A PACKAGE TRACKING APPLICATION BASED ON SOFTWARE AGENTS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Vindhya Jonnalagadda

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

April 2012

Fargo, North Dakota

North Dakota State University

Graduate School

Title

A PACKAGE TRACKING APPLICATION

BASED ON SOFTWARE AGENTS

By

VINDHYA JONNALAGADDA

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard

Chair

Dr. Changhui Yan

Dr. Jun Kong

Dr. Limin Zhang

Approved:

06/14/12

Date

Dr. Brian Slator

Department Chair

ABSTRACT

JADE (Java Agent Development Framework) is a software environment, which is fully developed in the Java language. It supports multi-agent systems using an extensible agent model and predefined program classes. This software environment built agent systems in compliance with the FIPA specifications for interoperable multi-agent systems. The goal of JADE is to simplify development while ensuring standard compliance through a set of system services and agents.

In this project I propose and implement a multi-agent application. The objective of the application is to track packages on a real-time basis using a Multi-agent System, from when the package moves from one warehouse to another until it reaches the destination warehouse. Carrier agents and warehouse agents will carry packages. A tracker agent is used as a mediator, and will read the data from the carrier and warehouse agents and write the data to the database.

ACKNOWLEDGEMENTS

Sincere thanks to my advisor, Dr. Kendall Nygard, for his valuable suggestions, guidance, and support throughout my Master's paper and also for suggesting a topic using JADE framework. This is how I was introduced and became more interested in Multi-Agent Framework.

Special thanks to my friend Satheesh Chakravarthi for offering valuable suggestions and encouragement throughout my study at NDSU.

Finally, thanks to my parents and my husband; without their support and encouragement, I would not be able to come to this country and pursue my dream of graduating with a Master's degree in computer science.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

1. INTRODUCTION

1.1. RFID

It was not long ago that package tracking was almost entirely guess work. If we ordered a few packages that required shipping, which could come in two days or two weeks, where the package was between the shipper and recipient was a mystery. That is going to change with Radio Frequency Identification (RFID) [1].

RFID is a contactless and non-line of sight capturing technology, designed for automatic process. RFID is a process like barcode scanning, but it has more advantages.

The barcode system uses a label that is coded and requires visible symbols and light to carry-over the data from the coded tag to the code scanner. RFID uses a reader and a tag, which is connected to the package, and it uses the radio frequency transmission to send as well as receive the information from the RFID chip to the reader.

1.1.1. Advantages

RFID advantages over other identification technologies:

1. RFID tags either read only or read and write.
2. It can do read or write functions repeatedly.
3. It doesn't need any line of sight process.
4. It will operate under different environmental circumstances.
5. It returns a superior level of data accuracy.
6. This technology is difficult to forge and is extremely secure.
7. It will not do duplicate calculating [2].

1.1.2. Types of RFID Tags

1. Passive Tag: tags that rely on a reader or interrogator to communicate and provide the power essential to react and send information are called passive tags.

2. Active Tag: These tags have their own battery life and can send data a great distance. They can also send data perpetually.

These tags have three categorizations:

1. Read Only (ex. CD)
2. Read-Write (ex. CD-RW)
3. Read Only/Read-Write (ex. CD-R)

1.1.3. RFID Reader

An RFID reader uses low power radio signals to retrieve data from a tag. It can also program RFID tags. The RFID reader functions as an antenna to send/receive signals to/from tags. This antenna can be setup indoors or outdoors. Once the data is retrieved from the tag it is processed by the packaging tracking software system to send, via wired/wireless network system, back to the database. This information will be used by a principal computer (host).

1.2. Impact of RFID in Supply-Chain Management

Supply chain management's destination is to gain long-term functioning of individual companies and the total supply chain by maximizing the buyer value and reducing the supply prices. Not all companies accomplish these goals equally. A supply chain is either agile or lean, and based on this a distinct approach to improve efficiency and potency is adopted. Companies such as Wal-Mart and Dell have accomplished the efficiencies by getting a clear impression and secure commitment to return customer value by maximizing not only the value, but also adjusting their collaborators' concern to create unique supply chains.

Information systems are the backbone of every supply chain, and they are supported by automatic data finding formulas to fulfill the goal of gathering information. RFID is a technology with specific features that makes it appropriate to enhance the data collection operations along the supply chain.

1.2.1. Benefits

1. Enhancing Supply-Chain Control

As the location or warehouse part can be discovered accurately at every individual location, the entire supply chain will operate with nearly 100% accuracy.

2. Security and Authentication

An RFID tag can be attached with a unique identifier chosen by the company. This ID can be utilized to authenticate part of a parcel. This technology also allows encryption and additional security models so that a tag cannot be easily forged or faked.

3. Enhanced Support

The RFID technology can enhance customer service by providing quicker checkouts, tracking the location of packages, and personalizing service [3].

1.3. Package Tracking

Computers interconnected through a large network are used to track packages. However, post offices/courier companies generally do not track each package or parcel. Standard mail is normally not tracked, and advanced mail, such as premium mail services, will feature automatic tracking selection. A few services require additional payments from the sender to get the tracking request.

Generally, the cost of the mail or parcel depends on its weight and destination and is determined by the post office. Tracked mail can get lost as well, but it occurs infrequently. The

post office provides insurance, which will compensate the sender for the package value if the package gets lost or damaged.

A return receipt will have a tracking number and can be used to track the parcel using the Internet. Mail that is tracked goes through a particular process. It will be checked different times until it gets to the destination, and is updated to the database every time it is checked so that the customer can view the location of the package. Tracked mail is more cautiously reported and generally separated from regular packages to avoid confusion and incorrect delivery.

A tracking barcode or RFID chip will be attached to the package when it is set up for delivery. Every time a package reaches a new warehouse it will be scanned, and the data of the location will be automatically updated into the online database. Every post office that gets the package will be accountable for scanning the barcode or reading the RFID chip using an RFID reader. Tracking packages does not ensure that they will be delivered quicker. The main aim of this tracking system is for both the post office and the customer to know the location of the package.

1.4. Classification of Technologies to Track Packages

1.4.1. The Barcode

The Universal Product Code (UPC) is a barcode symbol for tracking packages in varying sectors. This is listed as one of the Top 20 Innovations that Rocked the World. This barcode has streamlined the process of purchasing, transporting, receiving, and selling different products or packages. Barcodes set the standard for digitizing and automating business, paving the way for the future of e-commerce.

The barcode is readable by an optical machine. The multiple digit code is unique and also can identify any product. Initially, the barcode proved to be a problem because presses would

smear the lines of ink and make it unreadable. To fix the problem, designers advised that parcels should be printed in the direction of the barcode stripes.

1.4.2. Radio Frequency Identification (RFID)

RFID tags are typically used in product packaging for tracking inventory. There are a lot of uses for RFID tags; for example, every book in a library has a tag, and there is a chip on every student identity card or library card. As people walk out with books they will automatically be added to the online bookshelf rather than having to wait in line and check out manually at the desk.

1.4.3. QR Codes

QR (Quick Response) codes carry the original UPC barcode to the next level. These two-dimensional codes have the ability to save individual product details and also unique links to websites. The contents of QR codes can be decoded at high speed because of its advanced design. The boxes in the code correspond to the alignment and positioning, which lets the scanners read and accredit the content immediately [4].

1.5. Proposed Idea

1.5.1. Goals and Objectives

The main goal of this paper is to demonstrate how radio frequency identification (RFID) chips can be utilized in a package tracking environment where software agents automatically sense the presence of RFIDs and communicate with other software agents to track a package. Software agents work great in a distributed environment. One of the important properties of software agents is that they can be coordinated and work as a team to perform a task, which is vital to implement a system of this type. The majority of the tracking is based on barcodes, and the barcodes need a line of sight, which might also limit the accuracy of the sensed data.

This paper implements a set of software agents to perform the tracking task, which are explained in detail in later sections.

1.5.2. Constraints

There are a few constraints identified that need to be addressed before the actual design of the system can be developed. The communication of RFIDs are complex, as they need a sensor and RFID chips, and interfacing with software agents needs special drivers, and considering the scope of this paper, the communication between RFIDs are abstracted. In the implementations, we assume the communication happens in a streamlined fashion and only the communication between various software is elaborated, which is vital in the context of this paper. Also, the trucks that carry the packages to and from different locations are also abstracted, and only the software agents, which directly interact, are explained.

1.6. Intelligent Agents

Intelligent agents are all natural intelligence or artificial intelligence. Intelligent agents are a wide range of entities like software. Intelligent agents can be used to understand the environment using sensors, and in certain environments, an intelligent agent can sense the environment through sensors and impact it using effectors. Intelligent agents can’t work alone without communicating with and sending messages to other agents. Individual agents can function actively and independently, which makes distributed AI possible, and has enormous practical value in such areas as parallel programming, computer communications, network management, and control.

Most of the important issues related to agents need to be addressed. Although there are a wide variety of objects like animals and software, and physical robots in a narrow sense, only

software robots are called agents or software agents. These represent mobile computing entities that can perform a set of operations as active service.

