

CS 411W Lab II

Prototype/Product Specification
For
AIR Tracker

Prepared by: Rahil Patel, Green Team

Date: 03/25/2009

Table of Contents

| | |
|---|----|
| 1 Introduction | 3 |
| 1.1 Purpose | 4 |
| 1.2 Scope..... | 6 |
| 1.3 Definitions, Acronyms, and Abbreviations..... | 8 |
| 1.4 References..... | 9 |
| 1.5 Overview | 9 |
| 2 General Description..... | 10 |
| 2.1 Prototype Architecture Description..... | 10 |
| 2.2 Prototype Functional Description..... | 12 |
| 2.3 External Interfaces..... | 13 |
| 3 Specific Requirements | 14 |
| 3.1 Functional Requirements | 15 |
| 3.2 Performance Requirements | 27 |
| 3.3. Assumptions and Constraints | 32 |
| 3.4 Non-Functional Requirements | 33 |
| Appendix..... | 34 |
| Formal Resource Request Document | 34 |

List of Figures

| | |
|---|----|
| Figure 1. Bag Flow | 5 |
| Figure 2. RWP and Prototype Comparison | 7 |
| Figure 3. Prototype Major Functional Components Diagram..... | 11 |
| Figure 4. Demonstration Kit Set-up..... | 16 |
| Figure 5. Airport Schema | 18 |
| Figure 6. AIR Tracker Schema..... | 18 |
| Figure 7. Gate State Diagram Figure 8. Cart State Diagram..... | 20 |
| Figure 9. Algorithm Input Screen | 22 |
| Figure 10. Handheld Alert System GUI Site Map | 25 |
| Figure 11. Master Baggage Handler GUI Site Map..... | 26 |

List of Tables

| | |
|--|----|
| Table 1. Assumptions, Constraints, and Dependencies..... | 32 |
|--|----|

1 Introduction

Mishandled luggage costs the airline industry as a whole \$3.8 billion a year [SITA, 2009]. The term “mishandled” includes bags that are sent to the incorrect destination, too late for the passenger’s itinerary or lost altogether. Returning mishandled luggage not only costs a lot, but it also frustrates travelers.

Travelers can become frustrated after their bag is mishandled for several reasons. People may have necessities, such as clothes or business documents, in their bag. They may have invaluable items that cannot be replaced by a small sum of money. Also, waiting for the bag to be retrieved may cause the traveler anxiety. Any of these reasons will make the traveler reconsider booking a flight with that airline again.

Airlines suffer even more; government provided statistics prove this. According to the U.S. Department of Transportation, in August 2008, bags were mishandled at a rate of 8,700 a day by American-based airlines. 61% of all mishandled bags were caused by transfer related issues, and an additional 7% of them were caused by unloading errors [SITA, 2009]. Each mishandled bag costs the airport an average of \$90 per bag [SITA, 2009]. In order to recover bags, the airlines must hire more personnel and use more resources to save them. Some airlines are known to have a high rate of mishandled bags which customers try to avoid. For example, Delta Airlines has been dubbed Don’t Expect Luggage to Arrive!

Mishandled luggage has proved to be a major problem in today’s society and needs to be mitigated. To alleviate this problem, a better quality assurance (QA) system is needed. The system should use current technology, have more checkpoints near areas involving transfers, and provide complex reports through an effective GUI. AIR Tracker has all of the previously stated characteristics and more.

1.1 Purpose

AIR Tracker is a QA system with intentions to ultimately reduce the amount of mishandled bags. It provides real-time tracking of bags throughout the entire Ground-level Routing Process (GRP). It alerts the airport baggage handling staff when bags get off track. Lastly, it provides a historical summary of alerts to assist the airport in finding problem areas within the baggage handling system.

AIR Tracker has all of the previously stated characteristics and more. In addition to the checkpoints current quality assurance systems have (check-in and sorting), AIR tracker is able to track bags when they are loaded and unloaded from the cart which is crucial during transfers. Not only does the system track the bags, it can also sense when a bag has strayed away from its correct route and can safely send an alert to the responsible staff member. Furthermore, the cart scans would eliminate the need of scanning the bag at the bottom of the belt loader, thus effectively reducing the time for all flights, including transfers.

AIR Tracker uses five well placed checkpoints to gather data: check-in, TSA inspection, loading the pusher, loading the cart, and loading the airplane (see Figure 1). The first three checkpoints are provided by most existing baggage handling systems, whereas the latter two are provided exclusively from AIR Tracker due to the innovative use of cart sensors and wireless antennas (highlighted in Figure 1). During transfers, the two exclusive checkpoints are used twice as much, because the bag must travel from a plane to the pusher and vice versa. This allows AIR Tracker to provide crucial information and more possibilities to save mishandled bags with real-time alerts.

