device control in contrast to the centralised frameworks that systems today rely upon. In this respect, it aimed to demonstrate how all participating entities in the system might work together within a peer-to-peer system architecture, having control, communications, intelligence and decision-making capability endowed upon all of them. The resulting proposal of a Distributed Intelligence System employed a model dividing the system objectives into global and local responsibilities, handled by holons and agents respectively. Furthermore, application of the proposed DIS to a simple security system went through a systems development cycle process of design, development, implementation and analysis. Although the resulting system was less than ideal and modifications required to the initial goals, it contributed towards the primary objective of the undertaking. The exercise also instilled valuable experience and important lessons related to systems engineering, on the fronts of hardware and software along the way. It is desired that the work contained herein will contribute to future efforts in the field of DIS research.

A paradigm shift is required, if systems engineering is to find a viable and comprehensive means of addressing the many problems faced today, brought about by relying upon centralised frameworks for the construction of large-scale systems. The alternative proposed by the DIS emerged from the confluence of many interacting factors, including the emergence of P2P networks and their applications; holon-agent models for system architectures and widespread adoption and advancement of OSS such as GNU/Linux in embedded systems. The following quote is perhaps a fitting manner in which to end this thesis:

"*The formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill. To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science.*"

- A. Einstein and L. Infeld, 1938

APPENDICES

Appendix 1: Partial Ordering Using Lattices

Building on the application of the security system considered thus far, the isomorphism of a lattice applies to a set of three nodes - the motion sensor, network camera and RFID tag reader HUs respectively. Consider Figure A1.1, which illustrates the isomorphism of the DIS security system as a Hasse diagram. All further explanation of organization within the system references this representation. Let 'D' be the entire set of devices in the DIS. system. The elements that form the set are represented in set notation as {MS, NC, RS}, representing the Motion Sensor, Network Camera and RFID Scanner, respectively. The fourth unconnected element in the diagram {SU} is representative of the system administrator, also known as the Supervisor.

In this context, reflexivity is the property where for all the elements MS, NC and RS $\in$D, any selected element must be identical to itself, e.g. MS ≤ MS. Antisymmetry is the property where in a comparison of two apparently distinct elements using both the ≤ and ≥ operators, if both conditions evaluate as being true, it implies that the elements must in fact be identical. Antisymmetry enforces the actual ordering of elements and prevents the possibility of two elements ordered by a binary relationship being simultaneously greater and less than each other, e.g. MS ≤ NC and MS ≥ NC at the same time. Finally, the property of transitivity means that deductions can be made based on various binary relations among elements created by the partial ordering. For example, if in order of security clearance level MS ≤ RS and RS ≤ NC, it can be logically deduced that MS ≤ NC.

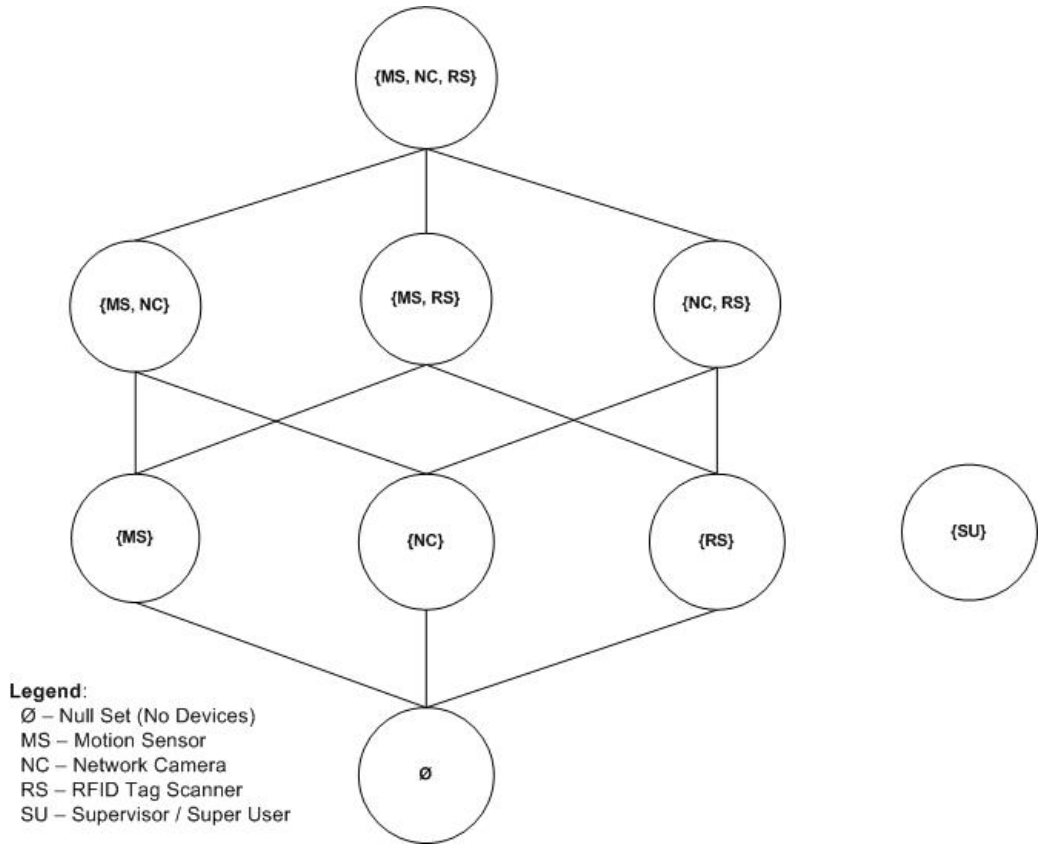

Figure A1.1. Hasse diagram for a DISC security system with 3 peers.

With these properties in mind, the Hasse diagram of the DIS security system shows that the first level of the null (empty) set contains no devices. The single elements of the Motion Sensor (MS), Network Camera (NC) and RFID Tag Reader (RS) are one level higher and disjoint from one another. The null set is a subset of each of these elements, and connected to all of them. The third level represents the various pairings of the single elements. The individual elements of the first level are members of only the set of which they are a subset, e.g. {MS} is a subset of both {MS, NC} and {MS, RS}. The final and fourth level of the lattice is representative of the complete set of devices.

In like manner, a Hasse diagram can be constructed for an isomorphism of any number of nodes of a finite poset of devices in a DIS. Consider the isomorphism of a poset D' = {MS, NC, RS and SU}. Figure 17 shows the Hasse diagram of D'. As Figure 17 illustrates, the structural ordering brought into the architecture of a DIS with four nodes is of a greater degree of complexity than that of a three-node DIS. Another important conclusion that emerges upon examination is that the isomorphism of D' can

be generated by algebraic structures that operate on the extra element of the Supervisor node and its associated pairings with the nodes of D in Figure A1.2.

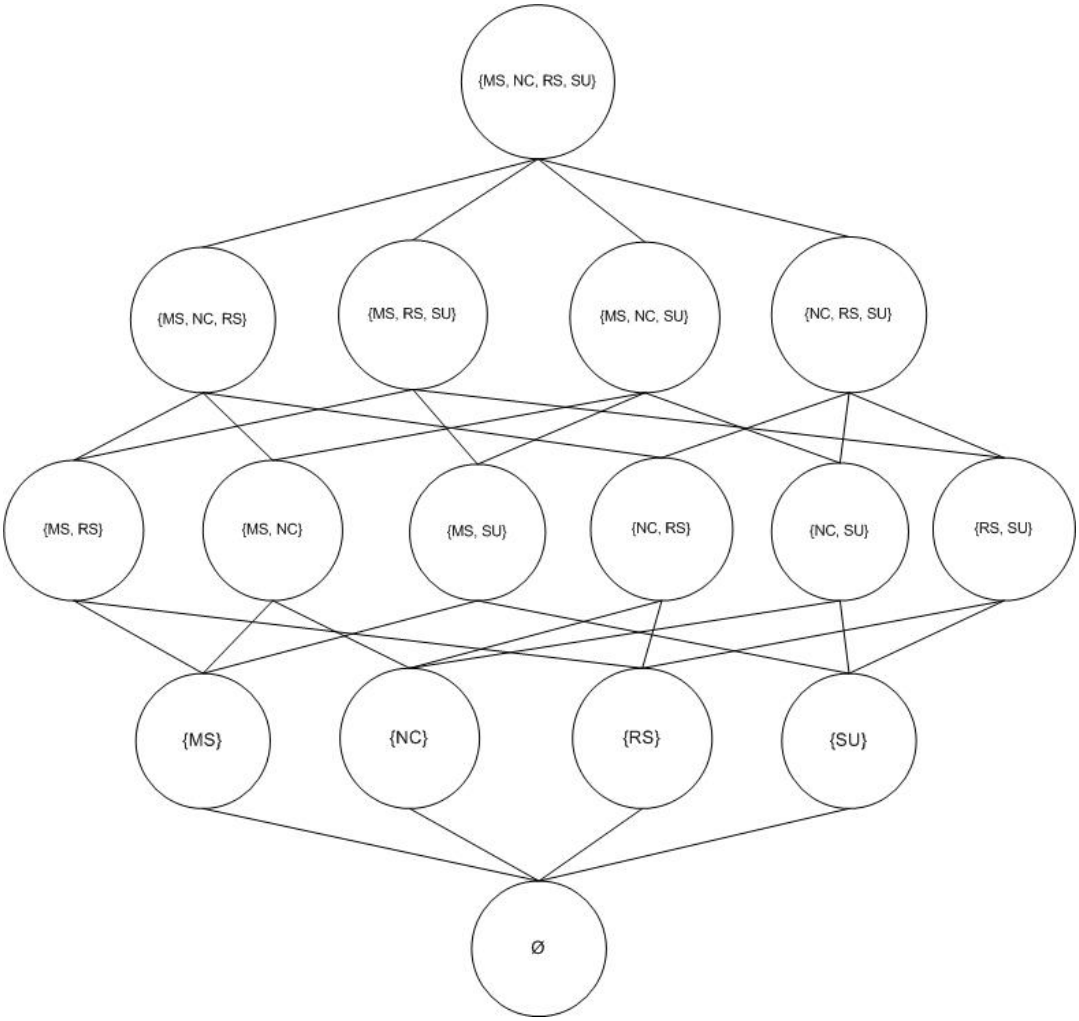

Figure A1.2: Hasse diagram of a poset D' created from 4 elements.

Appendix 2: Technologic Systems TS-7300 SBC

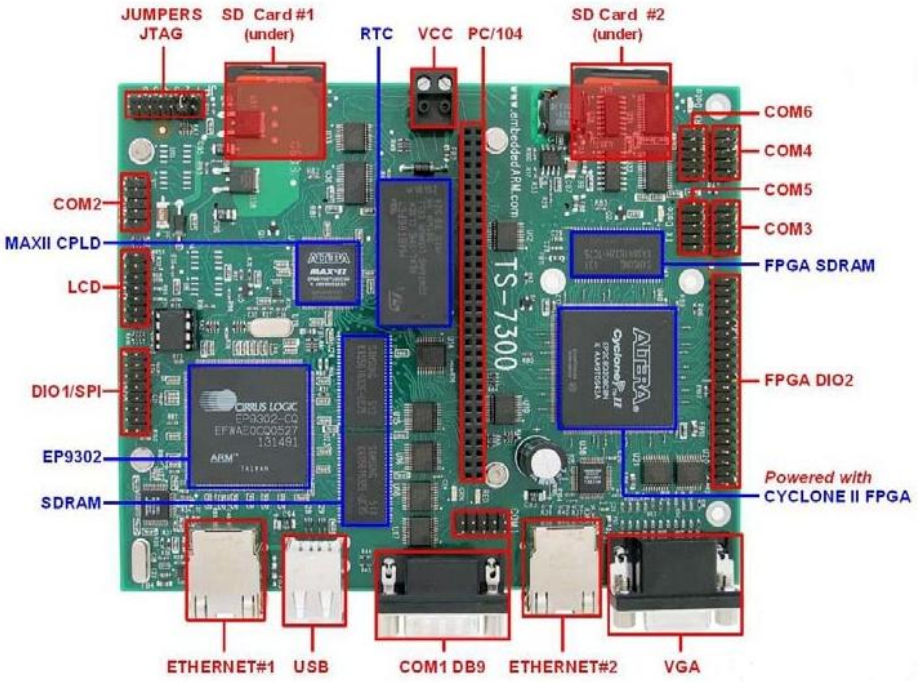

Figure A2.1: Block diagram of TS-7300 SBC and on-board components [134]

The TS-7300 SBC is a product meant for implementation of embedded systems. It contains an on-board Cirrus EP9302 ARM9 [152] processor, complemented by various peripheral devices and interfaces (Ethernet, USB, RS-232, VGA, SD Card, Compact Flash, Analog to Digital Converters, Digital I/O pins, etc.). In addition, the TS-7300 SBC possesses an on-board Cyclone II [138] Field Programmable Gate Array (FPGA) manufactured by Altera Corporation. The Cyclone II FPGA is capable of being reprogrammed on-the-fly, with primary use to provide additional peripheral logic such as Digital Signal Processing, Digital I/O and other capabilities as needed. Figure 4 is a diagram of the TS-7300 SBC showing its various components [134].