The term “active service” has two meanings:

1. Active Adaptation

In active adaption, the agent uses the information about the operating objects, users’ expectations, and preferences to do future tasks.

2. Active Implementation

In active implementation, agents are able to perform future operations based on the current state based on certain conditions without user inputs. The ability of the agent to access, coordinate, communicate and use resources is called Mobile [5] [6].

1.7. Definition of an Agent

An agent is used to describe a common hardware and software system that has the following four characteristics:

Autonomy: Able to operate on its own without the instructions issued by the user.

Social ability: Able to communicate, coordinate, and exchange information with other agents.

Reactivity: Able to respond to the change of the external environment or user behavior.

Proactive: Agent should not only be able to respond to the environment, but should also be able to display purpose-driven behaviors after receiving instructions.

It is very hard to define the properties of an agent, but it should at least have the following attributes:

1. Acting on behalf of others

This is the most important property of the agent. Agents should work for a user and should be able to access resources on behalf of users. They act as a bridge between users and resources.

2. Autonomy

This is the ability to allocate the required resources and services that suit the user without the user's help. Autonomy explains the ability of the agent to work independently without outside intervention. In an unpredictable and dynamic environment, agents should act independently, operate procedures, and provide solutions to the problems.

3. Proactive

Agents should exhibit purpose-driven behaviors and be able to take proactive actions in accordance with previous commitments. Agents on the Internet should be able to roam the entire network to get the information about users.

4. Reactivity

Agents should be able provide proper response to relative events.

5. Social ability

Social ability is the ability to conduct communications and exchange messages with users, resources, and other agents.

6. Intelligence

Agents should be able to understand the users' requirements in human language. To do this, agents should have a certain degree of intelligence, such as pre-denied rules and a self-learning AI reasoning machine. Agents should be able to predict users' intentions and realize these intentions for users [12].

1.8. Classification of Agents

Agents can be divided into several types based on different criteria.

- Classified as two types of agents based on a functionality interface agent (also known as a user agent or personal agent) and a software agent (also known as a task agent).
- Classified as four types of agents based on characteristics such as intelligent agent, cooperative agent, autonomous agent, and emotional agent.
- Classified as three types of agents based on architecture, such as hybrid agent, cognitive agent, and reactive agent.

1.9. Multi-Agent Systems

If it is difficult to reach goals using individual agents or a monolithic system, they may be reached using a Multi-Agent System (MAS), which is a system composed of several agents. Agents in MAS may range from hardware robots to software agents realized as processes/threads (softwoods) or interacting in distributed systems. Although some of the results apply to hybrid MAS, including robots and soft bots, this chapter is only focused on software agents. Multi-Agent System (MAS) agents may cooperate or compete, but there is some common infrastructure that results in the collection being a system.

A Multi-Agent System is considered the standard solution for addressing problems that require collective negotiation, intelligence, and cooperation. A Multi-Agent System is also used in opposition to big monolithic applications that are very hard to maintain and extend. MAS has no central management; they have local information.

While MAS can be used as centralized components, it shares many aspects of peer-to-peer systems.

More interesting systems are open and composed by agents that can be dynamically added and removed from the system composed of a fixed set of interacting agents that are created at the beginning of the application [5].

2. INTRODUCTION TO JADE

2.1. JADE Framework

JADE (Java Agent Development Framework) is a software development framework for developing multi-agent systems and applications meeting FIPA (Foundation for Intelligent Physical Agents) standards for intelligent agents. It includes two main products: a FIPA-compliant agent platform and a package to develop Java agents. JADE has been fully developed and coded in Java.

JADE is written in Java language and is made of several Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application-dependent tasks. Java was the computer programming language of choice because it has many attractive features especially geared toward object-oriented programming in distributed heterogeneous environments; a few of these features are Object Serialization, Reflection API, and Remote Method Invocation (RMI) [7].

jade.core implements the kernel of the system. It includes the Agent class that must be carried by application programmers; a Behavior class hierarchy is contained in the jade.core.behaviours sub-package. Behaviors carry out the tasks, or intentions, of an agent. They are logical activity units that can be compiled in several ways to accomplish complex execution patterns and that can be executed at the same time. Application programmers specify agent operations, writing behaviors and agent execution courses interlinking them.

The jade.lang.acl sub-package is offered to process Agent Communication Language according to FIPA standard specifications. The jade.content package carries a set of classes to support user-defined ontologies and content languages. A separate tutorial describes how to use the JADE support to message content.

The jade.domain package contains all the Java classes that represent the Agent Management entities defined by the FIPA standard, specifically the AMS and DF agents, which provide life-cycle, white and yellow page services.

The subpackage jade.domain.FIPAAgentManagement contains the FIPA-Agent-Management Ontology and all the classes representing its methods. The subpackage jade.domain.JADEAgentManagement contains the JADE extensions for Agent-Management (e.g., for sniffing contents and controlling the life-cycle of agents), including the Ontology and all the classes representing its concepts.

The subpackage jade.domain.introspection contains the concepts applied to the domain of discourse between the JADE tools (e.g., the Sniffer and the Introspector) and the JADE kernel. The subpackage jade.domain.mobility contains all concepts used to communicate about mobility.

The jade.gui package contains a set of generic classes useful to create GUIs to display and edit Agent-Identifiers, Agent Descriptions, and ACLMessages.

The jade.mtp package contains a Java interface that every Message Transport Protocol should implement in order to be readily integrated with the JADE framework, and the implementation of a set of these protocols.

The jade.proto package contains classes to model standard interaction protocols (e.g., fipa-request, fipa-query, fipa-contract-net, fipa-subscribe and others defined by FIPA), as well as classes to help application programmers create protocols of their own.

The FIPA package contains the IDL module specified by FIPA for IIOP-based message transport.

Finally, the jade.wrapper package provides wrappers of the JADE higher-level functionalities that allow the usage of JADE as a library, where outside Java applications launch JADE agents and agent containers [7].

JADE is fully developed in Java and is based on the following driving principles:

1. Interoperability – JADE is compliant with FIPA specifications. As a result, JADE agents can interoperate with other agents, provided that they comply with the same standard.

2. Uniformity and portability – JADE provides a homogeneous set of APIs that are independent of the underlying network and Java version. Specifically, the JADE run-time offers the same APIs for the J2EE, J2SE, and J2ME environment. In principle, application developers could determine the Java run-time environment at deploy-time.

3. Easy to use – The complexity of the middleware is covered by a simple and intuitive set of APIs.

4. Pay-as-you-go philosophy – Programmers don't necessarily need to use all the features offered by the middleware. The features that are not needed can be ignored by the programmer without any negative effects, and having the feature set doesn’t essentially add any computational overhead [8].

2.2. Why Use JADE?

JADE is a middleware that simplifies the evolution of applications. A lot of companies are already using it for very different application sectors, including supply chain management, holonic manufacturing, rescue management, fleet management, auctions, and tourism. A few papers of this special issue of the EXP journal provide evidence of the types of usage, while this section tries to identify which application features best benefit from JADE [8].

2.3. Distributed Applications and Autonomous Entities

First, JADE simplifies the evolution of distributed applications composed of independent entities that need to convey and cooperate in order for the entire system to work. A software framework that covers the complexity of the distributed architecture is made usable to application developers, who can focus their software development on the logic of the application instead of middleware issues, such as discovering and getting through the entities of the system.

This case of distributed applications enabled by JADE, in detail when applied to the mobile environment, ignited a new trend of evolution called smart-device smart-interconnection: the software on each device is equipped with autonomy, intelligence, and potentiality of collaboration, and the value of the system is shown by the devices' interaction and collaboration capabilities. This is quite different from the ubiquitous access trend, in which the value of the system is given by the content and the capacity of getting the content from anywhere [8].

2.4. Negotiation and Coordination

JADE makes developing applications that need negotiation and coordination within a group of agents, where the resources and the control logics are handed out in the environment, easy. In fact, easy-to-use software libraries to apply P2P (peer-to-peer) communication and interaction protocols (e.g., patterns of interaction between autonomous entities) are offered to developers by JADE.

2.4.1. Pro-activity

JADE agents control their own thread of execution and, hence, they can be easily designed to start the execution of actions without human interference just on the basis of a destination and state changes. This feature, generally known as proactivity, makes JADE an appropriate environment for the realization of machine-to-machine (m2m) programs, for

example, for industrial plant automation, traffic control, and communication network management.

2.4.2. Multi-party Applications

P2P (peer-to-peer) architectures are more effective than client-server architectures for designing and developing multi-party programs, as the server might become the bottleneck and the point of failure of the whole system. Because JADE agents can both offer and take services, they remove any need to differentiate between clients and servers. JADE agents let clients communicate with one another without the interference of a key server. Furthermore, intelligence, data, and control are distributed, which allows the realization of programs in which the ownership is allotted among the peers (agents), given that each peer may be capable, and authorized to execute, just a subset of actions of the program.