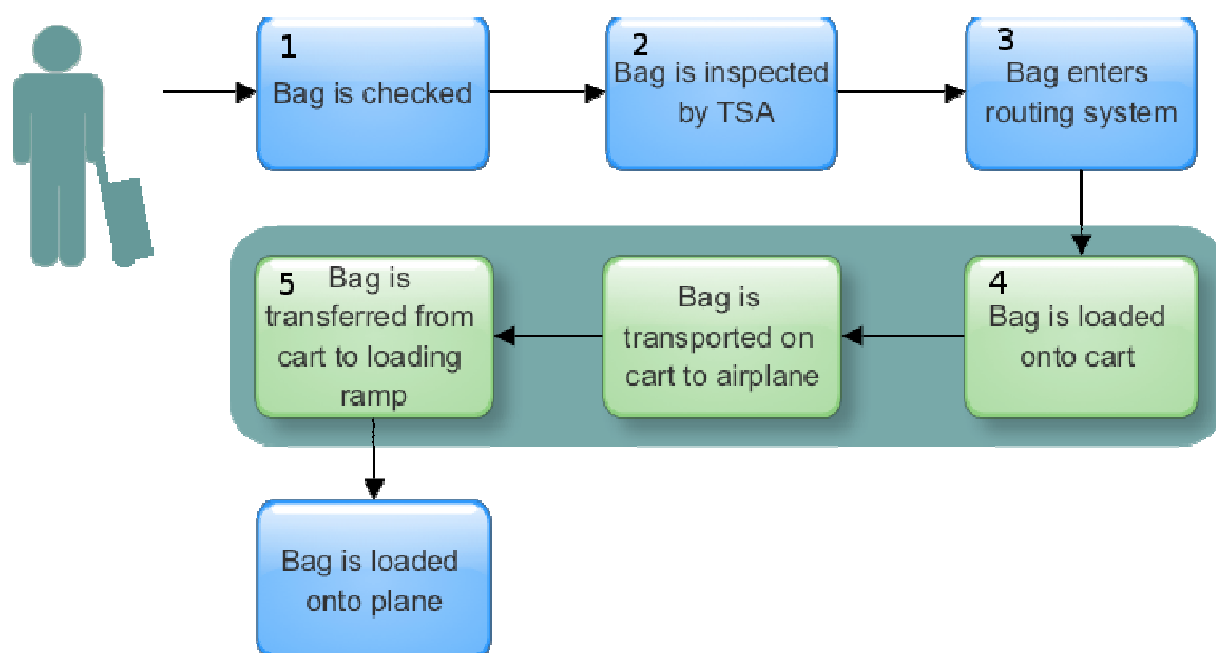


Figure 1. Bag Flow

AIR Tracker has several features, many of which are unique and put the product above any of the competition. The main attraction is the cart reader. The cart reader scans bags at the gate and the airplane, sending bag data to the applications server wirelessly. This removes the need for a scan before the cargo bay, which effectively saves time. The readers also send alerts to the driver if a bag falls off, which would directly reduce the number of mishandled bags, since the bag can be saved.

Since AIR Tracker is custom-built to fit the airport, the routing system will be able to generate more accurate reports than existing systems. This tight integration allows AIR Tracker to offer real-time alerts. Alerts, since the time AIR Tracker was implemented, can then be displayed and reported.

Airports of all sizes can benefit from AIR Tracker, but the primary targets are airline hubs. Hubs are airports that an airline uses as a transfer point, and consequently where many bags are mishandled. The perfect target would be hubs with high rates of mishandled bags. Although airlines own the baggage handlers and are responsible for any bag-related problems, the airport is still the customer because the

system will be implemented at specific locations within the airport. The airport will have to convince airlines that they need to get a new baggage handling quality assurance system such as AIR Tracker in place. Indirectly, airlines may pay for the system, but the check for the Group will come from the airport.

Airports and airlines are not the only ones to benefit from AIR Tracker either. Baggage handlers, passengers, and administrators all receive upgrades over the current way things work. Baggage handlers do not need to physically search anymore because bags are alerted in real-time providing the bag's location. Administrators will be equipped with an easy-to-use GUI, which requires fewer clicks to do more work. Lastly passengers, the ultimate customer, benefit the most due to the higher standards in baggage handling.

The ultimate goal of AIR Tracker is to reduce the rate of mishandled baggage. Immediately, the system will start saving bags that fall off the cart. After a fair amount of time has passed, AIR Tracker can evaluate the performance of the GRP, by creating reports based on past alerts. These reports are the key to saving mishandled bags, because they will tell the customer exactly where in the GRP bags are being mishandled.

1.2 Scope

The prototype of the AIR Tracker is designed to prove the feasibility of the product using very limited resources. The prototype will be scaled down and separated into four modules: hardware, databases, alert generation, and reporting. The hardware module can prove that ID readers can be used to gather information on-the-fly. The database module can prove that databases can be created and accessed according to the performance requirements in section 3.2.2. The alert generation module can prove that an algorithm can be used to determine if a bag is off track, and that it can send an alert GUI. Lastly, the reporting module can prove a GUI can be made that will query the database for past alerts.

First and foremost, the prototype's main goals are to show the ease of use, to demonstrate that the product is feasible, and be able to scale to the real world product. As seen in Figure 2, the real world product and prototype have many differences in hardware, which will directly reflect the prototypes overall scalability. The main differences are that the prototype will be using RuBee instead of RFID and virtual databases instead of separate servers.

| Features | Real-World Product | Prototype |
|------------------|---|------------------------------------|
| tracking device | RFID tag on bag | RuBee tag and simulated tags |
| scanners | RFID scanners | RuBee scanner |
| alert mechanism | handheld assistant | simulated alert |
| airport database | airport database on airport database server | simulated MySQL database on laptop |
| product database | AIR Tracker database on database server | simulated MySQL database on laptop |

Figure 2. RWP and Prototype Comparison

Although it seems quite different from the RWP, the prototype should still be able to complete a myriad of objectives. The objectives will be completed in logical order, starting with the RFID scanner and ending with reports. First, the scanner has to be able to grab data from a tag and send it to the simulated AIR Tracker application. Then, the prototype will be able to send alerts if the bag is not its projected path. Lastly, the data from the GRP simulation will have to be sent to the virtual AIR Tracker database, which then can be queried to create a report.

If the prototype is able to clear the aforementioned objectives, it proves that the product can and should work. The success of the prototype will attract funding, and future marketing capabilities. However, if the prototype is not successful, the Team would have to either revise the prototype or alter the project's architecture or scrap the project altogether.

1.3 Definitions, Acronyms, and Abbreviations

belt loader – the final portion of a checked-in bag’s journey from start to finish. After being unloaded from the cart, bags will travel up the belt loader and into the airplane, where they will remain throughout the flight.

cart – the in-between portion of a checked-in bag’s journey from start to finish. After traveling through the pusher, bags will be loaded onto carts to travel to a proper gate, where they will be placed on the belt loader for their correct flight.

gate – the location of arriving or departing flights in which loading baggage occurs.

ground-level routing process (GRP) – the start of the baggage-handling process until the moment the bag is placed onto the belt loader. All transfers of baggage through the pusher and cart portions of the checked-in bag’s journey are included in the GRP.

MySQL – an open-source relational database management system.

pusher – a conveyer system which consists of any number of conveyers and directional maze-like routes that “push” the bag toward its correct cart.

Radio-Frequency Identification reader – a radio wave-receiving device capable of identifying RFID tags.

Radio-Frequency Identification tag – a radio wave-transmitting device capable of being identified by an RFID reader.

RuBee reader – a magnetically-based receiving device capable of identifying RuBee tags.

RuBee tag – a magnetically-based transmitting device capable of being identified by a RuBee reader.

1.4 References

SITA. (2009). Baggage Report 2008. Retrieved October 25, 2008, from SITA Web site:

<http://www.sita.aero/content/baggage-report-2008>

Patel, Rahil. (2009). Lab 1 - AIR Tracker Product Description. Chesapeake, VA: Author.

U.S. Department of Transportation. (2008, August). Air Travel Consumer Report. Retrieved September 4, 2008, from U.S. Department of Transportation Web site:

[http://airconsumer.dot.gov/reports/2008 ... 08atcr.pdf](http://airconsumer.dot.gov/reports/2008...08atcr.pdf)

1.5 Overview

The following portions of this document provide the specifications for the AIR Tracker prototype. Section 2 contains general descriptions of the prototype's functions and their relations with other functions. It also contains information pertaining to the hardware, software, and user interfaces. Section 3 provides a more in-depth description of the requirements, which must be met to confirm the prototype's functionality.