Appendix 3: Device Agent Configuration File

```
//Main section Deliminators = ::
//Subsection Deliminators = :
----------
<<DEVICE_AGENT.MAIN_SECTION>>::
<SUBSECTION.IDENTITY>:
PARENT_HOLONIC_UNIT=MotionSensorHolonicUnit
DEVICE_AGENT_NAME=MotionSensorDeviceAgent
DEVICE_AGENT_IDENTIFICATION_NUMBER=1
<SUBSECTION.CONNECTIVITY>:
MOTION_SENSOR_HOLONIC_UNIT=192.168.0.50
NETWORK_CAMERA_HOLONIC_UNIT=192.168.0.51
RFID_READER_HOLONIC_UNIT=192.168.0.52
SUPERVISOR_HOLONIC_UNIT=192.168.0.102
<SUBSECTION.LOCATION_OF_MESSAGE_FILES>:
PATH_TO_MOTION_SENSED_MESSAGE=/home/sage/workspace/DISC/Messages/M
otionSensed.txt
PATH_TO_TAG_READING_MESSAGE=/home/sage/workspace/DISC/Messages/StartT
agReading.txt
PATH_TO_TAKE_SNAPSHOT_MESSAGE=/home/sage/workspace/DISC/Messages/Ta
keSnapshot.txt
<SUBSECTION.LOCATION_AND_NAMES_OF_SCRIPT_FILES>:
PATH_TO_MAIN_SCRIPT_DIRECTORY=/home/sage/workspace/DISC/Scripts/
SCRIPT_HANDLING_FILE_TRANSFER_TO_MOTION_SENSOR_HOLONIC_UNIT=./ft
pToMotionSensorHolonicUnit.sh
SCRIPT_HANDLING_FILE_TRANSFER_TO_NETWORK_CAMERA_HOLONIC_UNIT=
./ftpToNetworkCameraHolonicUnit.sh
SCRIPT_HANDLING_FILE_TRANSFER_TO_RFID_READER_HOLONIC_UNIT=./ftpTo
RFIDReaderHolonicUnit.sh
SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC_UNIT=./ftpToS
upervisor.sh
<SUBSECTION.OPERATING_SYSTEM_SPECIFICATIONS>:
SYSTEM_COMMAND_FOR_OPERATING_SYSTEM_SHELL=/bin/bash
<SUBSECTION.RULES>:
PRIMARY_INCOMING_MESSAGE_TO_MONITOR=/home/sage/workspace/DISC/Mess
ages/MotionSensed.txt
<SUBSECTION.BEHAVIOUR>:
FIRST_COURSE_OF_ACTION=./ftpToRFIDReaderHolonicUnit.sh
SECOND_COURSE_OF_ACTION=./ftpToSupervisor.sh
```

Appendix 4: RFID Subsystem Configuration File

```
//Main section Deliminators = ::
//Subsection Deliminators = :
----------
<<RFID.MAIN_SECTION>>::
<SUBSECTION.RFID_OPCODES>:
GET_VERSION=FF00031D0C
BOOT_FIRMWARE=FF00041D0B
SET_REGION=FF0197014BBC
SET_TAG_PROTOCOL=FF02930005517D
SET_TX_POWER=FF029209C4489D
SET_ANTENNA_PORT=FF02910101703B
READ_TAGS=022103E8
READ_MULTIPLE_TAGS=022203E8
GET_TAG_ID_BUFFER_ONE_TAG=02290001
CLEAR_TAG_ID_BUFFER=002A
<SUBSECTION.LOCATION_OF_FILE_STORING_TAGS_READ>
PATH_TO_TAG_ID_FILE=/home/sage/workspace/DISC/RFIDReader/TagID.txt
----------
<<USER_INTERFACE.MAIN_SECTION>>::
<SUBSECTION.GLOBAL_VARIABLE_SETTINGS>:
UNSIGNED_READ_BUFFER_LENGTH_IN_BYTES=260
<SUBSECTION.CLASS_INSTANTIATION_SETTINGS>:
MESSAGE_ARRAY_ELEMENT_CONTAINING_LENGTH_FIELD_OF_RESPONSE_ME
SSAGE=1
MESSAGE_ARRAY_ELEMENT_CONTAINING_NUMBER_OF_RFID_TAGS_FOUND=
5
----------
<<SERIAL_PORT_UTILITIES.MAIN_SECTION>>::
<SUBSECTION.PORT_NAME_OPTIONS>:
PORT_NAME_01=/dev/ttyUSB0
PORT_NAME_02=/dev/ttyS1
```

PORT_NAME_03=/dev/ttyS2
PORT_NAME_04=/dev/ttyS3
PORT_NAME_05=/dev/ttyS4
PORT_NAME_06=/dev/ttyS5
PORT_NAME_07=/dev/ttyS6
PORT_NAME_08=/dev/ttyS7
<SUBSECTION.PORT_SPEED_OPTIONS>:
PORT_SPEED_01=9600
PORT_SPEED_02=19200
PORT_SPEED_03=38400
PORT_SPEED_04=57600
PORT_SPEED_05=115200
PORT_SPEED_06=230400
PORT_SPEED_07=460800
<SUBSECTION.PORT_OPENING_SETTINGS>:
SERIAL_PORT_TIMEOUT_VALUE=3000
<SUBSECTION.SERIAL_PORT_PARAMETERS>:
BAUD_RATE=9600
DATABITS_8 =
STOPBITS_1 =
PARITY_NONE =
<SUBSECTION.GET_RESPONSE_METHOD_SETTINGS>:
MESSAGE_HEADER_SIGNATURE_VALUE=-1
MESSAGE_ARRAY_ELEMENT_CONTAINING_HEADER_SIGNATURE=0
OFFSET_VALUE_TO_READ_COMMAND_STATUSWORD_AND_CRC=5
OFFSET_VALUE_TO_DETERMINE_TOTAL_LENGTH_OF_MESSAGE=2
<SUBSECTION.GET_RFID_TAG_METHOD_SETTINGS>:
SIZE_OF_ARRAY_TO_HOLD_TAG_ID=12
OFFSET_VALUE_USED_IN_PROCESSING_MESSAGE_ARRAY_TO_RETRIEVE_TA
G_ID=9
<SUBSECTION.GET_DATE_AND_TIME_METHOD_SETTINGS>:
OFFSET_VALUE_FOR_DETERMINING_YEAR=1900
OFFSET_VALUE_FOR_DETERMINING_MONTH=1

Appendix 5: File Handler Source Code Listing

```
package Package_ToolsAndUtilities;

//Importing the Java I/O package
import java.io.*;
//Importing the String class from the Java Language package
import java.lang.String;

public class PackageToolsAndUtilities_FileHandler
{
  //Constant replacements
  private final static String END_OF_FILE_INDICATOR = null;
  private final static boolean FILE_IS_BEING_READ = true;
  //private final static boolean FILE_IS_NOT_BEING_READ = false;

  //Class variables
  public static String WORKING_PATH_DIRECTORY = "";
  public static int SIZE_OF_INITIALIZATION_FILE_IN_LINES = 0;
  private static FileReader handlerOfTheFile;
  private static BufferedReader managerOfStreamFromFileReader;
  public String[] contentsOfFile;

  public PackageToolsAndUtilities_FileHandler()
  {}//This is the end of the class constructor

  public void OpenFile()
  {
    try
    {
            handlerOfTheFile = new FileReader(WORKING_PATH_DIRECTORY);
            managerOfStreamFromFileReader = new BufferedReader(handlerOfTheFile);
          } //This is the end of the try structure
          //If an error occurs during opening file, process the exception
          catch(IOException ioexceptionWhileOpeningFile)
          {} //This is the end of the catch structure
  } //This is the end of the method definition

  public void ParseFileIntoLines()
  {
    contentsOfFile = new String[SIZE_OF_INITIALIZATION_FILE_IN_LINES];
          String currentLineBeingRead = "";
          int markerOfElementBeingAdded = 0;

          while(FILE_IS_BEING_READ)
          {
     //Open parameters file and start reading
            try
```

```
            {
              currentLineBeingRead = managerOfStreamFromFileReader.readLine();
                    if(currentLineBeingRead == END_OF_FILE_INDICATOR)
                    {
                      break;
                    } //This is the end of the if structure
                    else
                    {
                      //Otherwise, store the current line being read into the array.
                      contentsOfFile[markerOfElementBeingAdded] = currentLineBeingRead;
          markerOfElementBeingAdded++;
                    } //This is the end of the else structure
      } //This is the end of the try structure
            catch(IOException ioExceptionWhileReadingFile)
            {} //This is the end of the catch structure
    } //This is the end of the while structure
  } //This is the end of the method definition

  public void CloseFile()
  {
    try
    {
      //Close file after reading into array
      managerOfStreamFromFileReader.close();
    } //This is the end of the try structure
          catch(IOException ioExceptionWhileAttemptingToCloseFile)
          {}//This is the end of the catch structure
  } //This is the end of the method defintion

  public void DisplayFileContents()
  {
    int sizeOfArrayRepresentingFile = contentsOfFile.length;
          int markerForCurrentElementOfArray = 0;
          String currentElementOfArrayAsString = "";

          for(markerForCurrentElementOfArray = 0; markerForCurrentElementOfArray <
sizeOfArrayRepresentingFile;
          markerForCurrentElementOfArray++)
          {
            currentElementOfArrayAsString                                                    =
contentsOfFile[markerForCurrentElementOfArray];
            if(currentElementOfArrayAsString == END_OF_FILE_INDICATOR)
            {
              break;
            } //This is the end of the if structure
            else
            {
              System.out.println(currentElementOfArrayAsString);
            } //This is the end of the else structure
          } //This is the end of the for loop
  } //This is the end of the method definition
```

```
/*
   //NOTE: This section is to be decommented for testing purposes ONLY.
   //It must be recommented afterwards.
         public static void main(String[] args){
           Handler filehandler = new Handler();
           filehandler.WORKING_PATH_DIRECTORY                                    =
"/home/kevthomas/workspace/RFIDTesting/fileUtility/RFIDPackageOperatingParameters
.txt";
           filehandler.OpenFile();
           filehandler.ParseFileIntoLines();
           filehandler.CloseFile();
           filehandler.DisplayFileContents();
         } //This is the end of method main()
*/
} //This is the end of the class definition
```

Appendix 6: File Parser Source Code Listing

```
package Package_ToolsAndUtilities;

import java.lang.String;

public class PackageToolsAndUtilities_FileParser
{
  //Constant replacements
  private final static String END_OF_FILE_INDICATOR = null;
  //Class variables (Public)
  public static String LOCATION_OF_TARGET_FILE = "";
  public static int SIZE_OF_INITIALIZATION_FILE_IN_LINES = 0;
  public String[] InitializationFileSections;
  public String[] InitializationFileSubsections;
  public String[] InitializationFileParameters;
  //Class variables (Private)
  private static String ignoreLinesBeginningWithOption01 = "";
  private static String ignoreLinesBeginningWithOption02 = "";
  private static String ignoreLinesBeginningWithOption03 = "";
  private static String ignoreLinesBeginningWithOption04 = "";

  //Class constructor for ParameterFileParser
  public PackageToolsAndUtilities_FileParser()
  {
  }//This is the end of the class constructor

  public void InitializeStructuresForExtractionProcedure()
  {
    InitializationFileSections = new
String[SIZE_OF_INITIALIZATION_FILE_IN_LINES];
          InitializationFileSubsections = new
String[SIZE_OF_INITIALIZATION_FILE_IN_LINES];
          InitializationFileParameters = new
String[SIZE_OF_INITIALIZATION_FILE_IN_LINES];
  } //This is the end of the method definiton

  public void ExtractMainSectionsFromTargetFile(String[]
InitializationFileContents)
  {
  ignoreLinesBeginningWithOption01 = "//";
  ignoreLinesBeginningWithOption02 = " ";
  ignoreLinesBeginningWithOption03 = " ";
  ignoreLinesBeginningWithOption04 = " ";
```

```
  ExtractionProcedure(InitializationFileContents, InitializationFileSections,
ignoreLinesBeginningWithOption01,
                    ignoreLinesBeginningWithOption02,
ignoreLinesBeginningWithOption03, ignoreLinesBeginningWithOption04);
  }// This is the end of the method definition

  public void ExtractSubsectionsFromTargetFile(String[] InitializationFileContents)
  {
    ignoreLinesBeginningWithOption01 = "//";
          ignoreLinesBeginningWithOption02 = "<<";
          ignoreLinesBeginningWithOption03 = " ";
          ignoreLinesBeginningWithOption04 = " ";
          ExtractionProcedure(InitializationFileContents,
InitializationFileSubsections, ignoreLinesBeginningWithOption01,
                              ignoreLinesBeginningWithOption02,
ignoreLinesBeginningWithOption03, ignoreLinesBeginningWithOption04);
  } //This is the end of the method definition

  public void ExtractParametersFromTargetFile(String[] InitializationFileContents)
  {
    ignoreLinesBeginningWithOption01 = "//";
          ignoreLinesBeginningWithOption02 = "<<";
          ignoreLinesBeginningWithOption03 = "<";
          ignoreLinesBeginningWithOption04 = "-";
          ExtractionProcedure(InitializationFileContents,
InitializationFileParameters, ignoreLinesBeginningWithOption01,
                              ignoreLinesBeginningWithOption02,
ignoreLinesBeginningWithOption03, ignoreLinesBeginningWithOption04);
  } //This is the end of the method definition

  public void ExtractionProcedure(String[] sourceArray, String[] targetArray, String
firstIndicatorToIgnore,
                    String secondIndicatorToIgnore, String thirdIndicatorToIgnore,
String fourthIndicatorToIgnore)
  {
    int sizeOfArrayHoldingInitializationFile = sourceArray.length;
          int counterForSourceArray = 0;
          int counterForTargetArray = 0;
          String currentElementExtractedFromArrayList = "";

          for(counterForSourceArray = 0;
          counterForSourceArray < sizeOfArrayHoldingInitializationFile;
counterForSourceArray++)
          {
            currentElementExtractedFromArrayList =
sourceArray[counterForSourceArray];
```

```
          if(currentElementExtractedFromArrayList ==
END_OF_FILE_INDICATOR)
          {
            break;
          } //This is the end of the if structure

if(currentElementExtractedFromArrayList.startsWith(firstIndicatorToIgnore) ||

currentElementExtractedFromArrayList.startsWith(secondIndicatorToIgnore) ||

currentElementExtractedFromArrayList.startsWith(thirdIndicatorToIgnore) ||

currentElementExtractedFromArrayList.startsWith(fourthIndicatorToIgnore))
          {
            continue;
          } //This is the end of the if structure
          else
          {
            AddExtractedElementToTarget(currentElementExtractedFromArrayList,
targetArray, counterForTargetArray);
            counterForTargetArray++;
          } //This is the end of the else structure
        } //This is the end of the for loop

  } //This is the end of the method definition

  public void AddExtractedElementToTarget(String elementToAdd, String[]
targetStructure, int indexOfTargetStructure)
  {
    targetStructure[indexOfTargetStructure] = elementToAdd;
  } //This is the end of the method definition

  public void ShowRequestedSectionExtractedOK(String[]
targetStructureToDisplay)
  {
    int counterForDisplayLoop = 0;
        int sizeOfMainSections = targetStructureToDisplay.length;
        String contentOfCurrentElement = "";

        for(counterForDisplayLoop = 0; counterForDisplayLoop <
sizeOfMainSections; counterForDisplayLoop++)
        {
          contentOfCurrentElement =
targetStructureToDisplay[counterForDisplayLoop];
          System.out.println(contentOfCurrentElement);
        }
```

```
  } //This is the end of the method definition

} //This is the end of the class definition
```

Appendix 7: File Parameter Retrieval Source Code Listing

```
package Package_ToolsAndUtilities;

//Import statements
import java.lang.String;

public class PackageToolsAndUtilities_FileParameterRetrieval
{
  //Class variables
  private static String extractedParameterLabelAndValue = "";
  private static int markerOfCurrentlyExaminedElement = 0;
  private static int sizeOfParameterList = 0;
  private static int markerOfEqualSignWithinString = 0;
  public static String extractedParameterValueAsString = "";
  public static int extractedParameterValueAsInteger = 0;

  public PackageToolsAndUtilities_FileParameterRetrieval()
  {
  } //This is the end of class constructor ParameterRetrieval

  public String ExtractRequestedParameterAsString(String[]
arrayContainingTargetParameter, String requestedParameterName)
  {
    sizeOfParameterList = arrayContainingTargetParameter.length;
          for(markerOfCurrentlyExaminedElement = 0;
          markerOfCurrentlyExaminedElement < sizeOfParameterList;
          markerOfCurrentlyExaminedElement++)
          {
            extractedParameterLabelAndValue =
arrayContainingTargetParameter[markerOfCurrentlyExaminedElement];

if(extractedParameterLabelAndValue.startsWith(requestedParameterName))
            {
              markerOfEqualSignWithinString =
extractedParameterLabelAndValue.indexOf("=");
              extractedParameterValueAsString =

          extractedParameterLabelAndValue.substring(markerOfEqualSignWithinSt
ring + 1);
              break;
            } //This is the end of the if structure
          } //This is the end of the for loop
          return extractedParameterValueAsString;
  } //This is the end of the method definition
```

```
  public int ExtractRequestedParameterAsInteger(String[]
arrayContainingTargetParameter, String requestedParameterName)
  {
    sizeOfParameterList = arrayContainingTargetParameter.length;
          for(markerOfCurrentlyExaminedElement = 0;
          markerOfCurrentlyExaminedElement < sizeOfParameterList;
          markerOfCurrentlyExaminedElement++)
          {
      extractedParameterLabelAndValue =
arrayContainingTargetParameter[markerOfCurrentlyExaminedElement];

if(extractedParameterLabelAndValue.startsWith(requestedParameterName))
            {
        markerOfEqualSignWithinString =
extractedParameterLabelAndValue.indexOf("=");
                    extractedParameterValueAsInteger =

          Integer.parseInt(extractedParameterLabelAndValue.substring(markerOfEq
ualSignWithinString + 1));
                    break;
            } //This is the end of the if structure
          } //This is the end of the for structure
          return extractedParameterValueAsInteger;
  } //This is the end of the method definition

} //This is the end of the class definition
```