2.4.3. Interoperability

JADE complies with the FIPA specifications that feature interoperability among agents of other agent platforms. All applications where inter-organization communication is required can benefit from interoperability, including machine-to-machine and holonic manufacturing.

2.4.4. Openness

JADE is an open-source project that implies the contributions and collaborations of the user community. This user-driven approach gives both users and developers suggestions and new code, which assures openness and usefulness for the APIs. Of course, anarchy must be avoided, and the JADE Governing Board officially controls the development of JADE in terms of new APIs and functionalities.

2.4.5. Versatility

JADE provides a consistent set of APIs that are autonomous from the underlying network and Java version. It allows the same APIs for the J2EE, J2SE, and J2ME environment. This feature permits application developers to reuse the same program code for a PC, a PDA, or a Java phone; it allows deferring this option as late as possible, in theory, until the deploy-time.

2.4.6. Ease of Use and Mobile Applications

JADE API's are easy to learn and practice. JADE has been designed and developed to simplify the management of communication and content transfer by making the management of the different communication levels used to send content from an agent to a different agent transparent to the developer, allowing the developer to focus on the logic of the application. Naturally, the outcome of this feature is to make the development of applications quicker. JADE reduces the application development time regarding the time necessary to design and develop the same application by using only Java standard packages. Particularly when developing distributed applications for mobile terminals, JADE APIs and ready-to-use functionalities strongly reduce the application development time and prices (a few estimations have been given that show reduction of development time up to 30%) [8].

2.5. NetBeans Integrated Development Environment (IDE)

The NetBeans IDE is an open source computer program developed in the Java programming language. It offers the services common to creating desktop products -- such as window and menu management, settings storage -- and is also the first IDE to fully support JDK 5.0 features. The NetBeans platform and IDE are free for commercial and non-commercial use, and they are a part of Sun Microsystems.

2.5.1. Steps for Creating an Application in NetBeans IDE

1. Create Java Application

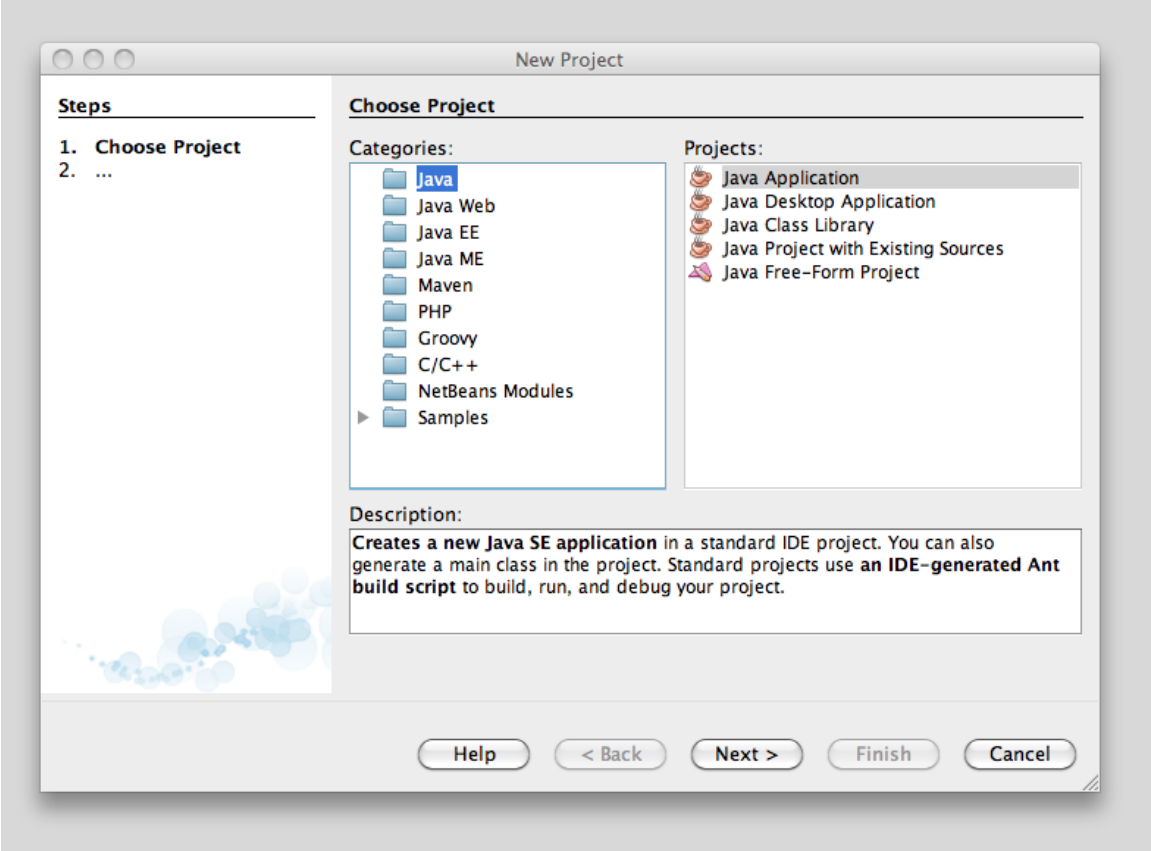

Figure 1. Creating a Java project in NetBeans IDE.

2. Create Source Packages and Class files for Agents and Utility.

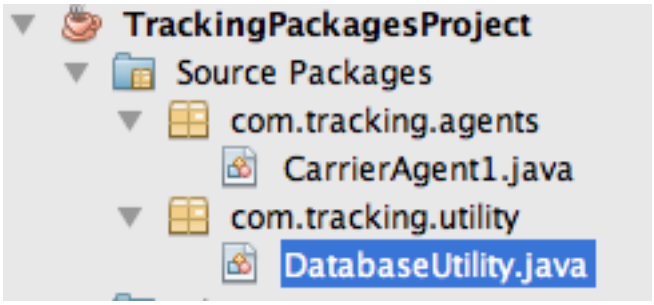

Figure 2. Creating source files packages.

3. Add JADE library to the existing project in order to extend JADE functionality.

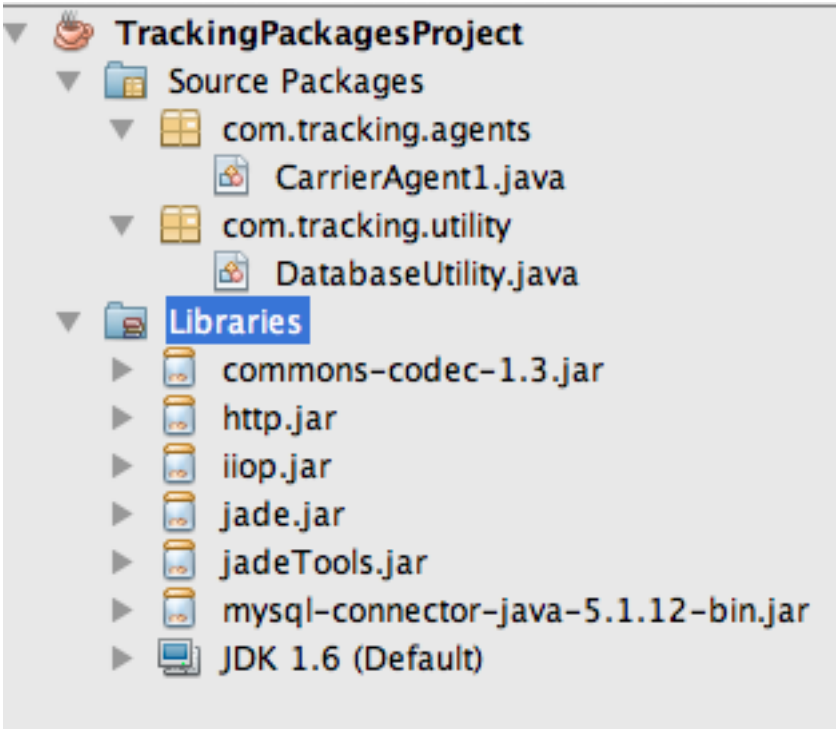

Figure 3. Adding libraries to the Java project.

The library also contains a MySQL adapter for database connectivity to the server. The figure below shows the final project explorer with all the required agent classes and utility classes.

2.6. MySQL

MySQL is a very fast, robust, relational database management system (RDBMS). A database enables people to efficiently save, search, order, and get data. The MySQL server controls access to the data to ensure that multiple users can work with it at the same time, to provide quick access to it, and to ensure that only authorized users have access. Hence, MySQL is a multi-user, multi-threaded server. It uses Structured Query Language (SQL), the standard database query language. MySQL has been publicly accessible since 1996, but has a development history going back to 1979. It is the world's most popular open source database and has won the Linux Journal Readers' Choice Award a number of times.