[THIS SPACE INTENTIONALLY LEFT BLANK]

2 General Description

This section includes a description of the AIR Tracker prototype's architecture, major functions, and external interfaces. The prototype architecture description will detail the hardware and software of the major components. The functional description will provide a summary of the functions and how they relate to each other. The external interfaces section describes each interface, detailing the information to be transferred.

2.1 Prototype Architecture Description

Extremely similar to the RWP's MFCD, the prototype MFCD (in Figure 3) only replaces servers and the handheld device with virtual machines. Again, the process flow begins with a RuBee tag being read by a RuBee reader. The data is inputted into the simulated application server, which is constantly tracking bags in a simulated GRP. Another source for data can be coming from a simulated airport application, which will act as the check-in counter. The data from the GRP simulation will be stored in the virtual AIR Tracker database. Finally, the simulated application server may create reports by querying the database.

[THIS SPACE INTENTIONALLY LEFT BLANK]

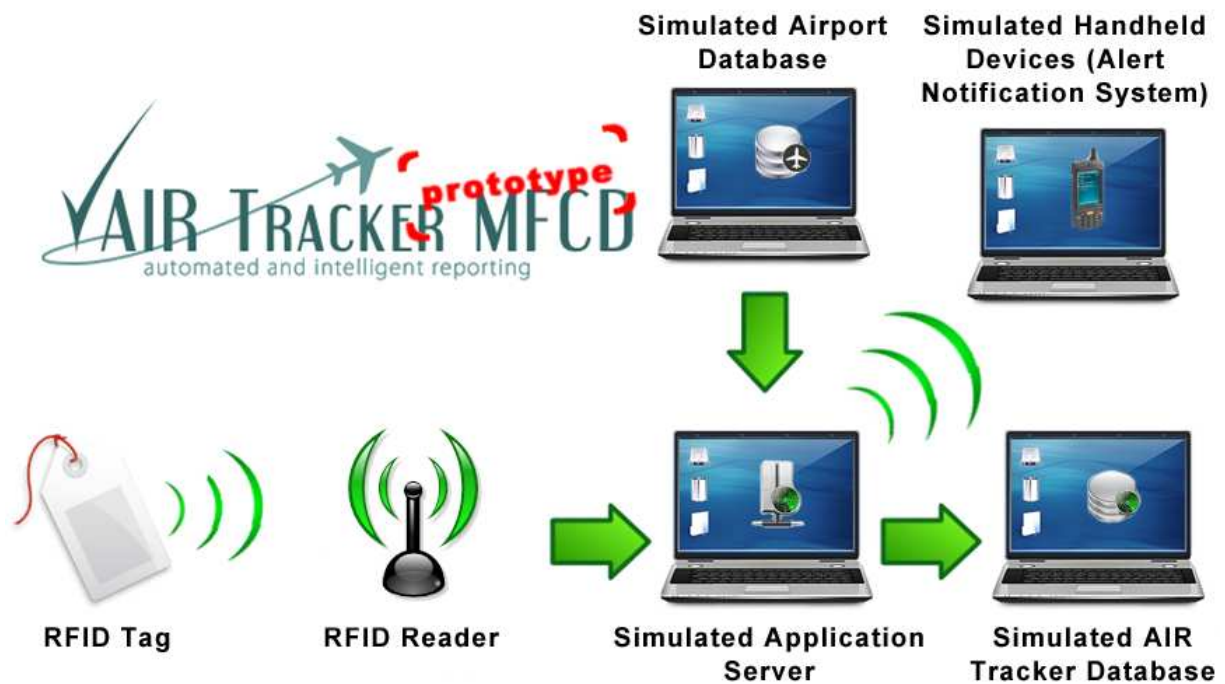


Figure 3. Prototype Major Functional Components Diagram

The main application and databases will be simulated on a single computer, whereas the RuBee tags and reader are real. Although airports use passive RFID, RuBee will be used in the prototype, which surpasses the specifications. All of the major hardware components for the AIR Tracker prototype are listed and detailed in below:

- RuBee Tags - The unique tags applied to model luggage
- RuBee Reader - The device which inputs data from the RuBee Tags and sends transmissions out
- Simulated Application server - The physical storage and computational hardware on which the AIR Tracker application is deployed
- Simulated Airport Database - The physical storage of an airport database
- AIR Tracker Database server - The physical storage of an AIR Tracker database
- Simulated Handheld Devices - Simulated system with graphical notification of mishandled luggage

The AIR Tracker prototype will make use of and be integrated with all hardware except the RuBee demonstration kit. This includes a simulation of the airport baggage handling system and the generation of real-time alerts. The prototype will also allow the user to create reports and modify factors of the simulation. The interfaces are discussed in section 2.3.2, and the prototype software is listed below:

- Simulated AIR Tracker application - Front-end for baggage data input and data manipulation. Also a back-end for data transfer to databases and RFID readers. The scaled version of this software wrapper uses simulated interfaces, which are non-disparate to the AIR Tracker application's source code.
- Simulated RFID firmware - Maintains that each bag on a particular cart is accounted for and routed correctly

2.2 Prototype Functional Description

The prototype process flow begins with a RuBee tag being read by a RuBee reader. For demonstration purposes, a RuBee demonstration kit will be used to show how RFID works, and how it will be used in the real world product. The laptop must be connected to the RuBee receiver in order to demonstrate. The demonstration kit includes software that will display tags found by the receiver and puts it into a text file.

To expand the demonstration, an interface between the text file and the AIR Tracker prototype will be made. This interface will continuously read the constantly changing text file, convert the data, and store it for later use in the AIR Tracker application.

The bag data from the RuBee tags are inputted into the airport database. Since there are only four tags in the demonstration kit, tags will also be generated in the airport simulation. The initial data from these tags (bag_id, flight_id) will also be forwarded to the AIR Tracker application discussed later. The simulation shows a bag going through the entire GRP. This includes check-in, AIR Tracker scanners

along the pusher, gates, carts, and the aircraft. In order to begin the simulation, the airport database must be populated with traveler itineraries. Using the flight number, an algorithm can be used to determine the path to the correct gate and eventually the destined aircraft.

The AIR Tracker application constantly receives bag information from the airport simulation. It gets the initial information from the beginning of the simulation and the rest is determined by using an algorithm on the information received from the scanners. By constantly comparing the current path of a bag to the correct path of a bag, AIR Tracker is able to find errors, and store them in the AIR Tracker database.