Appendix 8: File Display Source Code Listing

```
package Package_ToolsAndUtilities;

//Import statements
import java.lang.String;

public class PackageToolsAndUtilities_FileDisplay
{
  //Constant replacements
  private final static String END_OF_FILE_INDICATOR = null;

  public PackageToolsAndUtilities_FileDisplay()
  {} //This is the end of the class constructor

  public void DisplayRequestedSectionOfTargetFile(String[]
extractedSectionOfTargetFile)
  {
    int sizeOfArrayHoldingExtractedSection =
extractedSectionOfTargetFile.length;
          int counterForDisplayLoop = 0;
          String currentElementOfArray;

          for(counterForDisplayLoop = 0; counterForDisplayLoop <
sizeOfArrayHoldingExtractedSection;
          counterForDisplayLoop++)
          {
            currentElementOfArray =
                    extractedSectionOfTargetFile[counterForDisplayLoop];
            if(currentElementOfArray == END_OF_FILE_INDICATOR)
            {
              break;
            } //This is the end of the if structure
            else
            {
              System.out.println(currentElementOfArray);
            } //This is the end of the else structure
          } //This is the end of the for loop
  } //This is the end of the method definition

} //This is the end of the class definition
```

Appendix 9: File Writer Source Code Listing

```
package Package_ToolsAndUtilities;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class PackageToolsAndUtilities_FileWriter
{
  public static String PATH_TO_FILE = "";
  public static String NAME_OF_FILE = "";

  public PackageToolsAndUtilities_FileWriter()
  {} //This is the end of the class constructor

  public void WriteStringToFile(String stringToWriteIntoFile)
  {
    try
          {
          FileWriter outputFile = new FileWriter(NAME_OF_FILE, true);
                  BufferedWriter bufferedwriterWritingStreamToOutputFile = new
BufferedWriter(outputFile);
              System.out.println("CURRENT STRING BEING WRITTEN IS " +
stringToWriteIntoFile);
              bufferedwriterWritingStreamToOutputFile.write(stringToWriteIntoFile);
              bufferedwriterWritingStreamToOutputFile.newLine();
              bufferedwriterWritingStreamToOutputFile.close();
           } //This is the end of the try structure
     catch(IOException exceptionWhileAttemptingToWriteToFile)
          {} //This is the end of the catch structure
  } //This is the end of the method WriteStringToFile()

} //This is the end of the class definition
```

Appendix 10: Run Script File Source Code Listing

```
//Import statements
import java.io.*;
import java.lang.Runtime;

public class PackageToolsAndUtilities_RunScriptFile
{
  //Class constructor
  public PackageToolsAndUtilities_RunScriptFile()
  {} //This is the end of the method definition

  public static void main(String[] args)
  {
          File wd = new File("/bin");
          System.out.println(wd);
          Process proc = null;
          try {
            proc = Runtime.getRuntime().exec("/bin/bash", null, wd);
          }
          catch (IOException e) {
            e.printStackTrace();
          }
          if (proc != null) {
            BufferedReader in = new BufferedReader(new
InputStreamReader(proc.getInputStream()));
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(proc.getOutputStream())), true);
            out.println("cd ..");
            out.println("pwd");
            out.println("exit");
            try {
              String line;
              while ((line = in.readLine()) != null) {
                System.out.println(line);
              }
              proc.waitFor();
              in.close();
              out.close();
              proc.destroy();
            }
            catch (Exception e) {
              e.printStackTrace();
            }
          }
```

```
  } //This is the end of the method definition

} //This is the end of the class definition
```

Appendix 11: Serial Interface Access Class Source Code Listing

```
package serialInterfaceUtility;

import gnu.io.*;
import java.io.*;
import java.util.*;
//import fileUtility.*;

public class Access
{
  //Class variables
  //private static int responseLength = 0;
  public static int SERIAL_INTERFACE_TIMEOUT_VALUE = 0;
  public static String NAME_OF_APPLICATION = "";

  public Access()
  {
  } //This is the end of constructor Access()

  public static CommPortIdentifier GetSerialInterfaceID(String
stringRequestedInterfaceName)
  {
    //Get an enumeration of all ports known to JavaComm
         Enumeration enumerationSerialIntefaceIdentifiers =
CommPortIdentifier.getPortIdentifiers();

    // will be set if port found
    CommPortIdentifier commportidentifierRequestedSerialInterfaceID = null;

    // Check each port identifier if
    // (a) it indicates a serial (not a parallel) port, and
    // (b) matches the desired name.
    while (enumerationSerialIntefaceIdentifiers.hasMoreElements())
    {
      CommPortIdentifier commportidentifierCurrentInterfaceID =
(CommPortIdentifier) enumerationSerialIntefaceIdentifiers.nextElement();
      if(commportidentifierCurrentInterfaceID.getPortType() ==
CommPortIdentifier.PORT_SERIAL &&

commportidentifierCurrentInterfaceID.getName().equals(stringRequestedInterfac
eName))
      {
        commportidentifierRequestedSerialInterfaceID =
commportidentifierCurrentInterfaceID;
```

```
        break;
      } //This is the end of the if structure
    } //This is the end of the while structure
    return commportidentifierRequestedSerialInterfaceID;
  } //This is the end of method GetSerialPortID

  /*
   * Open the seral port
   */
  public static SerialPort OpenSerialInterface(CommPortIdentifier
commportidentifierSerialInterfaceID) {
      SerialPort serialportInitializedSerialInterface = null;
      try
      {
        // Name of the application and timeout for acquiring the port
        serialportInitializedSerialInterface = (SerialPort)
commportidentifierSerialInterfaceID.open(NAME_OF_APPLICATION,
SERIAL_INTERFACE_TIMEOUT_VALUE);
      }
      catch(Exception exceptionWhileOpeningSerialInterface){}
      return serialportInitializedSerialInterface;
  } //This is the end of method OpenSerialPort()

  /*
   * Initialize and associate the Input Stream with the seral port
   */
  public static InputStream InitializeSerialInterfaceInputStream(SerialPort
serialportOpenedSerialInterface)
  {
    InputStream inputstreamInputStreamForOpenedSerialInterface = null;
    try
    {
      inputstreamInputStreamForOpenedSerialInterface =
serialportOpenedSerialInterface.getInputStream();
    } //This is the end of the try structure
    catch (Exception exceptionWhileOpeningInputStreamForSerialPort)
    {
    } //This is the end of the catch structure
    return inputstreamInputStreamForOpenedSerialInterface;
  } //This is the end of method InitializeSerialPortInputStream()

  /*
   * Initizlizes and associates the Output Stream with the serial port
   */
  public static OutputStream InitializeSerialInterfaceOutputStream(SerialPort
serialportOpenedSerialInterface)
```

```
  {
    OutputStream outputstreamForSerialInterface = null;
    try
    {
      outputstreamForSerialInterface =
serialportOpenedSerialInterface.getOutputStream();
    }
    catch (Exception exceptionWhileInitializingSerialInterface)
    {
    } //This is the end of the catch structure
    return outputstreamForSerialInterface;
  } //This is the end of method InitializeSerialPortOutputStream()

} //This is the end of class Access
```

Appendix 12: Serial Interface Configuration Source Code Listing

```
package serialInterfaceUtility;

import gnu.io.*;
//import java.io.*;
//import java.util.*;

public class Configuration
{
        public static int integerSPECIFYING_BAUD_RATE = 9600;

        public Configuration()
        {
        } //This is the end of the Serial Interface constructor

   /*
    * Set serial port parameters
    */
   public static void SetSerialInterfaceParameters(SerialPort
serialportOpenedSerialInterfaceID) {
       try
       {
         // baud rate needs to be dynamic

serialportOpenedSerialInterfaceID.setSerialPortParams(integerSPECIFYING_BA
UD_RATE,
                    SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1,
                    SerialPort.PARITY_NONE);
       } //This is the end of the try structure
       catch (UnsupportedCommOperationException
exceptionWhileSettingSerialInterfaceParameters){
       } //This is the end of the catch structure
   }

} //This is the end of class InterfaceConfiguration
```

Appendix 13: RFID Reader Access Source Code Listing

```
package rfidReader;

import gnu.io.SerialPort;
import java.io.InputStream;

public class Access
{
  //Class variables
  private static int lengthOfResponseMessageFromReader = 0;

  public Access()
  {
  } //This is the end of constructor ReaderAccess()

  /*
   * Read bytes from m5e reader and converts the signed bytes
   * to unsigned.
   */
  public static int[] GetDataFromReader(InputStream inputstreamFromReader,
SerialPort serialportConnectedToReader)
  {
    //Data itself is 250 bytes max
    byte[] byteReadBuffer = new byte[260];
    int[] unsignedintegerReadBuffer = new int[260];
    byte testByteToCheckMessageHeader = 0;

    //Read the bytes from the m5e Reader
    try
    {
      //Test for the header byte of 0xFF before reading the rest of the message.
      // Signed 0xFF = -1
      while((testByteToCheckMessageHeader) != (-1))
      {
        inputstreamFromReader.read(byteReadBuffer, 0, 1);
        testByteToCheckMessageHeader = byteReadBuffer[0];
      } //This is the end of the while structure

      //Read data length 1 byte, offset 1, read 1 byte
      inputstreamFromReader.read(byteReadBuffer, 1, 1);

      //Converting from signed byte to unsigned to get the length of the data field
      byte temporaryHolderForDataFieldLength = byteReadBuffer[1];
      int dataFieldLength = temporaryHolderForDataFieldLength & 0xFF;
```

```
      //To also read the Command, Status Word and CRC
      dataFieldLength = dataFieldLength + 5;

      // Update the total length of the message
      lengthOfResponseMessageFromReader = dataFieldLength + 2;

      // +2 to account for the offset of the byte array
      for(int indicatorForOffsetProcedure = 2;
      indicatorForOffsetProcedure < dataFieldLength + 2;
indicatorForOffsetProcedure++)
      {
        inputstreamFromReader.read(byteReadBuffer,
indicatorForOffsetProcedure, 1);
      } //This is the end of the for structure
    } //This is the end of the try structure
    catch (Exception exceptionWhileAttemptingToReadBytesFromReader){
    } //This is the end of the catch structure

    // Convert signed bytes to unsigned bytes
    for(int markerOfCurrentElementOfReadBuffer = 0;
    markerOfCurrentElementOfReadBuffer <
lengthOfResponseMessageFromReader;
    markerOfCurrentElementOfReadBuffer++)
    {
      unsignedintegerReadBuffer[markerOfCurrentElementOfReadBuffer] =
byteReadBuffer[markerOfCurrentElementOfReadBuffer] & 0xFF;
    } //This is the end of the for structure

    return unsignedintegerReadBuffer;
  } //This is the end of the method GetDataFromReader()

  /*
   * Gets the message from the Reader's response and returns only
   * the Tag ID field. This function is specific to the sample EPC Tags
   *
   */
  public static int[] GetRFIDTagID(int[] integerarrayOfReadBuffer)
  {
    int[] integerarrayOfTagID = new int[12];

    for (int integerPointingToTagIDFieldInResponseMessage = 0;
integerPointingToTagIDFieldInResponseMessage < 12;
    integerPointingToTagIDFieldInResponseMessage++)
    {
      integerarrayOfTagID[integerPointingToTagIDFieldInResponseMessage] =
```

```
integerarrayOfReadBuffer[integerPointingToTagIDFieldInResponseMessage+9];
    } //This is the end of the for structure

    return integerarrayOfTagID;
  } //This is the end of method GetRFIDTagID()

  /*
   * Returns the length of the most recent processed message from the
   * m5e Reader
   *
   * Requires: N/A
   * Returns: Integer value of the length of the whole message
   */
  public static int GetLengthOfMessageFromReader()
  {
    return lengthOfResponseMessageFromReader;
  } //This is the end of method GetLengthOfMessageFromReader

  /*
   * CRC-16 Calculation used in all messages with the m5e reader.
   * calculated using the Command, Status Word, Data Length, and Data of the
   * message and is appended as the last 2 bytes of the message.
   */
  public static byte[] PerformCyclicRedundancyCheck16Procedure(byte[]
bytearrayOfDataToBeSentToReader)
  {
    int cyclicRedundancyCheckValue = 0xFFFF;          // initial value
    int integerPolynomialValue = 0x1021;   // 0001 0000 0010 0001  (0, 5, 12)

    byte[] bytearrayOfTemporaryCopyOfData =
bytearrayOfDataToBeSentToReader;

    for (int markerOfCurrentByteElement = 0; markerOfCurrentByteElement <
bytearrayOfTemporaryCopyOfData.length; markerOfCurrentByteElement++)
    {
      for (int i = 0; i < 8; i++)
      {
      boolean bit =
((bytearrayOfTemporaryCopyOfData[markerOfCurrentByteElement] >> (7-i) & 1)
== 1);
      boolean c15 = ((cyclicRedundancyCheckValue >> 15 & 1) == 1);
      cyclicRedundancyCheckValue <<= 1;
      cyclicRedundancyCheckValue |= (bit)?1:0;
        if(c15)
        {
```

```
          cyclicRedundancyCheckValue ^= integerPolynomialValue;
        } //This is the end of the if structure
      } //This is the end of the for loop
    } //This is the end of the for loop
      // Cut off unwanted bits
    cyclicRedundancyCheckValue &= 0xFFFF;

    //Convert crc to hex string
    String CRCValueAsHexadecimalString =
Integer.toHexString(cyclicRedundancyCheckValue);

    //Convert hex string to byte[]
    byte[] bytearrayContainingCRCValueAsHexadecimal = new byte[2];
    for (int markerOfCurrentElementOfArray = 0; markerOfCurrentElementOfArray
< bytearrayContainingCRCValueAsHexadecimal.length;
    markerOfCurrentElementOfArray++)
    {

bytearrayContainingCRCValueAsHexadecimal[markerOfCurrentElementOfArray]
=

(byte)Integer.parseInt(CRCValueAsHexadecimalString.substring(2*markerOfCurr
entElementOfArray, 2*markerOfCurrentElementOfArray+2), 16);
    } //This is the end of the for loop

    return bytearrayContainingCRCValueAsHexadecimal;
  } //This is the end of method PerformCyclicRedundancyCheck16Proedure()

  /*
    * Takes a byte[] and calls PerformCyclicRedundancyCheck16Proedure() and
uses that result to create
    * a proper message to the M5e Reader
    *
    * Requires: a byte[] without the header (0xFF)
    * Returns: returns the full message with header and crc ready to be sent
    */
  public static byte[] GenerateMessage(byte[] bytearrayReceivedContainingData)
  {
     //To account for the header (1 byte) and the CRC (2 bytes)
     byte[] bytearrayContainingMessage = new
byte[bytearrayReceivedContainingData.length + 3];

     //Calculate CRC
     byte[] bytearrayContainingCyclicRedundancyCheckValue = new byte[2];
```

```
   bytearrayContainingCyclicRedundancyCheckValue =
PerformCyclicRedundancyCheck16Procedure(bytearrayReceivedContainingData
);

   //Form the message, header, data, crc
   bytearrayContainingMessage[0] = (byte)0xFF;
   for(int integerPointingToCurrentElementOfDataArray = 1;
   integerPointingToCurrentElementOfDataArray <
bytearrayReceivedContainingData.length+1;
   integerPointingToCurrentElementOfDataArray++)
   {
     bytearrayContainingMessage[integerPointingToCurrentElementOfDataArray]
=

bytearrayReceivedContainingData[integerPointingToCurrentElementOfDataArray
-1];
   } //This is the end of the for structure
   bytearrayContainingMessage[bytearrayContainingMessage.length-2] =
bytearrayContainingCyclicRedundancyCheckValue[0];
   bytearrayContainingMessage[bytearrayContainingMessage.length-1] =
bytearrayContainingCyclicRedundancyCheckValue[1];

   return bytearrayContainingMessage;
  } //This is the end of method GenerateMessage()

} //This is the end of class ReaderAccess
```