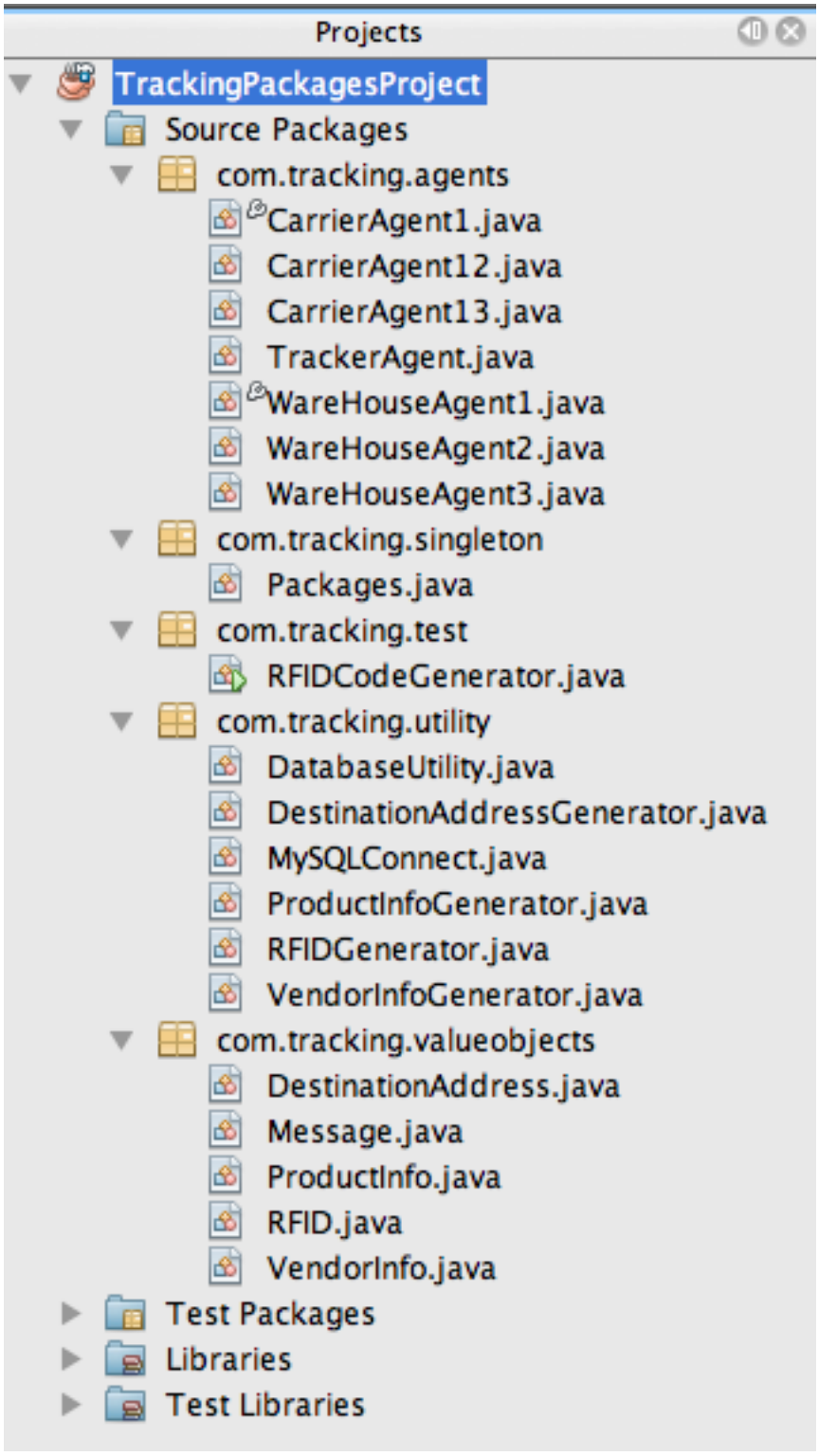

Figure 4. Project explorer window in NetBeans IDE showing all artifacts.

2.7. Hypertext Preprocessor (PHP)

PHP is a server-side scripting language designed and developed specifically for the Web. Inside an HTML page, people can embed PHP code that will be executed every time the page is visited. The PHP code is interpreted at the web server and generates HTML or other out-put that the visitor will monitor.

PHP is an open source project, which means people have access to the source code and can use, alter, and redistribute it for design and development without charge.

PHP in the beginning stood for Personal Home Page, but was changed in line with the GNU recursive naming convention (GNU = Gnu's Not Unix), and now represents PHP Hypertext Preprocessor.

2.7.1. Using PHP and MySQL

When planning to build a website, people can use many different products. They need to decide the following:

- Hardware for the web server
- An operating system
- Web server software
- A database management system
- A computer programming or scripting language

Some of these options are dependent on the others. For example, not all operating systems run on all hardware, and not all Web servers are compatible with all programming languages. One of the best characteristics of both PHP and MySQL is that they work with any major operating system and many of the minor ones.

The majority of PHP code can be developed to be portable between operating systems and web servers. There are some PHP functions that specifically relate to the file system that are dependent on the operating system.

2.7.2. Advantages of PHP

Some of PHP's main challengers are Perl, Microsoft ASP.NET, Ruby (on Rails or otherwise), Java Server Pages (JSP), and ColdFusion.

In comparison to these products, PHP has much strength, including the following:

a. Performance, b. Scalability, c. Interfaces to many different database systems, d. Built-in libraries for many common web tasks, e. Low cost and comfort of acquiring and use, f. Strong object-oriented support, g. Portability, h. Flexibility of development approach, i. Availability of source code, j. Availability of support and documentation.

2.7.3. Advantages of MySQL

MySQL's main challengers are PostgreSQL, Microsoft SQL Server, and Oracle. MySQL has much strength, including:

High performance, b. Low cost, c. Ease of configuration and learning, d. Portability, e. Availability of source code, f. Availability of support [30].

3. LITERATURE REVIEW

3.1. Evaluation of Agent-Based Frameworks

In the last decade, many methodologies for creating agent-based systems have been developed. A methodology is a set of rules and regulations for addressing the entire life cycle of system development, both technically and managerially. A methodology should provide the following: a full life cycle procedure; a broad set of concepts and models; a complete set of methods (regulations, guidelines, heuristics); a fully defined set of deliverables; a modeling language; a group of metrics; quality assurance; coding (and other) measures; reprocessing advice; and guidelines for project management. There are more than two dozen agent-oriented technologies. The numerous and varying methodologies has led to the following issues:

1. Industrial problem: choosing a methodology for developing an agent-based system is an important and a hard task, especially for industrial developers who accommodate specific requirements and constraints.

2. Standards problem: Multiple methodologies are counter-productive for reaching a standard. Without any achievable standard, potential industrial adopters of agent technology refrain from utilizing it.

3. Research problems: Excessive attempts are spent designing and developing agent-oriented technologies, which sometimes produces overlapping solutions. In addition, as a consequence of allocating resources to multiple methodologies, no methodology is assigned adequate research resources to address all aspects and offer a fully-fledged agent-oriented methodology [31].

3.1.1. Advantages of Agent-Oriented Framework

1. Distributes computational resources and potentialities over a network of interlinked agents.

2. Allows for the interconnection and interoperation of different existing legacy systems. By establishing an agent wrapper around such systems, they can be integrated into an agent society.

3. Models problems in terms of independent interacting component-agents, which is evidenced to be a natural way of representing job allocation, team designing, user preferences, and open environments.

4. Efficiently regains, filters, and globally organizes data from sources that are spatially distributed.

5. Provides answers in situations where expertise is spatially and temporally distributed.

6. Increases total system performance, specifically in the properties of computational efficiency, dependability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse [11].

3.1.2. Disadvantages of Agent-Oriented Framework

Since the agent-oriented prototype can be considered a development of the object-oriented paradigm, we considered the type of evaluations from the object-oriented methodologies.

These evaluations and comparisons arise from a number of basic flaws, as follows:

1. Applying an incompatible or inappropriate framework for executing the evaluation.

2. There's no agreement on what a methodology is or on what it should belong to.

3. Occasionally there's no grading for the support of a specific concept of a particular methodology. This results in disappointing results of the evaluation.

4. In many cases the evaluation cannot be duplicated [31].

3.2. How Delivery Services Works

Most package delivery companies like FedEx and UPS are not using the RFID technology effectively; they are using the barcode technology. Below is a brief overview of how their tracking systems work.

3.2.1. UPS Tracking System

The United Parcel Service (UPS) utilizes an advanced package-tracking system to supervise the status of the millions of parcels it delivers each day. The system, which has been running since 1992, allows a UPS driver to use a custom-made electronic clipboard, called a delivery information acquisition device (DIAD), to scan the parcels' barcode labels and record the recipients' signatures. The delivery data is then communicated as soon as the driver attaches the DIAD to an adapter in the truck. Transmission of the data is made via UPS nationwide cellular telephone network called UPSnet, which, in turn, communicates the data to UPS' Mahwah, NJ, data center. The UPS customer service team then has instant access to the delivery details [13][10].

3.2.2. FedEx Tracking System

FedEx, founded in 1971, picks up more than 4 million shipments per day from 220 countries, and has 42,573 drop boxes and approximately 1,800 service centers. FedEx generated revenues of $24.6 billion in fiscal year 2011.