When an error is found, an alert is triggered. The alert is displayed in the form of GUIs. One GUI is for the baggage handler's handheld device; the other is for the master baggage handler's PC. The GUI for the handheld displays an alert, giving the location of the mishandled bag. The GUI for the PC is for creating and printing reports containing information stored in the AIR Tracker database.

2.3 External Interfaces

This section identifies the logical and physical interfaces used within the AIR Tracker prototype and the type of information being transferred. The three types of interfaces are hardware, software, and user. The communication involved in each interface is required for the operability of the AIR Tracker prototype.

2.3.1 Hardware Interfaces

There is only one interface for the AIR Tracker prototype. It is the interface between the RuBee Demo kit and the laptop. They are connected by a RS-232 serial port and a high-speed USB Type A adapter. The information exchanged on this interface includes the `bag_id` and the `flight_id`. This information is stored in the tag, then signaled to the receiver, and finally interfaced to the laptop.

2.3.2 Software Interfaces

Airport Database Interface: The Airport Database uses MySQL and can be accessed by the airport simulation using a SQL query. The database will be populated before the prototype demonstration and will contain information regarding traveler itineraries.

AIR Tracker Database Interface: The AIR Tracker Database also uses MySQL and can be accessed by the AIR Tracker application using a SQL query. The database will be used to store temporary routing information and alerts.

Graphical User Interface: The GUIs in AIR Tracker will be made using Qt, a cross platform graphical widget toolkit for the development framework of GUI computer programs. The two GUIs are the handheld device interface and the reports creation interface.

2.3.3 User Interfaces

The AIR Tracker prototype will require a standard keyboard and mouse to provide input to the system. A monitor capable of at least VGA will display the prototype GUI screens. Due to the common usage of these devices, the prototype will have the potential for widespread use.

3 Specific Requirements

The following section describes the specific functional, performance, and non-functional requirements of the AIR Tracker prototype. Hardware, software, and interface requirements detail the specifications that will be needed during prototype creation. In addition to these requirements, the prototype-specific test harness helps to create the full prototype.

3.1 Functional Requirements

The functional requirements in this section describe the capabilities of the AIR Tracker prototype. The requirements describe what the prototype must do in order to meet the goals and objectives of the overall project. Hardware, software, and interface requirements are detailed as necessary for the proper operation of the prototype.

3.1.1 Hardware

A portion of the prototype demands very specific hardware in order to operate. The key component of the prototype is an ODU student laptop that will run all of the software components and will connect to the RuBee scanner system to provide a real-world input to test the input / output interface software setup. The laptop should have a USB port if the USB connection is to be used.

3.1.1.1 RuBee Scanner Set-Up

To demonstrate RuBee's ability to communicate with the AIR Tracker application, software and hardware will both be used that has been provided by Visible Assets, Inc. The following is an inventory of the RuBee demonstration kit's contents and the requirements for set-up:

1. Receiver
2. Ranger antenna
3. Four RuBee radio tags
4. CD-ROM with RuBee Finder software and demonstration kit manual
5. Associated cables and power supplies

In order to successfully demonstrate that the RuBee hardware will work as intended, the RuBee Finder software must be installed on the ODU student laptop in accordance with the provided Visible Assets software demonstration kit manual. The associated hardware will be connected as shown in Figure 4, which is copied from the RuBee CD-ROM. The USB port adapter may be used instead of the RS-232 serial port.

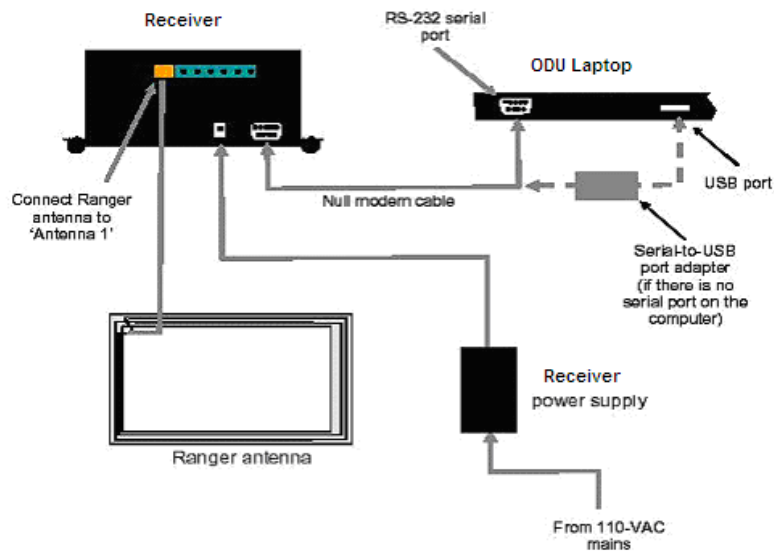


Figure 4. Demonstration Kit Set-up

Four RuBee tags come with the scanner and will be used with the cart module portion of the prototype. They will be used to prove the functionality of the “falling off of a cart” concept with the AIR Tracker system. The scanner and the tags meet the following requirements:

1. The RuBee receiver will be able to read a tag within the range of the antenna (Ref. 3.2.1).
2. The AIR Tracker application will read a seven-digit integer tag ID from a file of comma-separated values that will be created from the RuBee Finder software.
3. The RuBee Finder software and the RuBee reader will establish a connection via a nine-pin null modem cable.
4. The AIR Tracker application will be able to read the data file and extract a selection of information.
5. Any tags removed from within the range of the antenna are also removed from within the system.
6. Any tag that is brought back within range of the antenna after removing it will also be re-entered into the system.

3.1.1.2 Input/Output Interface (RuBee Software)

There shall be a method for data to be transferred from the RuBee reader to the AIR Tracker application. The functions of the reader, the tags, and the input / output interface must adhere to the following:

1. The RuBee Finder application and the RuBee reader will communicate via a nine-pin null modem cable.
2. The reader will read up to four tags that are within range and it will ignore tags that are out of range.
3. The RuBee finder application will write the tags that it has within range to a Comma-Separated Values (CSV) file.
4. The AIR Tracker application will read a seven-digit integer tag ID from the CSV file that will be created from the RuBee Finder application.
5. The CSV file will be updated with the current tags within range.
6. Establish that the AIR Tracker application removes the tag from the system when it is out of range of the antenna.

3.1.2 Databases

There will be two databases utilized in the AIR Tracker prototype. One will be the airport database, which is a simplified, simulated version of a real-world airport database. The other database will be the AIR Tracker database, which will maintain data pertinent to the AIR Tracker application.