Appendix 14: RFID Reader Configuration Source Code Listing

```
package rfidReader;

public class Configuration
{
  public byte[] OPCODE_BYTES_FOR_GET_VERSION = {(byte)0xFF,
(byte)0x00, (byte)0x03, (byte)0x1D, (byte)0x0C};
  public byte[] OPCODE_BYTES_FOR_BOOT_FIRMWARE = {(byte)0xFF,
(byte)0x00, (byte)0x04, (byte)0x1D, (byte)0x0B};
  public byte[] OPCODE_BYTES_FOR_SET_REGION = {(byte)0xFF, (byte)0x01,
(byte)0x97, (byte)0x01, (byte)0x4B, (byte)0xBC};
  public byte[] OPCODE_BYTES_FOR_SET_TAG_PROTOCOL = {(byte)0xFF,
(byte)0x02, (byte)0x93, (byte)0x00, (byte)0x05, (byte)0x51, (byte)0x7D};
  public byte[] OPCODE_BYTES_FOR_SET_TX_POWER = {(byte)0xFF,
(byte)0x02, (byte)0x92, (byte)0x09, (byte)0xC4, (byte)0x48, (byte)0x9D};
  public byte[] OPCODE_BYTES_FOR_SET_ANTENNA_PORT = {(byte)0xFF,
(byte)0x02, (byte)0x91, (byte)0x01, (byte)0x01, (byte)0x70, (byte)0x3B};
  public byte[] OPCODE_BYTES_FOR_READ_TAGS = {(byte)0x02, (byte)0x21,
(byte)0x03, (byte)0xE8};
  public byte[] OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS = {(byte)0x02,
(byte)0x22, (byte)0x03, (byte)0xE8};
  public byte[] OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG =
{(byte)0x02, (byte)0x29, (byte)0x00, (byte)0x01};
  public byte[] OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER =
{(byte)0x00, (byte)0x2A};

  public Configuration()
  {} //This is the end of the Configuration() constructor

  public byte[] ConvertStringToBytes(String stringToConvert, int
integerStringDivisor, int integerSubstringSize)
  {
    int integerCounterForSegmentationLoop = 0;
    int integerActualValueOfCurrentStringElement = 0;
    int integerMarkingFirstElementOfSubstring = 0;
    int integerSizeOfString = stringToConvert.length();
    int integerIndexOfbyteArray = 0;
    int integerRepresentationOfSubstring = 0;
          String stringSubstringSegment = "";
    byte[] byteRepresentationOfString = new byte[integerStringDivisor];

    for(integerCounterForSegmentationLoop = 0;
integerCounterForSegmentationLoop < integerSizeOfString;
    integerCounterForSegmentationLoop++)
```

```
    {
      integerActualValueOfCurrentStringElement =
(integerCounterForSegmentationLoop + 1);
      integerMarkingFirstElementOfSubstring =
(integerActualValueOfCurrentStringElement - integerSubstringSize);
      if((integerActualValueOfCurrentStringElement%integerSubstringSize) == 0)
      {
        stringSubstringSegment =
stringToConvert.substring(integerMarkingFirstElementOfSubstring,
integerActualValueOfCurrentStringElement);
        integerRepresentationOfSubstring =
Integer.parseInt(stringSubstringSegment, 16);
        //} //This is the end of the if structure
        byteRepresentationOfString[integerIndexOfbyteArray] =
(byte)(integerRepresentationOfSubstring);
        integerIndexOfbyteArray++;
        //System.out.println(stringSubstringSegment);
      } //This is the end of the if structure
    } //This is the end of the for structure
    //System.out.println(byteRepresentationOfString);
          return byteRepresentationOfString;
   } //This is the end of method ConvertStringToBytes()

} //This is the end of class Configuration
```

Appendix 15: RFID Tag Reading Source Code Listing

```
import gnu.io.*;
import java.io.*;
import java.util.*;
import serialInterfaceUtility.*;
import rfidReader.*;
import fileUtility.Writer;

public class TagReading extends Thread
{
  //Variables related to the RFID reader configuration
  private static byte[] OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS;
  private static byte[]
OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG;
  private static byte[] OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER;

  //Variables related to the serial interface
  private static int[] arrayHoldingTagID;
  private static CommPortIdentifier serialInterfaceID;
  private static SerialPort serialInterfaceForRFID;
  private static OutputStream outputstreamToSerialInterface = null;
  private static InputStream inputstreamFromSerialInterface = null;
  //Data itself is 250 bytes max
  private static int[] unsignedReadBuffer = new int[260];
  //The total length of the response from the m5e Reader
  private static int messageResponseLength = 0;
  //The number of tags read from READ MULTIPLE TAGS command
  private static int numberOfRFIDTags = 0;
  // The value of the length of the data field
  private static int lengthOfDataField = 0;
  private static int counterForTotalNumberOfTagsRead = 0;

  //Variables related to file writing
  private static String PATH_TO_TAG_ID_FILE = "";
  private static fileUtility.Writer writeToTagIDFile;
  private static String tagIDAsString = "";

  public TagReading(CommPortIdentifier serialInterfaceIDReceived, SerialPort
serialInterfaceForRFIDReceived,
                    byte[]
OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS_RECEIVED, byte[]
OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG_RECEIVED,
```

```
                byte[]
OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER_RECEIVED, String
PATH_TO_TAG_ID_FILE_RECEIVED)
 {
        this.serialInterfaceID = serialInterfaceIDReceived;
        this.serialInterfaceForRFID = serialInterfaceForRFIDReceived;
        this.OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS =
OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS_RECEIVED;
        this.OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG =
OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG_RECEIVED;
        this.OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER =
OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER_RECEIVED;
        this.PATH_TO_TAG_ID_FILE = PATH_TO_TAG_ID_FILE_RECEIVED;
        writeToTagIDFile = new fileUtility.Writer();
        //writeToTagIDFile.PATH_TO_FILE = PATH_TO_TAG_ID_FILE;
 } //This is the end of the class constructor

 public void run()
 {
    while(true){
    //-------------------------------------------------------------
    serialInterfaceForRFID =
serialInterfaceUtility.Access.OpenSerialInterface(serialInterfaceID);
    System.out.println("Opened serial interface...");

    //Create an InputStream associated with the port.
    inputstreamFromSerialInterface =
serialInterfaceUtility.Access.InitializeSerialInterfaceInputStream(serialInterfaceFo
rRFID);
    System.out.println("Opened input stream from interface...");

    //Create an OutputStream associated with the port.
    outputstreamToSerialInterface =
serialInterfaceUtility.Access.InitializeSerialInterfaceOutputStream(serialInterface
ForRFID);
    System.out.println("Opened output stream to serial interface...");

    try
    {

outputstreamToSerialInterface.write(rfidReader.Access.GenerateMessage(OPC
ODE_BYTES_FOR_READ_MULTIPLE_TAGS));
      outputstreamToSerialInterface.flush();
      System.out.println("Wrote opcode for READ MULTIPLE TAGS...");
    } //This is the end of the try structure
```

```
    catch(Exception exceptionWhileCreatingOutputStreamToInterface) {} //This is
the end of the catch structure

    //Set the baudrate, stop bits, data bits

serialInterfaceUtility.Configuration.SetSerialInterfaceParameters(serialInterfaceF
orRFID);
    System.out.println("Set baudrate, stop bits and data bits...");

    // Get the response from the m5e reader
    unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
    System.out.println("Got response from reader...");
    //-------------------------------------------------------------

    // Get the length field of the response message from the first element of the
read buffer
    lengthOfDataField = unsignedReadBuffer[1];
    System.out.println("Got length of data field from read buffer. Field length is "
+ lengthOfDataField);

    // Get the number of tags found from the fifth element of the read buffer
    numberOfRFIDTags = unsignedReadBuffer[5];
    System.out.println("Got number of tags in read buffer. Number of tags found:
" + numberOfRFIDTags);

    //-------------------------------------------------------------
    if(lengthOfDataField > 0 && numberOfRFIDTags !=132)
    {
        for(int counterForTagIDLoop = 0; counterForTagIDLoop <
numberOfRFIDTags; counterForTagIDLoop++)
      {
          try
        {

outputstreamToSerialInterface.write(rfidReader.Access.GenerateMessage(OPC
ODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG));
          outputstreamToSerialInterface.flush();
          System.out.println("Wrote opcode for GET TAG ID, BUFFER ONE
TAG...");
        } //This is the end of the try structure
        catch(Exception e) {} //This is the end of the catch structure

        //Get the response from the m5e reader
```

```
        unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
        System.out.println("Got response from reader...");

        //Get the total length of the message
        messageResponseLength =
rfidReader.Access.GetLengthOfMessageFromReader();
        System.out.println("Got total length of message from reader...");

        //Print information
        //This function is specific to the Sample EPOC Tags
         arrayHoldingTagID =
rfidReader.Access.GetRFIDTagID(unsignedReadBuffer);
         tagIDAsString =
Integer.toHexString(arrayHoldingTagID[counterForTagIDLoop]);

         System.out.println("Accessing Writer to write tag id's to text file...");
         System.out.println("TAG ID BEING WRITTEN IS " + tagIDAsString);
         writeToTagIDFile.WriteStringToFile(tagIDAsString);
       }//This is the end of the for loop
      } //This is the end of the if structure

      //Reset Values
      numberOfRFIDTags = 0;
      lengthOfDataField = 0;
      System.out.println("Reset counter for number of RFID Tags and length of
data field...");

      // Clear the Tag ID Buffer
      try
      {

outputstreamToSerialInterface.write(rfidReader.Access.GenerateMessage(OPC
ODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER));
        outputstreamToSerialInterface.flush();
        System.out.println("Wrote opcode for CLEAR TAG ID BUFFER...");
      } //This is the end of the try structure
      catch (Exception exceptionWhileClearingTheTagIDBuffer){} //This is the end
of the catch structure

      try
      {
        outputstreamToSerialInterface.close();
        System.out.println("Closed output stream to serial interface...");
        inputstreamFromSerialInterface.close();
```

```
        System.out.println("Closed input stream from serial interface...");
      } //This is the end of the try structure
      catch (Exception e) {} //This is the end of the catch structure
      serialInterfaceForRFID.close();
      System.out.println("Closed serial interface...");
     }//This is the end of the while loop
  } //This is the end of method run()

} //This is the end of the class definition
```