FedEx's vision spawned the proprietary Super tracker, a nomadic device that records the package's unique barcode and then transfers the data once the device is docked in the courier's

delivery vehicle. FedEx couriers can also wear a wireless transmission device if they are away from their vehicles for long periods. Parcel routing is decided once the package data reaches a central office.

3.3. Features of Mobile Clients

3.3.1. USPS Mobile App

- Find U.S Postal Service (USPS) stores in the app when you need to get to a post office, Automated Postal Center, or collection box with an easy-to-filter locator. The nearest choices will be represented and include driving, walking, or transit directions.
- Search a ZIP Code for the current location, or get the ZIP Code for any U.S. address.
- Calculate a price to measure how much postage is needed for the consignment – letter, card, envelope, or parcel. There is also the choice of retail or internet pricing, adding additional services, and assessing the aggregate.
- Use the Track & Confirm tool to track Express Mail® shipments and check the status of parcels sent with other services. There is also the exclusive option to provide the shipment with a nickname and save it in the app to easily keep tabs on items.
- Schedule a next-day Free Package Pickup and have the carrier pick up Priority Mail, Express Mail, Global Express Guaranteed, or Merchandise Returns shipments from either home or office.
- Order USPS supplies and have them delivered to a home or office.
- Scan the barcode on shipping labels with an iPhone camera. The app acknowledges the shipment and saves the label number, so its delivery status can be tracked [13] [10].

3.3.2. FedEx Mobile App

- Track the status of any package or cargo shipment using My FedEx.
- Receive shipment emails or messages.
- Find staffed and self-service FedEx locations.
- Schedule a pickup for FedEx Express, FedEx Ground, FedEx Home Delivery and FedEx Freight shipments.
- Create a Mobile Shipping Label that can be scanned at a FedEx Office or a FedEx World Service Center location to print a shipping label.
- Create and email a shipping label.
- Get account-specific rate quotes.
- Access the fedex.com Address Book.
- Provide feedback directly to FedEx.

4. DESIGN AND DEVELOPMENT USING JADE IN NETBEANS

4.1. Design Architecture

This package tracking application was developed using NetBeans and JADE framework, and it consists of three carrier agents, three warehouse agents, and one tracker agent. Carrier agents are designed to carry the packages by picking them up from one warehouse and delivering them to another warehouse. Warehouse agents are designed to keep the packages for delivery using carrier agents. These warehouse locations can vary by different locations. The tracker agent is designed to communicate between the carrier agent and warehouse agent and it will also retrieve/send the information from/to the database.

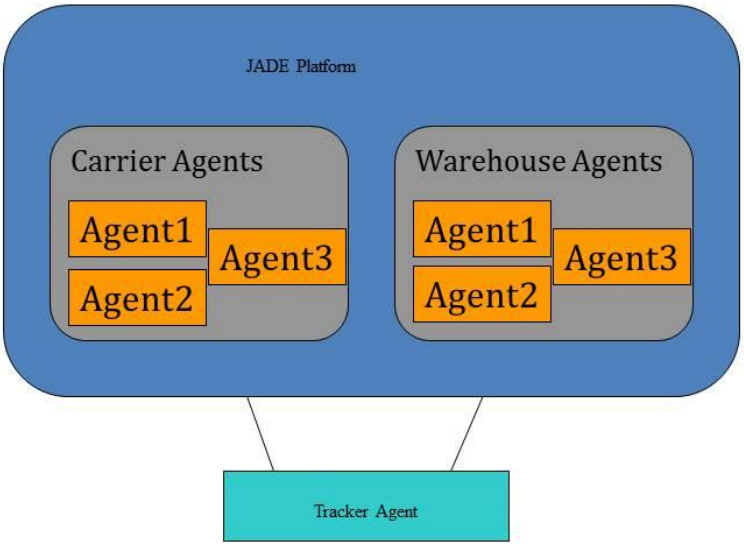

Figure 5a. High-level design architecture of the tracking application.

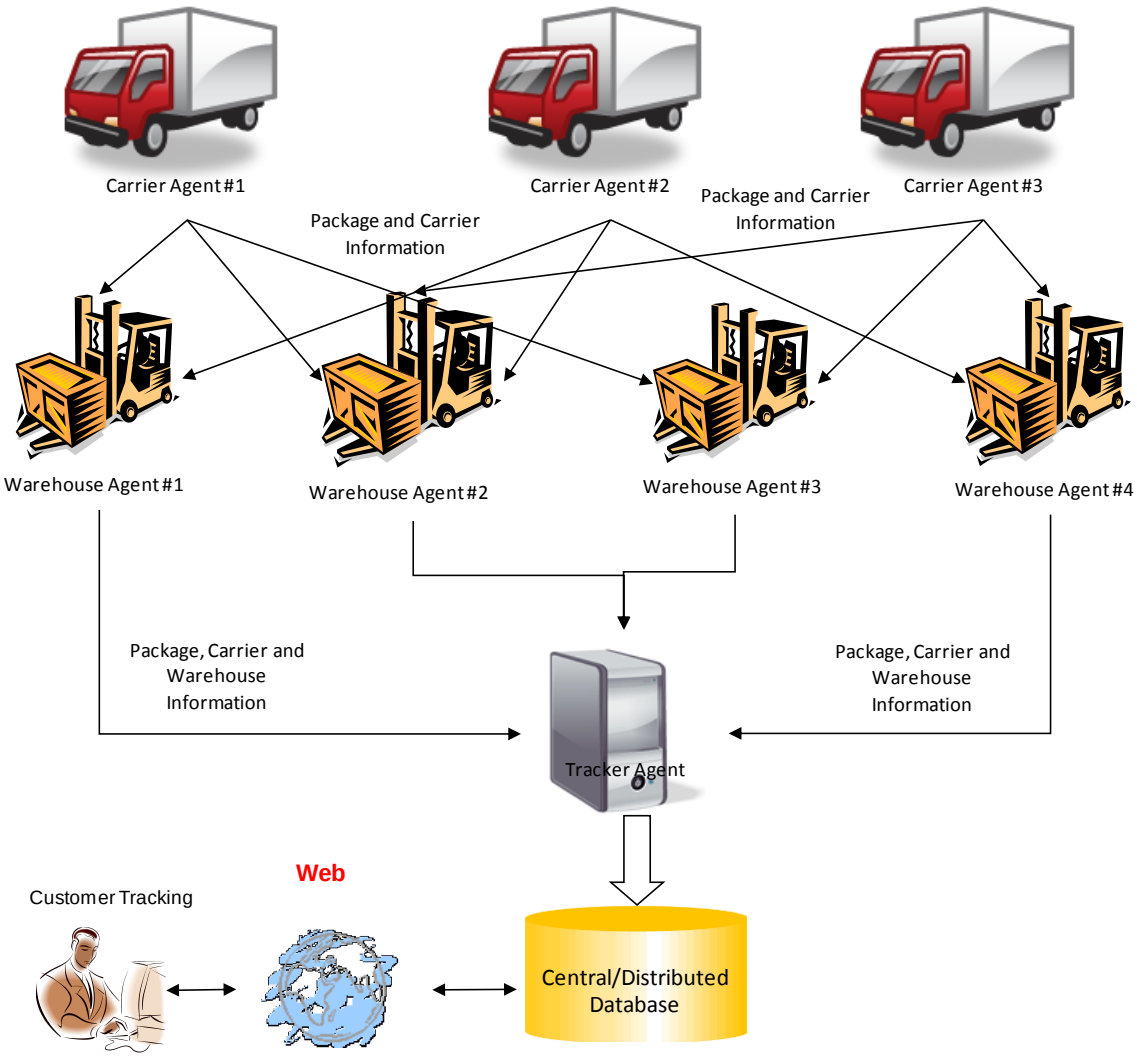

Figure 5b. Detailed design architecture of the tracking application.

4.2. Components of Package Tracking Application

There are various components involved in the design of the package tracking application, mainly regarding software agents.

1. Carrier agents
2. Warehouse agents
3. Tracker agent

4.2.1. Carrier Agents

Carrier agents are responsible for carrying the package from one warehouse or source location to a different warehouse or a destination location. Carrier agents work closely with warehouse agents. Carrier agents must ensure that all the packages are tagged using a unique RFID chip and also tagged with the source and the destination location so that each package can be uniquely identified and tracked.

4.2.2. Warehouse Agents

A warehouse agent senses the entry of a carrier agent, scans all the packages carried by the carrier agent, and updates the status of each package's location information to the tracker agent.

4.2.3. Tracker Agent

The tracker agent is responsible for receiving the communication from the warehouse agents and updating the information to a central/distributed database, where the information can be retrieved by a customer.

4.3. Demonstrating Package Tracking From a Development Perspective

4.3.1. Carrier Agent 1-1

Carrier agents are the agents that carry the packages from the warehouse and deliver them to another warehouse. There are three carrier agents for this project, and they will carry the packages in vehicles.

Carrier agent 1 will start to the destination first, and will communicate with warehouse agent 1 and the tracker agent.