3.1.2.1 Database Creation

The airport and AIR Tracker databases will be created with the schemas shown in Figure 5 and Figure 6, respectively. Both databases will be created in second normal form using database best practices. Both databases will be created using MySQL database software (v. 5.1).

| Flight | | |
|------------------|-------------|------|
| Field | Type | Null |
| <u>flight_id</u> | varchar(12) | No |
| airplane_id | varchar(20) | No |
| gate_no | varchar(4) | No |

| Traveler | | |
|--------------------|-------------|------|
| Field | Type | Null |
| <u>traveler_id</u> | bigint(20) | No |
| first_name | varchar(24) | No |
| last_name | varchar(24) | No |
| tflight_id | varchar(8) | No |

| Bag | | |
|---------------|------------|------|
| Field | Type | Null |
| <u>bag_id</u> | bigint(20) | No |
| t_id | bigint(20) | No |

Figure 5. Airport Schema

| Carrier | | |
|-----------|------------|------|
| Field | Type | Null |
| <u>id</u> | bigint(20) | No |
| bag_id | bigint(20) | No |
| cart_id | int(10) | No |

| Alert | | |
|-----------------|------------|------|
| Field | Type | Null |
| <u>alert_id</u> | int(10) | No |
| bag_id | bigint(20) | No |
| last_scanner | int(10) | No |
| alert_occurred | datetime | No |
| clear_type | varchar(8) | No |
| clear_occurred | datetime | No |

| Bag | | |
|---------------|-------------|------|
| Field | Type | Null |
| <u>bag_id</u> | bigint(20) | No |
| flight_id | varchar(12) | No |

| Alert_Type | | |
|----------------|-------------|------|
| Field | Type | Null |
| <u>type_id</u> | int(10) | No |
| type_descript | varchar(20) | No |

| Loader | | |
|------------------|-------------|------|
| Field | Type | Null |
| <u>loader_id</u> | int(10) | No |
| scanner_id | int(10) | No |
| gate_id | varchar(20) | No |

| Cart | | |
|----------------|-------------|------|
| Field | Type | Null |
| <u>cart_id</u> | int(10) | No |
| scanner_id | int(10) | No |
| gate_id | varchar(20) | No |

Figure 6. AIR Tracker Schema

3.1.2.2 Database Population

The initial population of the database will be in accordance with the following parameters:

1. The airport database will be manually populated with valid itinerary information for approximately 100 simulated passengers and their luggage.
2. The AIR Tracker database will be populated with information about passenger itineraries derived from the airport database and also the airport's ground routing hardware.
3. The AIR Tracker database will be manually populated with information on airport routing hardware with respect to the complexity of a particular scenario.

3.1.3 Airport Simulation

The simulated portions of the prototype will consist of internal and external routing modules. The internal routing simulation consists of tracking a bag from check-in until it is loaded onto a cart. The external routing simulation consists of tracking a bag from a cart to a belt loader scanner.

3.1.3.1 Internal Routing

Internal routing is the portion of the GRP that takes place inside the airport. This includes check-in, the TSA check, and routing along the pusher. Once the bag reaches the gate, the external routing function takes over (Ref. Figure 7 and Figure 8).

Since the RuBee demonstration kit has very limited resources, the AIR Tracker prototype will simulate the check-in process. The RuBee tags will be shown to work as per functional requirements 3.1.1.1 and 3.1.1.2, but in order to meet the functional requirements of the simulation, tags must be simulated to show operability for a large group of tags.

The airport internal routing simulation will be able to:

1. generate bags and associate them with randomly-generated itineraries
2. generate 50 scanners

3. generate 10 gates
4. automatically route bags to the correct gate in accordance with its flight by using a routing matrix style algorithm
5. let the user manually route a bag by altering the *route_id* via the test harness.

3.1.3.2 External Routing

Once the tag reaches the virtual loading bays, the tag effectively enters into the domain of the external routing algorithms. Figure 7 and 8 show the multiple states a tag can be in at any given time.

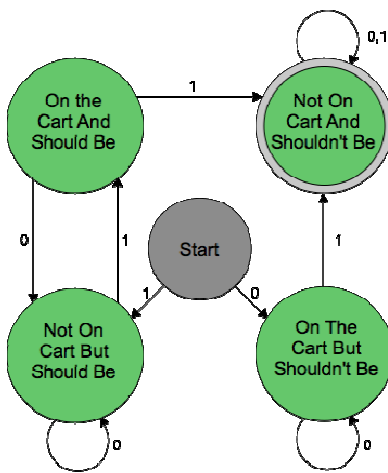


Figure 7. Gate State Diagram

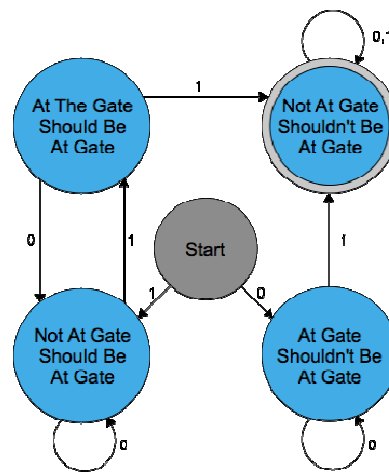


Figure 8. Cart State Diagram

The external routing simulation will show the following:

1. Once a tag is loaded onto a cart, the cart will check the Routing Algorithm to ensure that the tag is on the correct cart (Ref. 3.1.4).
 - i. If the tag is on the correct cart, the onboard cart module will update the tag's state as "On the Cart and Should Be." (Skip to #5.)
 - ii. If the tag is not on the correct cart, the onboard cart module will update the tag's state as "On the Cart But Shouldn't Be" and will flag an alert within the AIR Tracker system.
- (Continue to #2.)

2. An alert will be sent from the AIR Tracker system to the Baggage Handler Handheld GUI system (Ref. 3.1.5.1).
3. The alert will stay on the Handheld GUI system for a time period not to exceed that which is given in section 3.2.5.1.
4. Once the tag is removed from an incorrect cart, the onboard cart module will update the status within the AIR Tracker system.
5. If a tag is on the correct cart and is removed, the onboard cart module does a check via the AIR Tracker application to discover whether the tag is within range of the final destination gate.
 - i. If it is not within range of the final destination gate, the onboard cart module will update the tag's state to "Not On the Cart But Should Be" and will flag an alert within the AIR Tracker system. (Refer back to #2.)
 - ii. If it is within range of the final destination gate, the onboard cart module will update the tag's state to "Not On Cart And Shouldn't Be." It will then remove the tag from the onboard cart module.
6. Once a tag is in range of a gate scanner, the onboard gate module compares the tag's ID to that which is within the database.
 - i. If the tag is not destined for that gate, the onboard module checks the system to see if the tag is currently associated with a cart. If it is, no action is taken. If it is not, it will update the tag's state to "At Gate Shouldn't Be At Gate" and it will flag an alert within the AIR Tracker system. (Refer back to #2.)
 - ii. As soon as a tag is scanned by the correct gate's local scanner, the tag is removed from the active routing system.