Appendix 16: RFID System Embedded Source Code Listing

```
//Import the RXTX Serial API utility
import gnu.io.*;
import java.io.*;
import java.util.*;
import java.lang.String;
import serialInterfaceUtility.*;
import rfidReader.*;
import fileUtility.*;

public class RFIDSystemEmbedded
{
  //Class variables
  //Variables related to Initialization file manipulation
  private static String LOCATION_OF_INITIALIZATION_FILE =
"/home/sage/workspace/RFIDTesting/fileUtility/RFIDPackageOperatingParamete
rs.txt";
  private static int SIZE_OF_MANIPULATED_FILE_IN_LINES = 61;
  private static fileUtility.Handler handlerOfRFIDInitializationFile;
  private static fileUtility.Parser parserOFRFIDInitializationFile;
  private static fileUtility.ParameterRetrieval
retrieveRFIDInitializationFileParameters;
  //Variables related to file writing
  private static String PATH_TO_TAG_ID_FILE = "";
  //private static FileWriter outputFileOfTagIDs;
  //private static BufferedWriter bufferedwriterWritingStreamToOutputFile;
  //Variables related to tag reading
  private static TagReading tagReader;

  //Variables related to RFID Reader configuration
  private static rfidReader.Configuration configurationOfRFIDReader;
  private static String RFID_OPCODE_FOR_GET_VERSION = "";
  private static byte[] OPCODE_BYTES_FOR_GET_VERSION = {(byte)0xFF,
(byte)0x00, (byte)0x03, (byte)0x1D, (byte)0x0C};
  private static String RFID_OPCODE_FOR_BOOT_FIRMWARE = "";
  private static byte[] OPCODE_BYTES_FOR_BOOT_FIRMWARE = {(byte)0xFF,
(byte)0x00, (byte)0x04, (byte)0x1D, (byte)0x0B};
  private static String RFID_OPCODE_FOR_SET_REGION = "";
  private static byte[] OPCODE_BYTES_FOR_SET_REGION = {(byte)0xFF,
(byte)0x01, (byte)0x97, (byte)0x01, (byte)0x4B, (byte)0xBC};
  private static String RFID_OPCODE_FOR_SET_TAG_PROTOCOL = "";
  private static byte[] OPCODE_BYTES_FOR_SET_TAG_PROTOCOL =
{(byte)0xFF, (byte)0x02, (byte)0x93, (byte)0x00, (byte)0x05, (byte)0x51,
(byte)0x7D};
```

```
  private static String RFID_OPCODE_FOR_SET_TX_POWER = "";
  private static byte[] OPCODE_BYTES_FOR_SET_TX_POWER = {(byte)0xFF,
(byte)0x02, (byte)0x92, (byte)0x09, (byte)0xC4, (byte)0x48, (byte)0x9D};
  private static String RFID_OPCODE_FOR_SET_ANTENNA_PORT = "";
  private static byte[] OPCODE_BYTES_FOR_SET_ANTENNA_PORT =
{(byte)0xFF, (byte)0x02, (byte)0x91, (byte)0x01, (byte)0x01, (byte)0x70,
(byte)0x3B};
  private static String RFID_OPCODE_FOR_READ_TAGS = "";
  private static byte[] OPCODE_BYTES_FOR_READ_TAGS = {(byte)0x02,
(byte)0x21, (byte)0x03, (byte)0xE8};
  private static String RFID_OPCODE_FOR_READ_MULTIPLE_TAGS = "";
  private static byte[] OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS =
{(byte)0x02, (byte)0x22, (byte)0x03, (byte)0xE8};
  private static String RFID_OPCODE_FOR_GET_ID_TAG_BUFFER_ONE_TAG
= "";
  private static byte[]
OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG = {(byte)0x02,
(byte)0x29, (byte)0x00, (byte)0x01};
  private static String RFID_OPCODE_FOR_CLEAR_TAG_ID_BUFFER = "";
  private static byte[] OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER =
{(byte)0x00, (byte)0x2A};

  //Variables related to Serial Interface
  private static CommPortIdentifier serialInterfaceID;
  private static SerialPort serialInterfaceForRFID;
  private static OutputStream outputstreamToSerialInterface = null;
  private static InputStream inputstreamFromSerialInterface = null;
  private static String DESIGNATED_NAME_OF_APPLICATION = "Demo
Reader";
  private static int SERIAL_INTERFACE_TIMEOUT_VALUE = 0;
  private static String SERIAL_PORT_BEING_USED = "";
  //Data itself is 250 bytes max
  private static int[] unsignedReadBuffer = new int[260];

  public RFIDSystemEmbedded()
  {} //This is the end of the class constructor

  private static void ManipulateInitializationFile()
  {
    /*
          * Creates new object to handle the RFID parameter file
          * Sets the path to the location of the RFID parameter file
          * Assigns the data structure responsible for containing the complete
contents of the file
          * */
           System.out.println("Opening and reading Initialization File...");
```

```
        handlerOfRFIDInitializationFile = new fileUtility.Handler();
        Handler.WORKING_PATH_DIRECTORY =
LOCATION_OF_INITIALIZATION_FILE;
        Handler.SIZE_OF_INITIALIZATION_FILE_IN_LINES =
SIZE_OF_MANIPULATED_FILE_IN_LINES;

        /*
         * Opens the RFID initialization file, parses it and closes it once done
         * */
        handlerOfRFIDInitializationFile.OpenFile();
        handlerOfRFIDInitializationFile.ParseFileIntoLines();
        handlerOfRFIDInitializationFile.CloseFile();

        /*
         * Creates a new object to parse the RFID parameter file.
         * It is provided with the path to the location of the RFID parameter file.
         * Assigns the data structures that will hold the main sections, subsections
and parameters
         * of the parameter file.
         * */
        System.out.println("Parsing Initialization File...");
        parserOFRFIDInitializationFile = new fileUtility.Parser();
        Parser.LOCATION_OF_TARGET_FILE =
LOCATION_OF_INITIALIZATION_FILE;
        Parser.SIZE_OF_INITIALIZATION_FILE_IN_LINES =
SIZE_OF_MANIPULATED_FILE_IN_LINES;

        /*
         * Extracts Main Sections, Subsections and Parameters from RFID
initialization file
         * */

parserOFRFIDInitializationFile.InitializeStructuresForExtractionProcedure();

parserOFRFIDInitializationFile.ExtractMainSectionsFromTargetFile(handlerOfRFI
DInitializationFile.contentsOfFile);
        System.out.println("Main Sections Extracted From Initialization File...");

parserOFRFIDInitializationFile.ExtractSubsectionsFromTargetFile(handlerOfRFI
DInitializationFile.contentsOfFile);
        System.out.println("Subsections Extracted From Initialization File...");

parserOFRFIDInitializationFile.ExtractParametersFromTargetFile(handlerOfRFID
InitializationFile.contentsOfFile);
        System.out.println("Parameters Extracted From Initialization File...");
```

```
//parserOFRFIDInitializationFile.ShowRequestedSectionExtractedOK(parserOFR
FIDInitializationFile.InitializationFileParameters);
  } //This is the end of method ManipulateInitializationFile()

  private static void RetrieveAndSetParameterValues()
  {
    retrieveRFIDInitializationFileParameters = new fileUtility.ParameterRetrieval();

    //Retrieve and set RFID scanner opcodes
    System.out.println("Retrieving RFID Opcodes...");
    RFID_OPCODE_FOR_GET_VERSION =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"GET_VERSION");
    RFID_OPCODE_FOR_BOOT_FIRMWARE =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"BOOT_FIRMWARE");
    RFID_OPCODE_FOR_SET_REGION =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters, "SET_REGION");
    RFID_OPCODE_FOR_SET_TAG_PROTOCOL =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"SET_TAG_PROTOCOL");
    RFID_OPCODE_FOR_SET_TX_POWER =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"SET_TX_POWER");
    RFID_OPCODE_FOR_SET_ANTENNA_PORT =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"SET_ANTENNA_PORT");
    RFID_OPCODE_FOR_READ_TAGS =

          retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters, "READ_TAGS");
    RFID_OPCODE_FOR_READ_MULTIPLE_TAGS =
```

```
        retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"READ_MULTIPLE_TAGS");
   RFID_OPCODE_FOR_GET_ID_TAG_BUFFER_ONE_TAG =

        retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"GET_TAG_ID_BUFFER_ONE_TAG");
   RFID_OPCODE_FOR_CLEAR_TAG_ID_BUFFER =

        retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"CLEAR_TAG_ID_BUFFER");

   //Retrieve and set Serial port timeout value
   System.out.println("Retrieving Serial Port Timeout Value...");
   SERIAL_INTERFACE_TIMEOUT_VALUE =

        retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsInt
eger(parserOFRFIDInitializationFile.InitializationFileParameters,
"SERIAL_PORT_TIMEOUT_VALUE");

   //Retrieve and set name of serial port being used
   System.out.println("Retrieving Serial Port Being Used...");
   SERIAL_PORT_BEING_USED =

        retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsSt
ring(parserOFRFIDInitializationFile.InitializationFileParameters,
"PORT_NAME_01");

   //Retrieve and set path to file for storing Tag ID's
   PATH_TO_TAG_ID_FILE =
retrieveRFIDInitializationFileParameters.ExtractRequestedParameterAsString(pa
rserOFRFIDInitializationFile.InitializationFileParameters,
"PATH_TO_TAG_ID_FILE");
 }//This is the end of method RetrieveAndSetParameterValues()

 private static void PrepareSerialInterface()
 {
   ResetSerialInterface();

   serialInterfaceUtility.Access accessToSerialInterface = new
serialInterfaceUtility.Access();
   System.out.println("Setting serial interface timeout value...");
```

```
   accessToSerialInterface.SERIAL_INTERFACE_TIMEOUT_VALUE =
SERIAL_INTERFACE_TIMEOUT_VALUE;
   System.out.println("Setting Name of Application...");
   accessToSerialInterface.NAME_OF_APPLICATION =
DESIGNATED_NAME_OF_APPLICATION;

   //Identify the port to which the reader is attached to the PC.
   System.out.println("Getting serial interface ID...");
   serialInterfaceID =
accessToSerialInterface.GetSerialInterfaceID(SERIAL_PORT_BEING_USED);
   //Open and claim the ownership of the port.
   serialInterfaceForRFID =
accessToSerialInterface.OpenSerialInterface(serialInterfaceID);
   System.out.println("Serial port opened");
   //Create an InputStream associated with the port.
   inputstreamFromSerialInterface =
accessToSerialInterface.InitializeSerialInterfaceInputStream(serialInterfaceForRF
ID);
   System.out.println("Input stream from serial port created...");
   //Create an OutputStream associated with the port.
   outputstreamToSerialInterface =
accessToSerialInterface.InitializeSerialInterfaceOutputStream(serialInterfaceFor
RFID);
   System.out.println("Output stream to serial port created...");
 } //This is the end of method ConfigureRFIDReader()

 private static void ResetSerialInterface()
 {
        System.out.println("Resetting Serial Interface Parameters...");
        serialInterfaceID = null;
        serialInterfaceForRFID = null;
        outputstreamToSerialInterface = null;
        inputstreamFromSerialInterface = null;
 } //This is the end of method ResetSerialInterface()

 private static void PrepareRFIDReader()
 {
        configurationOfRFIDReader = new rfidReader.Configuration();
/*
        OPCODE_BYTES_FOR_GET_VERSION =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_GET
_VERSION", 5, 2);
        OPCODE_BYTES_FOR_BOOT_FIRMWARE =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_BOO
T_FIRMWARE", 5, 2);
```

```
        OPCODE_BYTES_FOR_SET_REGION =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_SET
_REGION", 6, 2);
        OPCODE_BYTES_FOR_SET_TAG_PROTOCOL =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_SET
_TAG_PROTOCOL", 7, 2);
        OPCODE_BYTES_FOR_SET_TX_POWER =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_SET
_TX_POWER", 7, 2);
        OPCODE_BYTES_FOR_SET_ANTENNA_PORT =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_SET
_ANTENNA_PORT", 7, 2);
        OPCODE_BYTES_FOR_READ_TAGS =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_REA
D_TAGS", 4, 2);
        OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_REA
D_MULTIPLE_TAGS", 4, 2);
        OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_GET
_ID_TAG_BUFFER_ONE_TAG", 4, 2);
        OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER =
configurationOfRFIDReader.ConvertStringToBytes("RFID_OPCODE_FOR_CLE
AR_TAG_ID_BUFFER", 2, 2);
*/
        //Get version
        try {

outputstreamToSerialInterface.write(OPCODE_BYTES_FOR_GET_VERSION);
          outputstreamToSerialInterface.flush();
          System.out.println("Wrote opcode for GET VERSION...");
        } //This is the end of the try structure
        catch (Exception e){} //This is the end of the catch structure

        serialInterfaceUtility.Configuration configurationOfSerialInterface = new
serialInterfaceUtility.Configuration();

   // Set the baudrate, stop bits, data bits
        serialInterfaceUtility.Configuration.SetSerialInterfaceParameters(serialInte
rfaceForRFID);

   // Get the response from the m5e reader
        unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
        System.out.println("Got response from RFID reader...");
```

```
   // Boot Firmware FF 00 04 (1D 0B)
   try
   {

outputstreamToSerialInterface.write(OPCODE_BYTES_FOR_BOOT_FIRMWAR
E);
     outputstreamToSerialInterface.flush();
     System.out.println("Wrote opcode for BOOT FIRMWARE...");
   } //This is the end of the try structure
   catch (Exception e){} //This is the end of the catch structure

   // Get the response from the m5e reader
   unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
   System.out.println("Got response from RFID reader...");

   try
   {

outputstreamToSerialInterface.write(OPCODE_BYTES_FOR_SET_REGION);
     outputstreamToSerialInterface.flush();
     System.out.println("Wrote opcode for SET REGION...");
   } //This is the end of the try structure
   catch (Exception e){} //This is the end of the catch structure

   //Get the response from the m5e reader
   unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
   System.out.println("Got response from RFID reader...");

   // Set Current Tag Protocol (GEN2)  FF 02 93 00 05
   try
   {

outputstreamToSerialInterface.write(OPCODE_BYTES_FOR_SET_TAG_PROT
OCOL);
     outputstreamToSerialInterface.flush();
     System.out.println("Wrote opcode for SET TAG PROTOCOL...");
   } //This is the end of the try structure
   catch (Exception e) {} //This is the end of the catch structure

   // Get the response from the m5e reader
```

```
   unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
   System.out.println("Got response from RFID reader...");

   // Set Read TX Power (25dBm) FF 02 92 09 C4
   try
   {

outputstreamToSerialInterface.write(OPCODE_BYTES_FOR_SET_TX_POWER
);
     outputstreamToSerialInterface.flush();
     System.out.println("Wrote opcode for SET TX POWER...");
   } //This is the end of the try structure
   catch (Exception e){} //This is the end of the catch structure

   //Get the response from the m5e reader
   unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
   System.out.println("Got response from RFID reader...");

   // Set Antenna Port (one-port configuration for port 1) FF 02 91 01 01
   try
   {

outputstreamToSerialInterface.write(OPCODE_BYTES_FOR_SET_ANTENNA_
PORT);
     outputstreamToSerialInterface.flush();
     System.out.println("Wrote opcode for SET ANTENNA PORT...");
   } //This is the end of the try structure
   catch (Exception e){} //This is the end of the try structure

   // Get the response from the m5e reader
   unsignedReadBuffer =
rfidReader.Access.GetDataFromReader(inputstreamFromSerialInterface,
serialInterfaceForRFID);
   System.out.println("Got response from RFID reader...");

   try
   {
     outputstreamToSerialInterface.close();
     System.out.println("Closed output stream to serial interface...");
     inputstreamFromSerialInterface.close();
     System.out.println("Closed input stream from serial interface...");
   } //This is the end of the try structure
```

```
    catch (Exception e){} //This is the end of the try structure

    serialInterfaceForRFID.close();
    System.out.println("Closed serial interface...");
  } //This is the end of method ConfigureRFIDReader()

  private static void CommenceTagReading(){
        System.out.println("Starting thread for tag reading...");
    //new Thread(tagReadingProcedure).start();
    tagReader = new TagReading(serialInterfaceID, serialInterfaceForRFID,
OPCODE_BYTES_FOR_READ_MULTIPLE_TAGS,
                OPCODE_BYTES_FOR_GET_ID_TAG_BUFFER_ONE_TAG,
OPCODE_BYTES_FOR_CLEAR_TAG_ID_BUFFER,
PATH_TO_TAG_ID_FILE);
    tagReader.start();
  } //This is the end of method CommenceTagReading()

/*
  private static void WriteTagIDToTextFile(int[] arrayOfTagIDs)
  {
        int markerOfCurrentTagID = 0;
        int numberOfTags = arrayOfTagIDs.length;
        String tagIDAsString = "";
    try
    {
          outputFileOfTagIDs = new FileWriter(PATH_TO_TAG_ID_FILE);
          bufferedwriterWritingStreamToOutputFile = new
BufferedWriter(outputFileOfTagIDs);
          for(markerOfCurrentTagID = 0; markerOfCurrentTagID < numberOfTags;
markerOfCurrentTagID++)
          {
            tagIDAsString =
Integer.toHexString(arrayOfTagIDs[markerOfCurrentTagID]);
            System.out.println("Current tag id being written is " + tagIDAsString);
            bufferedwriterWritingStreamToOutputFile.write(tagIDAsString);
          } //This is the end of the for structure
          bufferedwriterWritingStreamToOutputFile.close();
    } //This is the end of the try structure
        catch(IOException exceptionWhileAttemptingToWriteToFile)
        {} //This is the end of the catch structure
  } //This is the end of method
*/

  public static void main(String[] args)
  {
        System.out.println("Starting up RFID System...");
```

```
        ManipulateInitializationFile();
    RetrieveAndSetParameterValues();
    PrepareSerialInterface();
    PrepareRFIDReader();
        //AccessAndConfigureSystemClock();
    CommenceTagReading();
  } //This is the end of method main()

} //This is the end of the class definition
```

Appendix 17: Device Agent, Database Master Source Code Listing

```
package Package_DeviceAgent;

//Import statements
import Package_AbstractComponents.*;
import Package_DeviceAgent.*;
import java.io.Serializable;
import java.util.*;

public class PackageDeviceAgent_DatabaseMaster extends
PackageAbstractComponents_AbstractDatabaseMaster implements Serializable
{

 /************************/
 /* variable declarations */
 /************************/

 private int VersionNumber;


 /************************/
 /* Class Constructor     */
 /************************/
 public PackageDeviceAgent_DatabaseMaster()
 {
        VersionNumber = 1;
        this.SetOwner("Device Agent");
 } //This is the end of the method definition

 /*************/
 /* accessors */
 /*************/

 public int GetVersionNumber()
 {
        return VersionNumber;
 } //This is the end of the method definition

 /***********************************/
 /* software transfer functionality */
 /***********************************/

 public static void TestIfSoftwareHasBeenTransferredSuccessfully()
```

```
  {
        System.out.println("SYSTEM INFORMATION: The following class,
PackageDeviceAgent_DatabaseMaster, has been loaded successfully after the
transfer of byte codes.");
  } //This is the end of the method definition
} //This is the end of the class definition
```