Data Flow Diagram:

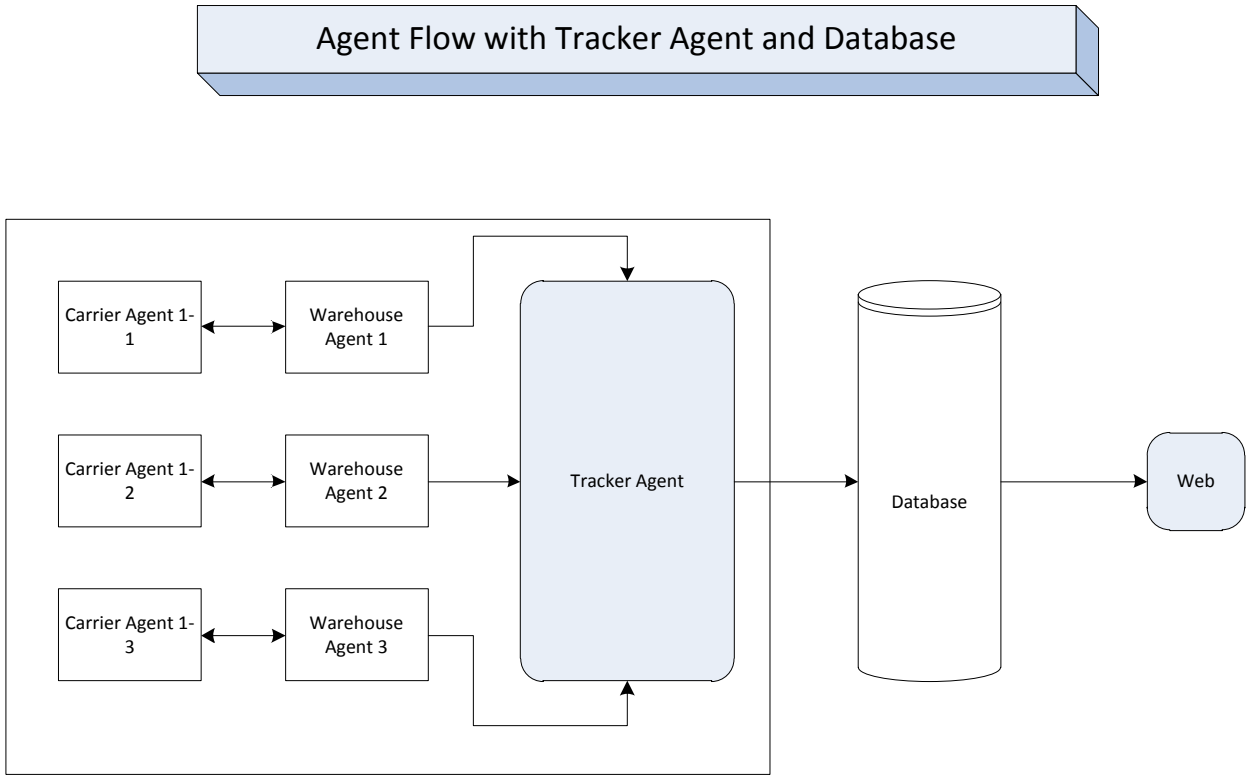

Figure 5c. Data Flow Diagram (DFD) of the tracking application.

Each time a carrier agent reaches a warehouse agent, the carrier agent will have an RFID chip, and the warehouse agent will scan that chip using an RFIF reader. The tracker agent will save the data to the database.

Carrier Agent	RFID	Warehouse Agent	Tracker Agent
1	THRF458F3457GHF	1	1

Table 1. Carrier Agent 1-1 Structure

Below is the program for carrier agent 1, which will communicate with warehouse agent 1 using the tracker agent.

```
package com.tracking.agents;
import com.tracking.singleton.Packages;
import com.tracking.utility.RFIDGenerator;
import com.tracking.valueobjects.RFID;
import jade.core.Agent;//Importing Agent Class from JADE Libraries
import jade.core.behaviours.*;//Importing All Behaviours
import jade.lang.acl.*;//Importing Agent Communication Language Functions
import jade.core.AID;//Importing Agent Identifier Functions
import java.util.*;
/**
*
@author Vindhya Jonnalagadda
*/
public class CarrierAgent1 extends Agent {
@Override
protected void setup() {
// sample package RFIDs.
RFIDGenerator rfidGenerator = new RFIDGenerator();
rfidGenerator.createPackages();
ArrayList<RFID> rfids = Packages.getInstance().getRfids();
Iterator itr = rfids.iterator();
while (itr.hasNext()) {
RFID rfid = (RFID) itr.next();
System.out.println("* " + rfid.getCode());
}
try {
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
// message to be sent.
// Object obj = (Object) rfids;
msg.setContentObject(rfids);

// whom to send.
msg.addReceiver(new AID("WareHouseAgent1", AID.ISLOCALNAME));
// send the message.
send(msg);
addBehaviour(new CyclicBehaviour(this) {
public void action() {
//get messages from RECEIVER, if it sends any.
ACLMessage msg = receive();
if (msg != null) {
System.out.println("\n - " + myAgent.getLocalName() + " received: " +
msg.getContent());
}
block(); // wait for a message to receive.
}
```

```
});
} catch (Exception e) {
System.err.println("ERROR OCCURRED! [DESC: " + e.getMessage() + "]");
}
}
}
```

Figure 6. Carrier agent [1] implementation.

4.3.2. Carrier Agent 1-2

Carrier agent 2 will start to the destination in the middle, and will communicate with warehouse agent 2 and the tracker agent. Each time a carrier agent reaches a warehouse agent, the carrier agent will have an RFID chip and the warehouse agent will scan that chip using an RFID reader. The tracker agent will save the data to the database.

Carrier Agent	**RFID**	**Warehouse Agent**	**Tracker Agent**
2	LKI342CVDWEY	2	1

Table 2. Carrier Agent 1-2 Structure

Below is the program for carrier agent 2, which will communicate with warehouse agent 2 using the tracker agent.

```
package com.tracking.agents;
import com.tracking.singleton.Packages;
import com.tracking.utility.RFIDGenerator;
import com.tracking.valueobjects.RFID;
import jade.core.Agent;//Importing Agent Class from JADE Libraries
import jade.core.behaviours.*;//Importing All Behaviours
import jade.lang.acl.*;//Importing Agent Communication Language Functions
import jade.core.AID;//Importing Agent Identifier Functions
import java.util.*;
/**
 *
 * @author Vindhya Jonnalagadda
 */
public class CarrierAgent12 extends Agent {
    @Override
    protected void setup() {
        // sample package RFIDs.
        RFIDGenerator rfidGenerator = new RFIDGenerator();
        rfidGenerator.createPackages();
        ArrayList<RFID> rfids = Packages.getInstance().getRfids();
        Iterator itr = rfids.iterator();
            while (itr.hasNext()) {
                RFID rfid = (RFID) itr.next();
                System.out.println("* " + rfid.getCode());
            }
        try {
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
            // message to be sent.
            // Object obj = (Object) rfids;
            msg.setContentObject(rfids);

            // whom to send.
            msg.addReceiver(new AID("WareHouseAgent2", AID.ISLOCALNAME));
            // send the message.
            send(msg);
            addBehaviour(new CyclicBehaviour(this) {
                public void action() {
                    //get messages from RECEIVER, if it sends any.
                    ACLMessage msg = receive();
                    if (msg != null) {
                        System.out.println("\n - " + myAgent.getLocalName() + " received:
```

```
" + msg.getContent());
                        }
                        block(); // wait for a message to receive.
                    }
                });
            } catch (Exception e) {
                System.err.println("ERROR OCCURRED! [DESC: " + e.getMessage() +
"]");
                e.printStackTrace();
            }
        }
    }
```

Figure 7. Carrier agent [1-2] implementation.

4.3.3. Carrier Agent 1-3

Carrier agent 3 will start to the destination last, and will communicate with warehouse agent 3 and the tracker agent. Each time a carrier agent reaches a warehouse agent, the carrier agent will have an RFID chip and the warehouse agent will scan that chip using an RFID reader. The tracker agent will save the data to the database.