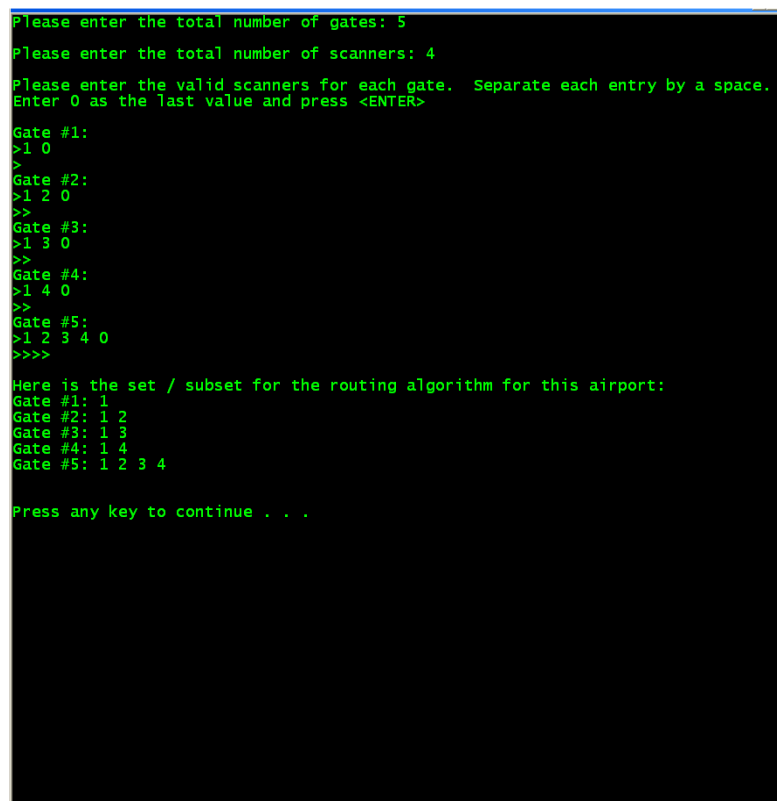
3.1.4 Routing Algorithms

There will be a master algorithm that will determine luggage states. If a piece of luggage is found to be in an incorrect state, the algorithm will generate an alert. The alert will then prompt the graphical user interfaces and store this information in the AIR Tracker database.

3.1.4.1 Routing Algorithm Creation

This function shall allow for the creation of the routing algorithm based on the virtual airport's configuration. The following functional requirements shall be provided:

1. An input display that will provide for any number of routing sets and subsets to be created based on the number of scanners, loading bays, carts, and gates. An example of part of the display is shown in Figure 9. This screen will be able to handle any number of gates and scanners and can be tailored for user preference.



```

Please enter the total number of gates: 5
Please enter the total number of scanners: 4
Please enter the valid scanners for each gate. Separate each entry by a space.
Enter 0 as the last value and press <ENTER>

Gate #1:
>1 0
>
Gate #2:
>1 2 0
>>
Gate #3:
>1 3 0
>>
Gate #4:
>1 4 0
>>
Gate #5:
>1 2 3 4 0
>>>>

Here is the set / subset for the routing algorithm for this airport:
Gate #1: 1
Gate #2: 1 2
Gate #3: 1 3
Gate #4: 1 4
Gate #5: 1 2 3 4

Press any key to continue . . .
  
```

Figure 9. Algorithm Input Screen

3.1.4.2 Routing Algorithm Access

A data set containing a collection of loading bays and associated scanners, and a collection of gates and associated carts shall be stored in an easily accessible manner by the system to allow for quick access to the information. Subsets of this data set will be loaded onto each cart and a list of bags will be checked every time a new bag is scanned to determine whether the new bag is in a correct state. A list of bag data within resident memory is preferred.

3.1.4.3 Alert Generation

Any time an alert is generated by the internal or external routing system as described in sections 3.1.3.1 and 3.1.3.2, the alert will be inserted into the database with the following information:

1. tag ID
2. timestamp of the alert
3. location within the routing system that the alert occurred (i.e. scanner number or cart number).

For the external routing system, an alert will be sent to the Handheld Alert System GUI and will remain active for ten seconds before it is forwarded to the master baggage handler if it is not cleared. The master baggage handler serves as the next tier of control in monitoring the baggage routing process.

3.1.4.3.1 Internal Routing Alert

Within the scope of the virtual internal routing of the tag, the system will be accessing the routing algorithm every time the tag is scanned by a belt scanner. Once the algorithms are compared, an alert will be flagged if the bag is on an incorrect path. This alert will be sent to the master baggage handler because the handlers would be unable to fix the problem.

3.1.4.3.2 External Routing Alert

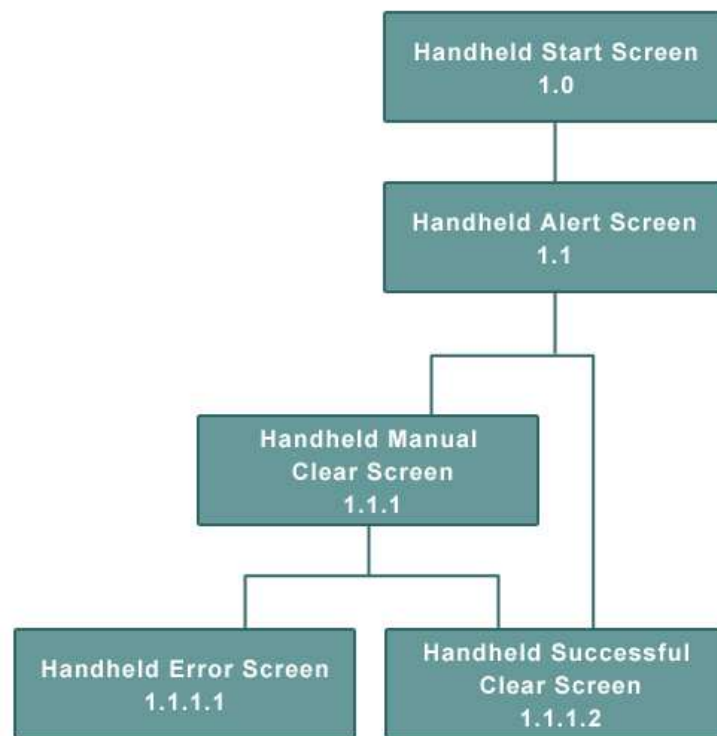
Within the scope of the virtual external routing, any time a tag is scanned by a cart or a gate, the routing algorithm will be compared. The results of the comparison are elaborated by the following:

1. If the comparison shows the tag is on the correct cart, update the state of the bag accordingly.
No alert will be flagged in this case.
2. If the comparison shows the tag is on an incorrect cart, an alert must be flagged and the state of the tag must be updated to reflect that.
3. Once a bag is removed from a cart, the state of the bag must be updated again. If the bag is removed from a correct cart, a query must be done to find if the tag is within range of the correct gate. If it is not, the gate onboard module will flag an alert within the AIR Tracker system which will be sent to the baggage handler handheld GUI.

3.1.5 Graphical User Interfaces

All graphical user interfaces (GUIs) will operate as visually appealing functions of the AIR Tracker application. Two sets of GUIs are required, elaborated in Figure 10 and Figure 11. The Handheld Alert System GUI serves the ground-level baggage handlers who physically handle the transport of baggage from check-in to loading onto the airplane.

[THIS SPACE INTENTIONALLY LEFT BLANK]



Handheld Alert System GUI Site Map

Figure 10. Handheld Alert System GUI Site Map

The Master Baggage Handler GUI serves the manager or administrator of the baggage handlers as a secondary alert system and report generator.

[THIS SPACE INTENTIONALLY LEFT BLANK]

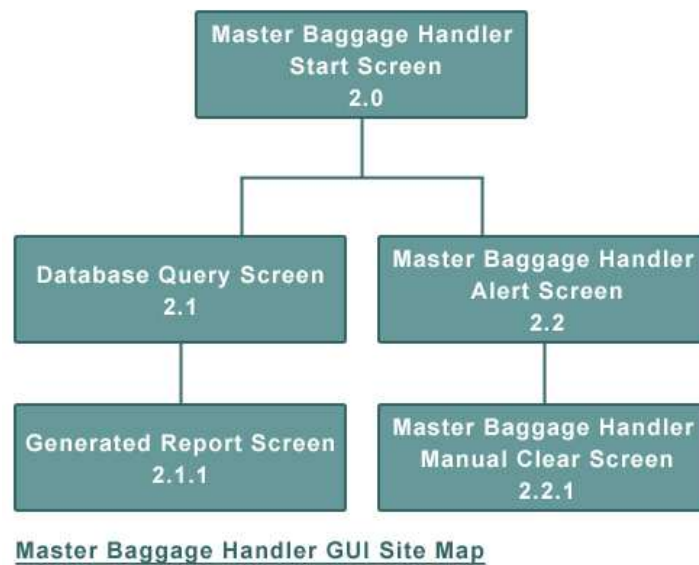


Figure 11. Master Baggage Handler GUI Site Map

3.1.5.1 Handheld Alert System GUI

This interface shall provide alerts on a simulated handheld device. The following functional capabilities shall be provided:

1. remain on a constant “ready” screen when there are no alerts to be handled, to include a brief message stating there are no alerts to be processed
2. switch to an “alert” screen when an alert has been received, to include the bag ID and scanner ID where the mishandling occurred
3. return to a “ready” screen when an alert has been automatically cleared (i.e. the application updates the status of the tag based on scanner input when the deficiency is corrected).
4. display multiple alerts in the “alert” screen when multiple alerts have been received
5. enable a baggage handler to manually clear an alert
6. return to a “ready” screen when an alert has been manually cleared
7. display an error if the alert is unable to be cleared due to false positives within the software or possibly an inoperable tag.

3.1.5.2 Master Baggage Handler GUI

This interface shall provide a means by which a master baggage handler may access near real-time alerts that have not been cleared by the ground baggage-handling crew (Ref. 3.2.5.2). The following functional capabilities shall be provided:

1. remain on a constant “ready” screen when there are no alerts to be handled, to include a brief message stating there are no alerts to be processed
2. switch to an “alert” screen when an alert has exceeded a set time limit on a baggage handler’s simulated handheld device, to include the tag ID and scanner ID where the mishandling occurred
3. return to a “ready” screen when an alert has been manually cleared

This interface shall also provide a means by which a master baggage handler may access historical data reports to assist in the generation of statistics (Ref. 3.2.5.2). The following functional capabilities shall be provided:

1. enable a master baggage handler to query the database of historical data in the AIR Tracker database, to include all parameters as determined by the AIR Tracker database
2. display the results of an AIR Tracker database query with the option to save as a report

3.2 Performance Requirements

In order to display maximum operability, AIR Tracker’s prototype must adhere to certain performance requirements. Many of the elements within the functional requirements have associated performance requirements that are in addition to any already mentioned. The test cases for the AIR Tracker prototype will be used to verify these goals and objectives.

3.2.1 Rubee Scanner And Tags

The RuBee scanner hardware must meet the following requirements:

1. able to scan up to four tags simultaneously
2. have a range of approximately 6 – 12 feet
3. constantly scan for tags and update the software at least every five seconds
4. ignore any tags that are removed from the range of the scanner
5. maintain a battery life of up to 4,000 hours

3.2.2 Database Queries

The “airport database to AIR Tracker” interface takes a selection of tag data from the airport database and sends it to the AIR Tracker database. This interface must adhere to the following performance requirements:

1. Pre-seeded airport database information will be queried by the AIR Tracker application.
2. Pre-seeded AIR Tracker database information will be made available to the AIR Tracker application (i.e. cart ID, relational information regarding gates and carts as well as loading bays and scanners).
3. The AIR Tracker application will be able to retrieve tag data and itinerary information from the AIR Tracker database.
4. The AIR Tracker application must be able to insert, delete, and update information in the AIR Tracker database.

The AIR Tracker database shall meet the following performance requirements:

1. handle 10 complex queries per second (queries requiring more than three table joins)
2. insert and update 20 database entries in the bag table per second
3. delete each bag entry in the bag table within 30 seconds of a completed transit

3.2.3 Algorithm

The algorithm is a major driving force behind the success of the AIR Tracker prototype. As such, there are certain requirements that must be met. These requirements for the implementation, the access, and the generation are displayed in the following sections.