Appendix 18: Device Agent Source Code Listing

```
package Package_DeviceAgent;

import java.io.Serializable;
import Package_AbstractComponents.*;
import Package_AlgorithmsAgent.PackageAlgorithmsAgent_DatabaseMaster;
import Package_ExecutablePackage.HolonicTechnologyPlatformMaster;

public class PackageDeviceAgent_DeviceAgent extends
PackageAbstractComponents_AbstractAgent implements Serializable
{
  /*************************/
  /* variable declarations */
  /*************************/

  //Variables related to HTP
  private int VersionNumber;
  private PackageDeviceAgent_DatabaseMaster DatabaseMaster;
  private HolonicTechnologyPlatformMaster HTPMaster;

  /**********************/
  /* Class Constructor */
  /**********************/
  public PackageDeviceAgent_DeviceAgent(HolonicTechnologyPlatformMaster
InputHTPMaster)
  {
    VersionNumber = 1;
          DatabaseMaster = new PackageDeviceAgent_DatabaseMaster();
          HTPMaster =  InputHTPMaster;
          this.SetAgentName("Device Agent");
  } //This is the end of the method definition

  /*************/
  /* Accessors */
  /*************/

  public PackageDeviceAgent_DatabaseMaster GetDatabaseMaster()
  {
                    return DatabaseMaster;
  } //This is the end of the method definition
/*
  public static void main(String[] args)
  {
```

```
    //PackageDeviceAgent_DeviceAgent deviceAgentForRFIDReader = new
PackageDeviceAgent_DeviceAgent(HTPMaster);
  } //This is the end of the method definition
*/

  /***********************************/
  /* Software transfer functionality */
  /***********************************/

  public static void TestIfSoftwareHasBeenTransferredSuccessfully()
  {
        System.out.println("SYSTEM INFORMATION: The following class,
PackageDeviceAgent_DeviceAgent, has been loaded successfully after the
transfer of byte codes.");
  } //This is the end of the method definition
} //This is the end of the class definition
```

Appendix 19: Device Agent Run DIS Event Source Code Listing

```
package Package_DeviceAgent;

//Import statements
import java.util.EventObject;
import java.io.Serializable;
import java.net.*;
import Package_ExecutablePackage.*;

public class PackageDeviceAgent_RunDISCEvent extends EventObject
implements Serializable
{
  /************************/
  /* variable declarations */
  /************************/

  private HolonicTechnologyPlatformMaster HolonicTechnologyPlatformMaster;
  private static String className = "PackageDeviceAgent_RunDISCEvent";

  /****************/
  /* constructors */
  /****************/

  public PackageDeviceAgent_RunDISCEvent(Object source,
HolonicTechnologyPlatformMaster InputHolonicTechnologyPlatformMaster)
  {
        super(source);
        HolonicTechnologyPlatformMaster =
InputHolonicTechnologyPlatformMaster;
  } //This is the end of the method definition

  /********************/
  /* accessor methods */
  /********************/

  public HolonicTechnologyPlatformMaster
GetHolonicTechnologyPlatformMaster()
  {
        return HolonicTechnologyPlatformMaster;
  } //This is the end of the method definition

  /**********************************/
  /* software transfer functionality */
  /**********************************/
```

```
  public static void TestIfSoftwareHasBeenTransferredSuccessfully()
  {
        System.out.println("SYSTEM INFORMATION: The following class" +
className + ", has been loaded successfully after the transfer of byte codes.");
  } //This is the end of the method definition

} //This is the end of the class definition
```

Appendix 20: Device Agent Run DIS Event Handler Source Code Listing

```
package Package_DeviceAgent;

import java.io.*;
import java.net.DatagramSocket;
import Package_AbstractComponents.*;
import Package_DeviceAgent.*;
import Package_ExecutablePackage.HolonicTechnologyPlatformMaster;
import Package_ToolsAndUtilities.*;

public class PackageDeviceAgent_RunDISCEventHandler implements
PackageDeviceAgent_RunDISCEventListener, Serializable
{
  /*************************/
  /* variable declarations */
  /*************************/
  //Variables related to configuration file manipulation
  public static String PATH_TO_AGENT_CONFIGURATION_FILE =
"/DISC/HTP/Package_DeviceAgent/DeviceAgentConfiguration.txt";
  private static int SIZE_OF_CONFIGURATION_FILE_IN_LINES = 30;
  private static Package_ToolsAndUtilities.PackageToolsAndUtilities_FileHandler
handlerOfAgentConfigurationFile;
  private static Package_ToolsAndUtilities.PackageToolsAndUtilities_FileParser
parserOfAgentConfigurationFile;
  private static
Package_ToolsAndUtilities.PackageToolsAndUtilities_FileParameterRetrieval
retrievalOfAgentConfigurationParameters;
  private static
Package_ToolsAndUtilities.PackageToolsAndUtilities_CheckTimestamp
inspectorOfNewMessages;
  //Variables related to Device Agent Identity
  private static String NAME_OF_PARENT_HOLONIC_UNIT = "";
  private static String DEVICE_AGENT_ID = "";
  private static int DEVICE_AGENT_IDENTIFICATION_NUMBER;
  //Variables related to Thread Handling
  private static Thread CheckIncomingMessageFolderForNewMessages;
  //Variables related to Message Handling
  public static String LOCATION_OF_MOTION_SENSED_MESSAGE = "";
  public static String LOCATION_OF_TAG_READING_MESSAGE = "";
  public static String LOCATION_OF_TAKE_SNAPSHOT_MESSAGE = "";
  public static String PRIMARY_INCOMING_MESSAGE_TO_MONITOR = "";
  //Variables related to File Monitoring
  private static long oldTimestampOnFile;
```

```
  private static long timestampOnModifiedFile;
  File fileHandleOnMessageFileBeingMonitored;
  //Variables related to Shell Script Execution
  PackageToolsAndUtilities_RunShellCommands executionOfShellScript = new
PackageToolsAndUtilities_RunShellCommands();
  public static String
SYSTEM_COMMAND_FOR_OPERATING_SYSTEM_SHELL = "";
  public static String LOCATION_OF_MAIN_SCRIPT_DIRECTORY = "";
  public static String
SCRIPT_HANDLING_FILE_TRANSFER_TO_MOTION_SENSOR_HOLONIC_U
NIT = "";
  public static String
SCRIPT_HANDLING_FILE_TRANSFER_TO_NETWORK_CAMERA_HOLONIC
_UNIT = "";
  public static String
SCRIPT_HANDLING_FILE_TRANSFER_TO_RFID_READER_HOLONIC_UNIT
= "";
  public static String
SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC_UNIT =
"";
  //Variables related to behaviour
  public static String FIRST_COURSE_OF_ACTION = "";
  public static String SECOND_COURSE_OF_ACTION = "";

  /****************/
  /* event method */
  /****************/
  public void SayHelloEventReceived(PackageDeviceAgent_SayHelloEvent
Event)
  {
    System.out.println("This is Device Agent, confirmed working.");
    System.out.println("Starting up DISC now.");
    ManipulateAgentConfigurationFile();
    AssignConfigurationParameters();
    CheckIncomingMessageFolderForNewMessages = new
Thread(RunThreadCheckingIncomingMessageFolder, "DISC Incoming Message
Monitoring Thread");
    CheckIncomingMessageFolderForNewMessages.start();
    while(true)
    {
      try
      {
        CheckIncomingMessageFolderForNewMessages.sleep(100);
      } //This is the end of the try structure
      catch(InterruptedException
exceptionThrownWhileMonitoringIncomingMessages)
```

```
    {
        CheckIncomingMessageFolderForNewMessages.run();
    } //This is the end of the catch structure
  } //This is the end of the while structure
 } //This is the end of the method definition

 /******************/
 /* utility methods */
 /******************/
 public void ManipulateAgentConfigurationFile()
 {
   /*
         * Creates new object to handle the Agent Configuration file
         * Sets the path to the location of the Agent configuration parameter file
         * Assigns the data structure responsible for containing the complete
contents of the file
    */
        System.out.println("Opening and reading Initialization File...");
        handlerOfAgentConfigurationFile = new
Package_ToolsAndUtilities.PackageToolsAndUtilities_FileHandler();
        PackageToolsAndUtilities_FileHandler.WORKING_PATH_DIRECTORY =
PATH_TO_AGENT_CONFIGURATION_FILE;
        PackageToolsAndUtilities_FileHandler.SIZE_OF_INITIALIZATION_FILE_I
N_LINES = SIZE_OF_CONFIGURATION_FILE_IN_LINES;

        /*
         * Opens the Agent configuration file, parses it and closes it once done
         */
        handlerOfAgentConfigurationFile.OpenFile();
        handlerOfAgentConfigurationFile.ParseFileIntoLines();
        handlerOfAgentConfigurationFile.CloseFile();

        /*
         * Creates a new object to parse the agent configuration file.
         * It is provided with the path to the location of the agent configuration file.
         * Assigns the data structures that will hold the main sections, subsections
and parameters
         * of the parameter file.
         */
        System.out.println("Parsing Agent Configuration File...");
        parserOfAgentConfigurationFile = new
Package_ToolsAndUtilities.PackageToolsAndUtilities_FileParser();
        PackageToolsAndUtilities_FileParser.LOCATION_OF_TARGET_FILE =
PATH_TO_AGENT_CONFIGURATION_FILE;
        PackageToolsAndUtilities_FileParser.SIZE_OF_INITIALIZATION_FILE_I
N_LINES = SIZE_OF_CONFIGURATION_FILE_IN_LINES;
```

```
        /*
         * Extracts Main Sections, Subsections and Parameters from RFID
initialization file
         */
        parserOfAgentConfigurationFile.InitializeStructuresForExtractionProcedur
e();
        parserOfAgentConfigurationFile.ExtractMainSectionsFromTargetFile(hand
lerOfAgentConfigurationFile.contentsOfFile);
        System.out.println("Main Sections Extracted From Initialization File...");
        parserOfAgentConfigurationFile.ExtractSubsectionsFromTargetFile(handl
erOfAgentConfigurationFile.contentsOfFile);
        System.out.println("Subsections Extracted From Initialization File...");
        parserOfAgentConfigurationFile.ExtractParametersFromTargetFile(handle
rOfAgentConfigurationFile.contentsOfFile);
        System.out.println("Parameters Extracted From Initialization File...");
        //parserOfAgentConfigurationFile.ShowRequestedSectionExtractedOK(pa
rserOfAgentConfigurationFile.InitializationFileParameters);
 } //This is the end of the method definition


 public void AssignConfigurationParameters()
 {
  retrievalOfAgentConfigurationParameters = new
Package_ToolsAndUtilities.PackageToolsAndUtilities_FileParameterRetrieval();

  //Retrieve and set Agent Parameters
        System.out.println("Retrieving Device Agent Identity Parameters...");
        NAME_OF_PARENT_HOLONIC_UNIT =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,
                                "PARENT_HOLONIC_UNIT");
        DEVICE_AGENT_ID =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,
                                "DEVICE_AGENT_NAME");
        DEVICE_AGENT_IDENTIFICATION_NUMBER =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAsI
nteger(parserOfAgentConfigurationFile.InitializationFileParameters,
                                "DEVICE_AGENT_IDENTIFICATION_NUMBER");

        System.out.println("Retrieving Message Location Parameters...");
        LOCATION_OF_MOTION_SENSED_MESSAGE =
```

```
        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,
                                "PATH_TO_MOTION_SENSED_MESSAGE");
        LOCATION_OF_TAG_READING_MESSAGE =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,
                                "PATH_TO_TAG_READING_MESSAGE");
        LOCATION_OF_TAKE_SNAPSHOT_MESSAGE =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,
                                "PATH_TO_TAKE_SNAPSHOT_MESSAGE");

        System.out.println("Retrieving Script Location Parameters...");
        LOCATION_OF_MAIN_SCRIPT_DIRECTORY =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,
                        "PATH_TO_MAIN_SCRIPT_DIRECTORY");
        SCRIPT_HANDLING_FILE_TRANSFER_TO_MOTION_SENSOR_HOLO
NIC_UNIT =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,

        "SCRIPT_HANDLING_FILE_TRANSFER_TO_MOTION_SENSOR_HOL
ONIC_UNIT");
        SCRIPT_HANDLING_FILE_TRANSFER_TO_NETWORK_CAMERA_HO
LONIC_UNIT =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,

        "SCRIPT_HANDLING_FILE_TRANSFER_TO_NETWORK_CAMERA_HO
LONIC_UNIT");
        SCRIPT_HANDLING_FILE_TRANSFER_TO_RFID_READER_HOLONIC
_UNIT =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,

        "SCRIPT_HANDLING_FILE_TRANSFER_TO_RFID_READER_HOLONI
C_UNIT");
```

```
        SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC_
UNIT =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,

        "SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC
_UNIT");

        System.out.println("Retrieving Operating System Specifications...");
        SYSTEM_COMMAND_FOR_OPERATING_SYSTEM_SHELL =

        retrievalOfAgentConfigurationParameters.ExtractRequestedParameterAs
String(parserOfAgentConfigurationFile.InitializationFileParameters,

        "SYSTEM_COMMAND_FOR_OPERATING_SYSTEM_SHELL");

        //Assign variable values depending on identity of Device Agent
        switch(DEVICE_AGENT_IDENTIFICATION_NUMBER)
        {
          case 1:
            PRIMARY_INCOMING_MESSAGE_TO_MONITOR =
LOCATION_OF_MOTION_SENSED_MESSAGE;
            FIRST_COURSE_OF_ACTION =
SCRIPT_HANDLING_FILE_TRANSFER_TO_RFID_READER_HOLONIC_UNIT;
            SECOND_COURSE_OF_ACTION =
SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC_UNIT;
            break;
          case 2:
            PRIMARY_INCOMING_MESSAGE_TO_MONITOR =
LOCATION_OF_TAG_READING_MESSAGE;
                FIRST_COURSE_OF_ACTION =
SCRIPT_HANDLING_FILE_TRANSFER_TO_NETWORK_CAMERA_HOLONIC
_UNIT;
                SECOND_COURSE_OF_ACTION =
SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC_UNIT;
                break;
          case 3:
                PRIMARY_INCOMING_MESSAGE_TO_MONITOR =
LOCATION_OF_TAKE_SNAPSHOT_MESSAGE;
                FIRST_COURSE_OF_ACTION =
SCRIPT_HANDLING_FILE_TRANSFER_TO_SUPERVISOR_HOLONIC_UNIT;
                break;
        } //This is the end of the switch structure
  } //This is the end of the method definition
```

```
 public void CreateHandleForFileMonitoring(String specifiedFileToMonitor)
 {
  boolean indicatorOfSuccessfulCreation;

        //Create new File class with abstract path handler
        File fileHandleOnMessageFileBeingMonitored = new
File(specifiedFileToMonitor);
        //Attempt to create new file in given path location
        try
        {
         indicatorOfSuccessfulCreation =
fileHandleOnMessageFileBeingMonitored.createNewFile();
         if(indicatorOfSuccessfulCreation == true)
         {
          System.out.println("File was successfully created.");
         } //This is the end of the if structure
         else if (indicatorOfSuccessfulCreation == false)
         {
                System.out.println("File already exists.");
         }
        } //This is the end of the try structure
        catch(IOException exceptionThrownWhileCreatingFile)
        {
                System.out.println("Exception: I/O error occurred.");
        } //This is the end of the catch structure
        //Read the timestamp on the created file
        oldTimestampOnFile =
fileHandleOnMessageFileBeingMonitored.lastModified();
        System.out.println("Timestamp of initially created file in " +
specifiedFileToMonitor + " is: " + oldTimestampOnFile);
 } //This is the end of the method definition

 public void MonitorFileAndTakeAction(String specifiedFileToMonitor)
 {
        File targetFileToInspect = new File(specifiedFileToMonitor);
        while(true)
        {
         //Check timestamp value
         timestampOnModifiedFile = targetFileToInspect.lastModified();
         //Has the timestamp value been modified?
         if(timestampOnModifiedFile != oldTimestampOnFile)
         {
          System.out.println("File has been modified. New timestamp of modified
file is: " + timestampOnModifiedFile);
           //Update last recorded timestamp value
           oldTimestampOnFile = targetFileToInspect.lastModified();
```

```
            //Take decision
            //Call function to perform file transfer
            System.out.println("Calling script to perform file transfer.");
            PerformFileTransfer(FIRST_COURSE_OF_ACTION);
      //PerformFileTransfer(SECOND_COURSE_OF_ACTION);
            //Continue checking timestamp value for modification
            continue;
          } //This is the end of the if structure
        } //This is the end of the while structure
  } //This is the end of the method definition

  public void PerformFileTransfer(String specifiedScriptHandlingFileTransfer)
  {
    executionOfShellScript.specificationOfWorkingDirectory =
LOCATION_OF_MAIN_SCRIPT_DIRECTORY;
        executionOfShellScript.systemCommandToExecute =
SYSTEM_COMMAND_FOR_OPERATING_SYSTEM_SHELL;
        executionOfShellScript.PerformSetupToRunShellCommands();
        executionOfShellScript.ExecuteShellCommands(specifiedScriptHandlingFi
leTransfer);
  } //This is the end of the method definition

  Runnable RunThreadCheckingIncomingMessageFolder = new Runnable()
  {
    public void run()
    {
      System.out.println("Starting thread to check incoming message folder for new
messages");

CreateHandleForFileMonitoring(PRIMARY_INCOMING_MESSAGE_TO_MONIT
OR);

MonitorFileAndTakeAction(PRIMARY_INCOMING_MESSAGE_TO_MONITOR);
    } //This is the end of the method definition
  }; //This is the end of the Thread definition

  /**********************************/
  /* software transfer functionality */
  /**********************************/
  public static void TestIfSoftwareHasBeenTransferredSuccessfully()
  {
        System.out.println("SYSTEM INFORMATION: The following class,
PackageDevelopmentAgent_CreateNewAgentCodeUsingTemplateEventHandler
, has been loaded successfully after the transfer of byte codes.");
  } //This is the end of the method definition
} //This is the end of the method definition
```