Carrier Agent	RFID	Warehouse Agent	Tracker Agent
3	RH4533DFRE45G	3	1

Table 3. Carrier Agent 1-3 Structure

Here is the program for carrier agent 3, which will communicate with warehouse agent 3 using the tracker agent.

```
package com.tracking.agents;
import com.tracking.singleton.Packages;
import com.tracking.utility.RFIDGenerator;
import com.tracking.valueobjects.RFID;
import jade.core.Agent;//Importing Agent Class from JADE Libraries
import jade.core.behaviours.*;//Importing All Behaviours
import jade.lang.acl.*;//Importing Agent Communication Language Functions
import jade.core.AID;//Importing Agent Identifier Functions
import java.util.*;
/**
```

```
 *
 * @author Vindhya Jonnalagadda
 */
public class CarrierAgent13 extends Agent {
    @Override
    protected void setup() {
        // sample package RFIDs.
        RFIDGenerator rfidGenerator = new RFIDGenerator();
        rfidGenerator.createPackages();
        ArrayList<RFID> rfids = Packages.getInstance().getRfids();
        Iterator itr = rfids.iterator();
            while (itr.hasNext()) {
                RFID rfid = (RFID) itr.next();
                System.out.println("* " + rfid.getCode());
            }
        try {
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
            // message to be sent.
            // Object obj = (Object) rfids;
            msg.setContentObject(rfids);
            // whom to send.
            msg.addReceiver(new AID("WareHouseAgent3", AID.ISLOCALNAME));
            // send the message.
            send(msg);
            addBehaviour(new CyclicBehaviour(this) {
                public void action() {
                    //get messages from RECEIVER, if it sends any.
                    ACLMessage msg = receive();
                    if (msg != null) {
                        System.out.println("\n - " + myAgent.getLocalName() + " received:
" + msg.getContent());
                    }
                    block(); // wait for a message to receive.
                }
            });
        } catch (Exception e) {
            System.err.println("ERROR OCCURRED! [DESC: " + e.getMessage() +
"]");
            e.printStackTrace();
        }
    }
}
```

Figure 8. Carrier agent [1-3] implementation.

4.3.4. Warehouse Agents

The warehouse agent consumes all of the information fed by the carrier agent, and communicates the package information and the carrier agent information to the tracker agent. This logs the messages when a package goes missing or is reported lost.

```
package com.tracking.agents;
import com.tracking.utility.DatabaseUtility;
import com.tracking.valueobjects.RFID;
import jade.core.Agent;//Importing Agent Class from JADE Libraries
import jade.core.AID;//Importing Agent Identifier Functions
import jade.core.behaviours.*;//Importing all behaviours
import jade.lang.acl.*;//Importing Agent Communication Language Functions
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
import com.tracking.valueobjects.DestinationAddress;
import com.tracking.valueobjects.Message;
import java.text.SimpleDateFormat;
import java.util.Date;
/**
 *
 * @author Vindhya Jonnalagadda
 */
public class WareHouseAgent1 extends Agent {
   static String[] scopeZipCodes = {"91762", "46221", "46203"};
   static ArrayList<String> scopeZipCodesList = new ArrayList<String>();
   static String city = "Los Angeles";
   static String state = "California";
   static String zip = "91762";
   // create agent.
   @Override
   protected void setup() {
      for (String string : scopeZipCodes) {
         scopeZipCodesList.add(string);
      }
      // add behaviour.
      addBehaviour(new CyclicBehaviour(this) {
         // peform the task within this method.
         @SuppressWarnings({"element-type-mismatch",          "element-type-
mismatch"})
```

```
                public void action() {
                   // recieve the messages send by the SENDER agent.
                   ACLMessage msg = receive();
                   java.util.Date utilDate = new Date();
                   SimpleDateFormat sdf = new SimpleDateFormat( "yyyy-MM-dd
HH:mm:ss" );
                   String currentTime = sdf.format(utilDate);
                   if (msg != null &&
msg.getSender().getLocalName().equalsIgnoreCase("CarrierAgent1")) {
                      ArrayList<RFID> rfids = null;
                      try {
                         ArrayList newRFID = new ArrayList<RFID>();
                         newRFID = DatabaseUtility.getRFIDPath("1","1");
                         rfids = (ArrayList<RFID>) msg.getContentObject();
                         for(int i = 0;i<rfids.size();i++)
                            {
                               if(newRFID.contains(rfids.get(i).getCode()))
                               {

                               }
                               else
                               {
                                  if(!DatabaseUtility.isReached(rfids.get(i).getCode()))
                                  {

DatabaseUtility.insertErrorLog(rfids.get(i).getCode(),"Missed Package in " +
rfids.get(i).getAddress().getCity() + " at : " + currentTime + ", Notification sent to
Warehouse facility in Chicago.");
                                     rfids.remove(rfids.get(i).getCode());
                                  }
                               }
                            }
                      }
                         // get rfids with wa1 & ca1

                         // if missing then log

                         // else do nothing

                       catch (UnreadableException ex) {

Logger.getLogger(WareHouseAgent1.class.getName()).log(Level.SEVERE, null, ex);
                      }

                      // System.out.println(" - " + myAgent.getLocalName() + " received: " +
```

```
msg.getContent());
                        Iterator itr = rfids.iterator();
                        int index = 0;
                        while (itr.hasNext()) {
                           RFID rfid = (RFID) itr.next();
                           System.out.println("* " + rfid.getCode());
                           ACLMessage msg1 = new ACLMessage(ACLMessage.INFORM);

                           if (!DatabaseUtility.isReached(rfid.getCode())) {
                              try {
                                 // message to be sent.
                                 // Object obj = (Object) rfids;

                                 Message message = null;
                                 DestinationAddress address = rfid.getAddress();
                                 if (scopeZipCodesList.contains(address.getZip())) {
                                    message = new Message(rfid.getCode(), city,state, zip, 1);
                                    System.out.println("Destination Address reached...for RFID:
" + rfid.getCode());
                                    System.out.println("Removing RFID '" + rfid.getCode() +
"'");
                                    //ArrayList<RFID>                singletonList                =
Packages.getInstance().getRfids();
                                    //singletonList.remove(rfid);
                                    //Packages.getInstance().setRfids(singletonList);
                                 } else {
                                    message = new Message(rfid.getCode(), city,state,zip, 0);
                                 }
                                 msg1.setContentObject(message);

                              } catch (IOException ex) {

Logger.getLogger(WareHouseAgent1.class.getName()).log(Level.SEVERE, null, ex);
                              }
                              // whom to send.
                              msg1.addReceiver(new AID("Tracker", AID.ISLOCALNAME));

                              // send the message.
                              send(msg1);
                           }

                           index++;

                        }
```

```
                    // create reply to the SENDER agent.
                    ACLMessage reply = msg.createReply();

                    reply.setPerformative(ACLMessage.INFORM);

                    // set this content to send
                    reply.setContent("RECEIVED");

                    // send reply to the SENDER agent.
                    send(reply);
                } else if (msg != null &&
msg.getSender().getLocalName().equalsIgnoreCase("Tracker")) {
                    System.out.println("Message received from Tracker: " +
msg.getContent());
                }
                block(); // wait for the message to receive.
            }
        });
    }
}
```

Figure 9. Warehouse agent implementation.

4.3.5. Tracker Agents

The tracker agent consumes all of the information fed by the warehouse agent and updates the central/distributed database with the package information, which includes the time and the current location.

```
package com.tracking.agents;
import com.tracking.utility.DatabaseUtility;
import com.tracking.valueobjects.Message;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TrackerAgent extends Agent {
```

```
        @Override
        protected void setup() {
      // Do something with the Connection
            System.out.println("TrackerAgent has started running.....");

            addBehaviour(new CyclicBehaviour(this) {

               public void action() {
                  ACLMessage msg = receive(); //receive the product information from buyer agent
                  Message message = null;
                  if (msg != null) {
                     try {
                        message = (Message) msg.getContentObject(); //save the product info into BuyerRequest
                        String rfidCode = message.getRfidCode();
                        String currentCity = message.getCurrentCity();
                        String currentState = message.getCurrentState();
                        String currentZip = message.getCurrentZip();
                        java.util.Date utilDate = new Date();
                        SimpleDateFormat sdf = new SimpleDateFormat( "yyyy-MM-dd HH:mm:ss" );
                        String currentTime = sdf.format(utilDate);
                        // Convert it to java.sql.Date
                        int isReached = message.getIsReached();
                        if (isReached == 1) {
                           // update query to update 'is_reached' to 1.
                           // tbl - Packages.
                           DatabaseUtility.updatePackageToReached(rfidCode);
                           DatabaseUtility.updateTracker(rfidCode, currentCity, currentState, currentZip, currentTime);
                        } else {
                           DatabaseUtility.updateTracker(rfidCode, currentCity,currentState, currentZip, currentTime);
                           // add query to add a new row.
                           // tbl - Tracker.
                        }
                     } catch (UnreadableException ex) {

Logger.getLogger(TrackerAgent.class.getName()).log(Level.SEVERE, null, ex);
                     }
                     System.out.println("tracker received: " + message.getRfidCode());
                     try {
                        ACLMessage reply = msg.createReply();
                        reply.setPerformative(ACLMessage.PROPAGATE);
```

```
                        reply.setContent(message.getRfidCode() + "received from " +
myAgent.getLocalName());
                        reply.setSender(new AID("Tracker", AID.ISLOCALNAME));
                        send(reply);//send the product price as reply
                    } catch (NullPointerException npe) {
                    }
                }
            }
        });
    }
}
```

Figure 10. Tracker agent implementation.