3.2.3.1 Routing Algorithm Implementation

The algorithm creation process must allow a user to customize the setup of the following elements:

1. scanners
2. loading bays
3. carts
4. gates

It is in the ability to customize these elements that allow for the flexibility required to manage the routing of any given airport scenario. The data structure used to store these elements must be sufficient to maintain the setup for an airport with the following maximum capabilities:

1. 50 scanners
2. 10 loading bays
3. 10 carts
4. 10 gates

3.2.3.2. Routing Algorithm Access

Since the prototype will handle multiple tags with multiple scanners with multiple loading bays, quick access will be critical. It is recommended that for speed efficiency, a copy of the algorithm is maintained in a data structure in resident memory as well as a backup copy on the hard disk. The copy in memory will serve all scanner accesses while the copy on the hard disk will be used in re-creating the algorithm in the instance of a system reboot or failure.

3.2.3.3 Alert Generation

Another defining factor of the AIR Tracker prototype is that of the alerts that are generated upon tag mishandling. The alerts are absolutely necessary in providing the main scope of the prototype, which is to reduce the number of tags that are mishandled in any given simulated airport. As such, there are certain performance requirements that must be maintained. The database must be able to hold a historical collection of all alerts that occur that can be queried by the Master Baggage Handler. The storage should be sufficient to store 5,000 alerts.

3.2.4 Airport Simulation

Since the majority of this prototype is to be simulated, there are certain simulation methods that have strict requirements. These requirements ensure the system is able to handle the load that will be placed on it. The two facets of the simulation are the internal routing and the external routing.

3.2.4.1 Internal Routing

The internal routing of the bags encompasses the simulated check-in process and the routing through the belt scanners. The internal routing simulator must meet the following requirements:

1. allow for up to 50 scanners along differing paths
2. allow for up to 50 pushers along differing paths
3. allow for scanners to control the action of the pushers
4. allow for up to ten loading bays
5. allow a tag inter-arrival time of no less than five seconds at every scanner
6. provide a wait time between adjacent scans of approximately 5 seconds
7. allow scanners to misroute tags with user intervention

3.2.4.2 External Routing

The external routing of the bags encompasses the simulated carts and the final destination gates.

The external routing simulator must meet the following requirements:

1. allow for up to ten carts that will each service its own gate
2. allow for up to ten gates, each of which will be serviced by one cart
3. constant scanning of all tags on a cart
4. constant scanning of all tags near a gate
5. local scanning of a tag as it is loaded onto a gate belt loader
6. provide a method for the user to manually “drop” tags from a cart, or provide a method to dictate a percentage of random tags that are “dropped”.

3.2.5 GUI

All graphical user interfaces (GUIs) will perform as efficient functions of the AIR Tracker application.

Two sets of GUIs are required to perform as specified to prove the working functionality of the AIR

Tracker prototype. All performance requirements will satisfy scenarios both likely and unlikely to occur in a real-world situation.

3.2.5.1 Baggage Handler Handheld GUI

The Handheld Alert System Graphical User Interface shall meet the following performance requirements:

1. display up to five simultaneous alerts
2. display the automatic clearing of all five alerts
3. display the manual clearing of one alert
4. display an error in manual clearing

3.2.5.2 Master Baggage Handler GUI

The Master Baggage Handler Graphical GUI shall meet the following performance requirements:

1. display an alert after ten seconds has elapsed on a simulated handheld alert
2. display the manual clearing of an alert
3. display a list of parameters that will query the AIR Tracker database
4. display a historical data report in text format based on selected parameters
5. save a historical data report as a permanent text file

3.3. Assumptions and Constraints

As in any prototype, there are certain aspects of the operation that have assumptions, constraints, or dependencies placed on them. AIR Tracker is no different. Table 1 displays the assumptions, constraints and dependencies.

| Condition | Type | Effect on Requirements |
|---|------------|--|
| RuBee scanner and tag hardware will be available for the prototype demonstration | Dependency | In the case the hardware fails, the scanning process will be completely simulated |
| An ODU laptop is available to host the application. | Dependency | In the case of a laptop being unavailable, a student's laptop will be used. |
| A cart only services one gate | Assumption | Bounds the problem of "cross-talk" between cart antennae |
| Only valid data entries will be made | Assumption | Allows for minimal error checking within the code during the developmental and demonstration phases of the prototype |
| The itinerary for the tag does not change once it is entered into the system | Assumption | Allows for a one time query of the airport database, thereby transferring necessary data to the AIR Tracker Database |
| No passwords will be necessary for ensuring the baggage handlers are not abusing the alert clear function | Assumption | Allows for minimal error checking within the code during the developmental and demonstration phases of the prototype |

Table 1. Assumptions, Constraints, and Dependencies

3.4 Non-Functional Requirements

Aside from the functional requirements, there are several categories of non-functional requirements that must be met. They include security, maintainability, and reliability. The non-functional requirements of these elements are set forth in the sub-sections.

3.4.1 Security

There are no security concerns with this prototype due to the lack of a need for secure passwords or encryption.

3.4.2 Maintainability

The primary elements that need to be maintained are the databases and the algorithm. The databases will be maintained on the laptop used to run the application. The algorithm will be stored both within a database table and in resident memory for fast access.

3.4.3 Reliability

The AIR Tracker prototype has many elements that must work seamlessly together in order for the product to function properly. The overall goal of the prototype is to reduce the number of tags mishandled within the system, therefore minimizing the number of tags mishandled is a must with as little time as possible existing between mishandling and recovery. Although the time constraints are in place, there is some flexibility built into the system to allow for some fluctuation. The capacity of the system ensures that there are both processing power and sufficient time to route all tags. Since the prototype has a constraint of 100 tags routing through the virtual system at any given time, this allows for a slight ease of strain on the system so that it has the resources needed to accurately route all the tags.

Appendix

The appendix contains additional documentation for the AIR Tracker prototype.

Formal Resource Request Document

Team: AIR Tracker

Project Manager: Ashley Casper

The following resources are required **to be purchased** for the prototype development and demonstration of the AIR Tracker product:

Hardware Purchase (list all items required for purchase):

1. Serial Cable (female – female)
2. Serial to USB adapter

Software Purchase (list all items required for purchase):

1. None

The following **University resources** are required to support the prototype development and demonstration:

1. Laptop/ Second computer
 - a. It will be used to display data flow and database content during the live prototype demonstration.
 - b. Windows XP with connection to the internet
 - c. Quantity: 1
 - d. Deliver to: AIR Tracker c/o Neil Monday
 - e. PostgreSQL installed
 - f. RuBee software installed
 - g. Date required: As Soon As Possible