Appendix 21: Device Agent Run DIS Event Listener Source Code Listing

```
package Package_DeviceAgent;

public interface PackageDeviceAgent_RunDISCEventListener
{
  public void RunDISCEventReceived(PackageDeviceAgent_RunDISCEvent
Event);
} //This is the end of the class definition
```

Appendix 22: Motion Sensor Signal Detection Source Code Listing

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>

//Function prototypes
int WriteDataToFile(int dataReceived);

//Global variables
FILE *fileStoringTestData;
int fileClosureError;
char nameOfTestFile[] = "MotionSensed.txt";

//Constants
int MOTION_SENSED = 1;

int main(int argc, char **argv)
{
  volatile unsigned int *PORT_E_DATA_REGISTER,
                        *PORT_E_DATA_DIRECTION_REGISTER,
                        *PORT_B_DATA_REGISTER,
*PORT_B_DATA_DIRECTION_REGISTER,
                        *GPIO_PORT_B_DATA_BIT;
  int counterToBlinkLED;
  unsigned char stateOfPortBDataRegister;
  unsigned char *startingLocationForPortBDataRegister;
  int fileDescriptorHandlingMemoryLocation = open("/dev/mem", O_RDWR);
  startingLocationForPortBDataRegister = mmap(0, getpagesize(),
PROT_READ|PROT_WRITE, MAP_SHARED,
  fileDescriptorHandlingMemoryLocation, 0x80840000);
  PORT_B_DATA_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x04); //port b
  PORT_B_DATA_DIRECTION_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x14); //port b direction
  PORT_E_DATA_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x20); //port e data
  PORT_E_DATA_DIRECTION_REGISTER = (unsigned int
*)(startingLocationForPortBDataRegister + 0x24); //port e direction
  GPIO_PORT_B_DATA_BIT = (unsigned int
*)(startingLocationForPortBDataRegister + 0xC4); // debounce on port b
```

```
  *PORT_B_DATA_DIRECTION_REGISTER = 0xf0; //upper nibble output, lower
nibble input
  *PORT_E_DATA_DIRECTION_REGISTER = 0xff; //all output (just 2 bits)
  *GPIO_PORT_B_DATA_BIT = 0x01; //enable debounce on bit 0
  stateOfPortBDataRegister = *PORT_B_DATA_REGISTER; // read initial state

  while(stateOfPortBDataRegister & 0x01)
  { // wait until button goes low
    stateOfPortBDataRegister = *PORT_B_DATA_REGISTER; // remember bit 0
is pulled up with 4.7k ohm
  } //This is the end of the while structure

  WriteDataToFile(MOTION_SENSED);

// blink 5 times, sleep 1 second so it's visible
  for (counterToBlinkLED = 0; counterToBlinkLED < 5; counterToBlinkLED++)
  {
    *PORT_E_DATA_REGISTER = 0xff;
    sleep(1);
    *PORT_E_DATA_REGISTER = 0x00;
    sleep(1);
  } //This is the end of the for structure

  close(fileDescriptorHandlingMemoryLocation);
  return 0;
} //This is the end of function main()

int WriteDataToFile(int dataReceived)
{
  fileStoringTestData = fopen(nameOfTestFile, "w");
  if(fileStoringTestData == NULL)
  {
    fprintf(stderr, "Error: Unable to write to file.\n");
  } //This is the end of the if structure
  else
  {
    int indicationOfSuccessfulWrite = fprintf(fileStoringTestData, "%d\n",
dataReceived);
    if(indicationOfSuccessfulWrite == dataReceived)
    {
      printf("Successful write.\n");
    } //This is the end of the if structure
  } //This is the end of the else structure
  fileClosureError = fclose(fileStoringTestData);
  if(fileClosureError != 0)
  {
```

```
    printf("Error: File could not be closed.\n");
  } //This is the end of the if structure
  else
  {
    printf("File closed.\n");
  } //This is the end of the else structure
  return 0;
} //This is the end of function WriteDataToFile()
```

REFERENCE LIST

[1] J. Eberspacher and R. Schollimer, "Past and future," in *Peer-to-Peer systems and applications*. Heidelberg, Germany: Springer-Verlag, 2005, pp. 18-19.

[2] W. R. Ashby, *An Introduction to Cybernetics*, 2nd ed., London: Chapman and Hall, 1967.

[3] C. François. "Systemics and cybernetics in a historical perspective," in *Systems Research and Behavioral Science*, vol 16, pp. 203-219, 1999, Available: http://www.uni-klu.ac.at/~gossimit/ifsr/francois/papers/systemics_and_cybernetics_in_a_historical_perspective.pdf [Accessed Jan 2009].

[4] T.C Helvey,. *The Age of Information: An Interdisciplinary Survey of Cybernetics*, Englewood Cliffs, N.J.: Educational Technology Publications, 1971.

[5] F. Heylighen and C. Joslyn, "Cybernetics and second order cybernetics," in *Encyclopedia of Physical Science & Technology*, 3rd ed., vol. 4, R.A. Meyers, New York: Academic Press, pp. 155-170.

[6] R. J. Maly, "Comparison of Centralized (Client-Server) and Decentralized (Peer-to-Peer) Networking," Semester thesis, ETH Zurich, Zurich, Switzerland, 2003.

[7] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 4th ed., Harlow, Essex, Pearson Education Limited, 2005.

[8] "Client-server", Jan. 28th, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Client-server. [Accessed: Feb 1st, 2010].

[9] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, New Jersey: John Wiley and Sons, 2004.

[10] V. K. Garg, *Elements of Distributed Computing*, New York, John Wiley and Sons, 2002.

[11] A. S. Tanembaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*, 2nd ed., London: Prentice Hall, 2006.

[12] K. Thomas, W. A. Gruver, D. Sabaz, and C. Ng, “A hybrid protocol architecture for peer-to-peer control based on SIP and UPnP,” in *Proceedings of the 2009 IEEE International Systems Conference*, March 23-26, 2009, Vancouver, Canada. New York, NY: IEEE Systems Council, 2009.

[13] D. Trentesaux, "Distributed control of production systems," in *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 971-978, Oct. 2009.

[14] I. Seilonen, Ã. T. Pirttioja, and K. Koskinen, "Extending process automation systems with multi-agent techniques," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 1056-1067, Oct. 2009.

[15] D. Schoder, K. Fischbach, and C. Schmitt, "Application Areas" in *Peer-to-Peer Systems and Applications*. Heidelberg, Germany: Springer-Verlag, 2005, pp. 25 – 32.

[16] R. J. Maly, "Comparison of Centralized (Client-Server) and Decentralized (Peer-to-Peer) Networking," Semester thesis, ETH Zurich, Zurich, Switzerland, 2003, p. 8.

[17] R. Miller, "Facebook Now Running 10,000 Web Servers," Data Center Knowledge, Apr, 23, 2008. Available: Data Center Knowledge, http://datacenterknowledge.com. [Accessed Jan 31, 2009].

[18] M. Ahmed, "Google search finds seafaring solution," The Times, Sept 15, 2008. Available: The Times Online, http://timesonline.co.uk. [Accessed Jan 31, 2009].

[19] J. R. Leigh, *Applied Control Theory*, 2nd ed., London: Peter Peregrinus Ltd, 1987.

[20] S. C. Sugarman, *HVAC Fundamentals*, Liburn: The Fairmont Press, 2004.

[21] "Direct Digital Control", Oct. 5th, 2009. [Online]. Available: http://en.wikipedia.org/wiki/Direct_Digital_Control. [Accessed: Feb 1st, 2010].

[22] "Programmable Logic Controller", Feb 1st, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Programmable_logic_controllers. [Accessed: Apr 23rd, 2010].

[23] T. M. Stout and T. J. Williams. "Pioneering Work in the Field of Computer Process Control" IEEE Annals of the History of Computing, 1995.

[24] JRD TATA Automation Training Centre, *Distributed Control Systems and Applications in Control Industry for Industrial Automation and Control Engineers*. Pune, India: TATA Honeywell, 2005.

[25] "Distributed Control System", Feb. 2nd, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Distributed_Control_System. [Accessed: Feb 2nd, 2010].

[26] The Modbus Organization, “Modbus Technical Resources,” The Modbus Organization [Online]. Available: http://www.modbus.org/. [Accessed: Feb. 2, 2010].

[27] BACNet, “Tutorials,” BACNet [Online]. Available: http://www.bacnet.org/. [Accessed: Feb. 2, 2010].

[28] "BACNet", Jan. 23rd, 2010. [Online]. Available: http://en.wikipedia.org/wiki/BACNet. [Accessed: Feb 1st, 2010].

[29] "Profibus PA System Description” Aug, 2007. [Online]. Available: http://www.profibus.com/nc/downloads/downloads/profibus-pa-technology-and-application-system-description/download/191/. [Accessed: Feb 1st, 2010].

[30] “Technology,” CAN in Automation [Online]. Available: http://www.can-cia.de/. [Accessed: Feb. 2, 2010].

[31] R. Bosch GmbH. *CAN Specification 2.0*, Stuttgart, Germany: Robert Bosch GmbH, 1991. [E-book] Available: Robert Bosch GmbH Automotive Semiconductors and Sensors.

[32] "Controller Area Network", Feb. 2nd, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Controller_Area_Network. [Accessed: Feb 2nd, 2010]

[33] “DeviceNet Technology Overview,” Open DeviceNet Vendor Association [Online]. Available: http://www.odva.org/. [Accessed: Feb. 2, 2010]

[34] “DeviceNet Technology Overview,” 2004. [Online]. Available: http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00026R1.pdf. [Accessed: Feb. 2, 2010].

[35] “Welcome to the UPnP Forum!,” Dec. 31, 2009. [Online]. Available: http://www.upnp.org/. [Accessed: Feb. 3, 2010].

[36] UPnP Forum, UPnP Device Architecture version 1.1, 2008.

[37] UPnP Forum, UPnP Vendor's Implementation Guide, 2001.

[38] UPnP Forum, UPnP DeviceType:V Device Template Version 1.01, 2001.

[39] UPnP Forum, UPnP ServiceType:V Service Template Version 2.00, 2008.

[40] V. Gupta, et al., *Peer-to-Peer Application Development: Cracking the Code*, New York: USA: Hungry Minds, 2002, pp. 1-4.

[41] I. Taylor and A. Harrison, *From P2P and Grids to Services on the Web: Evolving distributed communities*, 2nd ed., London, UK:, Springer-Verlag, 2009.

[42] J. F. Buford, H. Yu and E. K. Lua, *P2P Networking and Applications*. Burlington, MA, USA: Morgan Kaufman, 2009, pp. 8-12.

[43] K. Wehrle, et al., *Peer-to-Peer Systems and Applications*. Heidelberg, Germany: Springer-Verlag, 2005, p. 1.

[44] R. Steinmetz and K. Werhle, "What is this peer-to-peer about?" in *Peer-to-Peer Systems and Applications*, Heidelberg, Germany: Springer-Verlag, 2005, pp. 10-15.

[45] "BitTorrent", 2010 [Online]. Available: http://www.bittorrent.com/. [Accessed: Feb. 3, 2010].

[46] "BitTorrent (software)", Feb. 2nd, 2010. [Online]. Available: http://en.wikipedia.org/wiki/BitTorrent_(software). [Accessed: Feb 6th, 2010].

[47] "Skype", 2010 [Online]. Available: http://www.skype.com/getconnected/. [Accessed: Feb. 3, 2010].

[48] TorrentFreak, "Mininova, 5 billion downloads and counting," TorrentFreak, May 26, 2008. [Online]. Available: http://torrentfreak.com/mininova-5-billion-downloads-and-counting-080526/. [Accessed Feb. 6, 2010].

[49] TorrentFreak, "BitTorrent trio hit a billion pageviews a month," TorrentFreak, June 11, 2008. [Online]. Available: http://torrentfreak.com/bittorrent-trio-hit-a-billion-pageviews-a-month-080611/. [Accessed Feb. 6, 2010].

[50] TorrentFreak, "Survey shows huge demand for legal P2P," TorrentFreak, June 16, 2008. [Online]. Available: http://torrentfreak.com/survey-shows-huge-demand-for-legal-p2p-080616/. [Accessed Feb. 6, 2010].

[51] TorrentFreak, "EA choose BitTorrent for warhammer online distribution," TorrentFreak, August 13, 2008. [Online]. Available: http://torrentfreak.com/ea-choose-bittorrent-for-warhammer-online-distribution-080813/. [Accessed Feb. 6, 2010].

[52] TorrentFreak, "BitTorrent searches skyrocket as sites grow," TorrentFreak, September 01, 2008. [Online]. Available: http://torrentfreak.com/bittorrent-searches-skyrocket-as-sites-grow-080901/. [Accessed Feb. 6, 2010].

[53] TorrentFreak, "The pirate bay tops 15 million peers," TorrentFreak, September 15, 2008. [Online]. Available: http://torrentfreak.com/the-pirate-bay-tops-15-million-users-080921/. [Accessed Feb. 6, 2010].

[54] TorrentFreak, "Mininova breaks download records," TorrentFreak, September 23, 2008. [Online]. Available: http://torrentfreak.com/mininova-breaks-download-records-080923/. [Accessed Feb. 6, 2010].

[55] TorrentFreak, "'Shocking' 61% of all upstream internet traffic is P2P," TorrentFreak, October 21, 2008. [Online]. Available: http://torrentfreak.com/shocking-61-of-all-upstream-internet-traffic-is-p2p-081021/. [Accessed Feb. 6, 2010].

[56] TorrentFreak, "Dutch university uses bittorrent to update workstations," TorrentFreak, March 6, 2008. [Online]. Available: http http://torrentfreak.com/university-uses-utorrent-080306/. [Accessed Feb. 6, 2010].

[57] TorrentFreak, "Mininova to launch bitTorrent video streaming," TorrentFreak, March 19, 2008. [Online]. Available: http://torrentfreak.com/mininova-bittorrent-video-streaming-080319/. [Accessed Feb. 6, 2010].

[58] TorrentFreak, "Download torrents on PS3, iPhone and web-enabled devices," TorrentFreak, June 10, 2008. [Online]. Available: http://torrentfreak.com/download-torrents-on-ps3-iphone-and-other-web-enabled-devices-080610/. [Accessed Feb. 6, 2010].

[59] TorrentFreak, "P2P-Next introduces live bittorrent streaming," TorrentFreak, July 18, 2008. [Online]. Available: http://torrentfreak.com/p2p-next-introduces-live-bittorrent-streaming-080718/. [Accessed Feb. 6, 2010].

[60] B. Jones, "EZTV trials tv-torrent streaming," TorrentFreak, July 26, 2008. [Online]. Available: http://torrentfreak.com/eztv-trials-streaming-080726/. [Accessed Feb. 6, 2010].

[61] TorrentFreak, "Pioneer's live bittorrent streaming device," TorrentFreak, September 11, 2008. [Online]. Available http://torrentfreak.com/pioneers-live-bittorrent-streaming-device-080911/. [Accessed Feb. 6, 2010].

[62] TorrentFreak, "BitTorrent to speed up game distribution," TorrentFreak, September 15, 2008. [Online]. Available: http://torrentfreak.com/bittorrent-to-speed-up-game-distribution-080915/. [Accessed Feb. 6, 2010].

[63] TorrentFreak, "DistriBrute: P2P powered desktop deployment," TorrentFreak, October 16, 2008. [Online]. Available: http://torrentfreak.com/distribrute-p2p-powered-desktop-deployment-081016/. [Accessed Feb. 6, 2010].

[64] "DistriBrute," 2009. [Online]. Available: http://www.4m88.com/index.php/Products/products.html. [Accessed: Feb. 6, 2010].

[65] TorrentFreak, "Stanford university embraces bittorrent," TorrentFreak, October 18, 2008. [Online]. Available: http://torrentfreak.com/stanford-university-embraces-bittorrent-081018/. [Accessed Feb. 6, 2010].

[66] TorrentFreak, "Tribler set to make bittorrent sites obsolete," TorrentFreak, October 28, 2008. [Online]. Available: http://torrentfreak.com/tribler-set-to-make-bittorrent-sites-obsolete-081028/. [Accessed Feb. 6, 2010].

[67] "Tribler," 2009. [Online]. Available: http://www.tribler.org/trac/wiki. [Accessed: Feb. 6, 2010].

[68] TorrentFreak, "BBC trials bittorrent powered HD video streaming," TorrentFreak, December 03, 2009. [Online]. Available: http://torrentfreak.com/bbc-trials-bittorrent-powered-hd-video-streaming-091203/. [Accessed Feb. 6, 2010].

[69] TorrentFreak, "BitTorrent powered TV is coming," TorrentFreak, April 14, 2009. [Online]. Available: http://torrentfreak.com/bittorrent-powered-tv-is-coming-090414/. [Accessed Feb. 6, 2010].

[70] W.A. Gruver and D. Sabaz, "Distributed intelligent systems: What makes them

'intelligent'", in *Proceedings of IEEE Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications,* Beijing, China, 2005.

[71] R. Steeb, et al.., "Distributed Intelligence for Air Fleet Control", Santa Monica, CA: The Rand Corporation, 1981.

[72] J. C. Augusto, et al.., *Advances in Ambient Intelligence*, Amsterdam, Netherlands: IOS Press, 2007.

[73] R. Razavi, K. Mechitov, G. Agha and J. Perrot, "Ambiance: a mobile agent platform for end-user programmable ambient systems" in *Advances in Ambient Intelligence*, Amsterdam, Netherlands: IOS Press, 2007. pp. 81 - 106.

[74] B. Ganter, et al., *Formal Concept Analysis: Foundations and applications*, Heidelberg, Germany: Springer-Verlag, 2005.

[75] C. Ng, "Framework for developing Distributed Intelligence Systems in a peer-to-peer environment," MASc. Thesis, School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada, 2007, pp. 59 – 62, pp. 109 – 115, pp. 72 – 99, pp. 50 – 51, pp. 62 – 64, p. 72, pp. 88 - 93.

[76] Dictionary.com, "architecture," in Dictionary.com Unabridged. Source location: Random House, Inc. http://dictionary.reference.com/browse/architecture. Available: http://dictionary.reference.com. Accessed: November 23, 2009.

[77] Wikipedia.org, "Centralized system," in Wikipedia.org. http://en.wikipedia.org/wiki/Centralized_system. Available: http://en.wikipedia.org/wiki/. Accessed: November 23, 2009.

[78] R. J. Maly, "Comparison of Centralized (Client-Server) and Decentralized (Peer-to-Peer) Networking," Semester thesis, ETH Zurich, Zurich, Switzerland, 2003, pp. 1 – 12.

[79] D. G. Luenberger, *Optimization by Vector Space Methods*. New York, USA: John Wiley and Sons, 1969.

[80] Dictionary.com, "node," in The Free On-line Dictionary of Computing. Source location: Denis Howe. http://dictionary.reference.com/browse/node. Available: http://dictionary.reference.com. Accessed: November 28, 2009.

[81] Dictionary.com, "peer," in The Free On-line Dictionary of Computing. Source location: Denis Howe. http://dictionary.reference.com/browse/peer. Available: http://dictionary.reference.com. Accessed: November 28, 2009.

[82] A. Koestler, *Janus: A Summing Up*, Random House, 1978.

[83] C. Nikolai, and G. Madey, "Tools of the trade: a survey of various agent based modeling platforms", in *Journal of Artificial Societies and Social Simulation*, 2008.

[84] R.J. Allan, "Survey of agent based modelling and simulation tools", STFC Daresbury Laboratory, Warrington, UK, 2009.

[85] Jade - Java Agent DEvelopment Framework Available: http://jade.tilab.com/. Accessed: November 28, 2009.

[86] A. Helsinger, M. Thome, and T. Wright, "Cougaar: a scalable, distributed multi-agent architecture", in *IEEE International Conference on Systems, Man, and Cybernetics Conference*, 2004.

[87] BBN Technologies Development Staff, "Cougaar Architecture Document", BBN Technologies, 2004.

[88] jxta: JXTA Community Projects, Available: https://jxta.dev.java.net/. [Accessed: Nov. 28, 2009].

[89] F. Bellifemine, et al., "JADE: A White Paper," Telecom Italia Laboratory, Torino, Italy, 2003.

[90] Sun Microsystems, JXTA Java Standard Edition v. 2.5: Programmers Guide, Sun Microsystems, 2007.

[91] “iDEA Lab”. [Online]. Available: http://www2.ensc.sfu.ca/idea/. [Accessed: Feb. 12, 2010].

[92] T. Beiser, et al., "Requirements Engineering" in *Multiagent Engineering*. Heidelberg, Germany: Springer-Verlag, 2006, p. 364.

[93] G. Weiss, Multiagent Systems, Cambridge, MA: MIT Press, 2000.

[94] M. Wooldridge, “Intelligent Agents,” in *Multiagent Systems*, Cambridge, MA: MIT Press, 2000, p. 49.

[95] S. Sen and G. Weiss, “Learning in Multiagent Systems,” in *Multiagent Systems*, Cambridge, MA: MIT Press, 2000, p. 269.

[96] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and design*, 4th ed., Harlow, Essex: Pearson Education Limited, 2005, pp. 177 – 220, pp. 201-207.

[97] "Agent-Based Models of Industrial Ecosystems" [Online], Available: http://policy.rutgers.edu/andrews/projects/abm/default.htm. [Accessed: Dec. 11, 2009].

[98] M. N. Huhns and L. M. Stephens, "Multiagent Systems and Societies of Agents" in *Multiagent Systems*, Cambridge, MA: MIT Press, 2000, pp. 79 – 120, pp. 83 - 84.

[99] G. Grätzer, *Lattice Theory: First concepts and distributive lattices*, San Francisco, CA : W. H. Freeman and Co, 1971.

[100] J. G. Hocking and G. S. Young, *Topology*, Reading, MA: Addison-Wesley, 1961.

[101] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, 2nd ed., Cambridge, UK: Cambridge University Press, 2001, p. 95.

[102] B. Ganter et al., *Formal Concept Analysis: Foundations and applications*, Heidelberg, Germany: Springer-Verlag, 2005, p. 43.

[103] B. S. W. Schröder, *Ordered Sets: An introduction*, Boston, MA: Birkhäuser, 2003.

[104] H. A. Priestly, B. A. Davey, *Introduction to Lattices and Order*, Cambridge, UK: Cambridge University Press, 1990.

[105] S. Ovcharenko, "Mobile agent hardware design for distributed wireless networks," M.Eng. Report, School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada, 2006.

[106] IEEE, “IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary-Scan Architecture -Description,” IEEE, [Online]. Available: http://http://standards.ieee.org/reading/ieee/std_public/description/testtech/1149.1-1990_desc.html. [Accessed: Feb. 13, 2010].

[107] “OpenEmbedded”, Feb. 6, 2010. [Online]. Available: http://wiki.openembedded.net/index.php/Main_Page. [Accessed: Feb. 12, 2010].

[108] “The Angstrom Distribution: Embedded Power”, Nov. 30, 2009. [Online]. Available: http://www.angstrom-distribution.org/. [Accessed: Feb. 12, 2010].

[109] "Java Virtual Machine", Feb. 13, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Java_Virtual_Machine. [Accessed: Feb 13th, 2010].

[110] Oracle Corporation, “The Java Virtual Machine Specification,” Oracle Corporation, 2010. [Online]. Available: http://java.sun.com/docs/books/jvms/. [Accessed: Feb. 13, 2010].

[111] Oracle Corporation, “Java Technology,” Oracle Corporation, 2010. [Online]. Available: http://sun.com/java/. [Accessed: Feb. 13, 2010].

[112] “Kaffe.org”, Feb. 26, 2008. [Online]. Available: http://www.kaffe.org/. [Accessed: Feb. 13, 2010].

[113] “CacaoVM”, Mar. 16, 2009. [Online]. Available: http://www.cacaovm.org/. [Accessed: Feb. 13, 2010].

[114] “JamVM”, Feb. 13, 2010. [Online]. Available: http://sourceforge.net/projects/jamvm/. [Accessed: Feb. 13, 2010].

[115] Technologic Systems, TS-7300 Manual Hardware and Software, Technologic Systems, 2008.

[116] IOGEAR, GWU523 Wireless-G USB 2.0 Adapter, IOGEAR Incorporated, 2005.

[117] IEEE Standards Association, “IEEE Standard for Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” IEEE. [Online]. Available: http://standards.ieee.org/getieee802/802.11.html. [Accessed: Feb. 13, 2010].

[118] "IEEE 802.11", Feb. 4, 2010. [Online]. Available: http://en.wikipedia.org/wiki/IEEE_802.11. [Accessed: Feb 13th, 2010].

[119] "Open-source software", Feb. 9, 2010. [Online]. Available: http://en.wikipedia.org/wiki/Open_source_software. [Accessed: Feb 13th, 2010].

[120] “The GNU Operating System”, Feb. 12, 2010. [Online]. Available: http://www.gnu.org/. [Accessed: Feb. 13, 2010].

[121] Technologic Systems, Linux for ARM on TS-72XX User's Guide, Technologic Systems, 2008.

[122] “BusyBox”, Jan. 27, 2010. [Online]. Available: http://busybox.net/. [Accessed: Feb. 13, 2010].

[123] G. Sally, *Pro Linux Embedded Systems*, New York, NY: Springer-Verlag New York, 2010, pp. 293 - 308.

[124] “Debian”, Feb. 12, 2010. [Online]. Available: http://www.debian.org/. [Accessed: Feb. 13, 2010].

[125] Fairchild Semiconductor, LM2902,LM324/LM324A,LM224/LM224A Quad Operational Amplifier, Fairchild Semiconductor, 2002.

[126] ThingMagic Incorporated, MERCURY5e World-class UHF RFID Engine, ThingMagic Incorporated, 2009.

[127] ThingMagic Incorporated, “Mercury5e,” ThingMagic Incorporated, 2009. [Online]. Available: http://www.thingmagic.com/embedded-rfid-readers/mercury5e. [Accessed: Feb. 13, 2010].

[128] Axis Communications, AXIS 210/211 Network Cameras - User’s Manual, Axis Communications, 2007.

[129] “BASH”, Nov. 20, 2006. [Online]. Available: http://www.gnu.org/software/bash/. [Accessed: Feb. 13, 2010].

[130] C. Newham and B. Rosenblatt, *Learning the bash Shell*, Sebastopol, CA: O'Reilly and Associates, 1998.

[131] K. O. Burtch, *Linux Shell Scripting with Bash*, Indianopolis, IN: Sams Publishing, 2004.

[132] “Ubuntu”, 2010. [Online]. Available: http://www.ubuntu.com/. [Accessed: Feb. 13, 2010].

[133] J. Postel and J. Reynolds, "RFC959 - File Transfer Protocol, " RFC959, october, 1985.

[134] Technologic Systems, TS-7300 Datasheet, Technologic Systems, 2007.

[135] “RXTX”, Feb. 4, 2009. [Online]. Available: http://www.rxtx.org/. [Accessed: Feb. 13, 2010].

[136] Oracle Corporation, “Java Communications,” Oracle Corporation, 2010. [Online]. Available: http://java.sun.com/products/javacomm/index.jsp. [Accessed: Feb. 13, 2010].

[137] Intel Corporation, “Intel Atom Processor: Intel’s Smallest Chip,” Intel Corporation, 2010. [Online]. Available: http://www.intel.com/technology/atom/. [Accessed: Apr. 5, 2010].

[138] Cirrus Logic, “EP9301 User’s Guide,” Cirrus Logic Incorporated, 2004.

[139] Altera, “Cyclone II Device Handbook,” Altera Corporation, 2008.