4.3.6. Database Utility

When the shipment company initially receives a package from a vendor, an entry will be created in the database with the package id, which is actually stored in the RFID chip after that for future tracking. After that, the package will be placed in the appropriate carrier agent for transportation, and once the carrier agent reaches a warehouse, the warehouse agent reads the package contents held by the carrier agent and updates the tracker agent with the package information, which in turn gets stored in the database by the tracker agent.

```
package com.tracking.utility;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Vindhya Jonnalagadda
 */
public class DatabaseUtility {
```

```
        private static Connection conn = null;

        public DatabaseUtility() {
        }

        public static synchronized void updatePackageToReached(String rfidCode) {
            try {
                conn = MySQLConnect.getConnection();
                Statement s = conn.createStatement();
                s.executeUpdate("UPDATE Packages SET is_reached='1' WHERE
rfid_code = '" + rfidCode + "'");
            } catch (SQLException ex) {
                Logger.getLogger(DatabaseUtility.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }

        public static synchronized ArrayList getRFIDPath(String warehouse_id,String
truck_id) {
            try {
                ResultSet rs;
                conn = MySQLConnect.getConnection();
                Statement s = conn.createStatement();
                ArrayList newRFID = new ArrayList();
                rs = s.executeQuery("select rfid_code from Routing where warehouse_id =
'" + warehouse_id + "'" + " and truck_id = '" + truck_id + "'");
                //rs.last();
                //int numberOfRows = rs.getRow();
                while (rs.next()) {
                 newRFID.add(String.valueOf(rs.getString(1)));
             }
                return newRFID;
            } catch (SQLException ex) {
                Logger.getLogger(DatabaseUtility.class.getName()).log(Level.SEVERE,
null, ex);
                return null;
            }
        }

        public static synchronized void updateTracker(String rfidCode, String
currentCity,String currentState, String currentZip, String currentTime) {
            try {
                conn = MySQLConnect.getConnection();
                Statement s = conn.createStatement();
                s.executeUpdate("INSERT INTO Tracker ( rfid_code,
```

```
current_city,current_state, current_zip, timesnap ) VALUES ( '" + rfidCode + "' , '" +
currentCity + "' , '" + currentState + "' ,'" + currentZip + "', '" + currentTime + "');");
                } catch (SQLException ex) {
                    Logger.getLogger(DatabaseUtility.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
            public static synchronized boolean isReached(String rfidCode) {
                boolean isReached = false;
                try {
                    conn = MySQLConnect.getConnection();
                    Statement                                    s                                    =
conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                            ResultSet.CONCUR_UPDATABLE);
                    ResultSet rs = s.executeQuery("SELECT is_reached FROM Packages
WHERE rfid_code = '" + rfidCode + "'");
                    rs.next();
                    if (rs.getRow() == 1) {
                        int reached = Integer.parseInt(rs.getString("is_reached"));
                        if (reached == 1) {
                            isReached = true;
                        }
                    } else {
                        System.out.println("Either 0 or more than 1 record returned!");
                    }
                    rs.close(); // close result set
                    s.close(); // close statement
                } catch (SQLException ex) {
                    Logger.getLogger(DatabaseUtility.class.getName()).log(Level.SEVERE,
null, ex);
                } finally {
                    if (conn != null) {
                        try {
                            conn.close();
                            System.out.println("Database connection terminated");
                        } catch (Exception e) {
                            /* ignore close errors */
                        }
                    }
                }
                return isReached;
            }

            public static void closeConnection() {
                if (conn != null) {
```

```
                try {
                    conn.close();
                } catch (SQLException ex) {
                    Logger.getLogger(DatabaseUtility.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
        }
    }
```

Figure 11. Database utility class implementation.

4.3.7. SQL Connection Class

This is the class file where we initiate the connection to the central database.

```
    package com.tracking.utility;
    import java.sql.Connection;
    import java.sql.DriverManager;

    public class MySQLConnect {
        public static java.sql.Connection getConnection() {
            Connection conn = null;

            try {
                String userName = "jonnalag";
                String password = "******";
                String hostURL = "jdbc:mysql://obiwan.cs.ndsu.nodak.edu/jonnalag";
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(hostURL, userName, password);
                //System.out.println ("Database connection established");
                return conn;
            } catch (Exception e) {
                System.err.println("Cannot connect to database server");
            }
            return null;
        }
    }
```

Figure 12. MySQL connect class implementation.

4.3.8. JADE Configuration for Boot Class Files

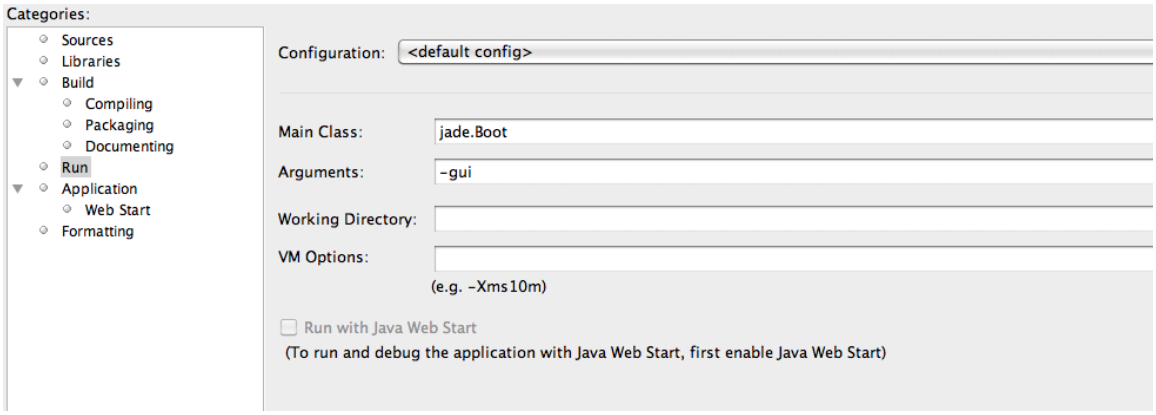

Figure 13. Default communication configuration.

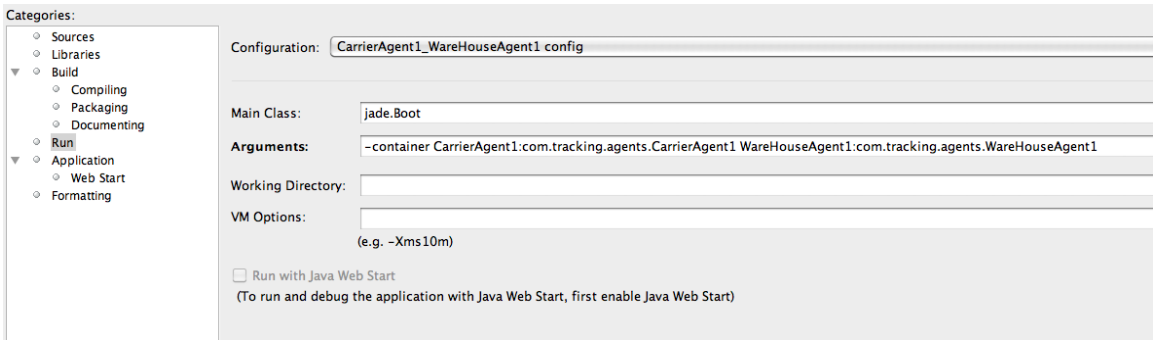

Figure 14. Carrier agent 1-1 with warehouse agent 1 communication configuration.

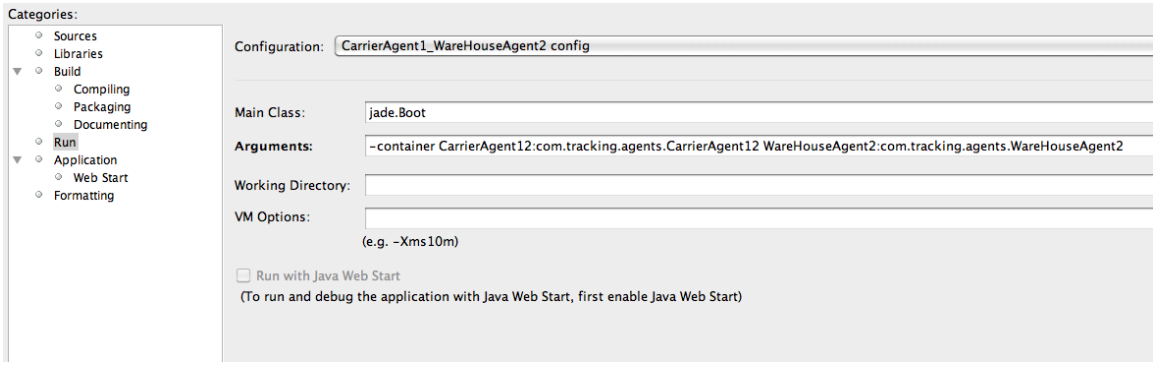

Figure 15. Carrier agent 1-2 with warehouse agent 2 communication configuration.

Figure 16. Carrier agent 1-3 with warehouse agent 3 communication configuration.

We need to select the configuration for running the instance of each class. In order to run the base class and the tracker agent we need to select the default configuration. After running the default configuration and the tracker agent class we need to run the carrier agent and warehouse agent configurations one by one.

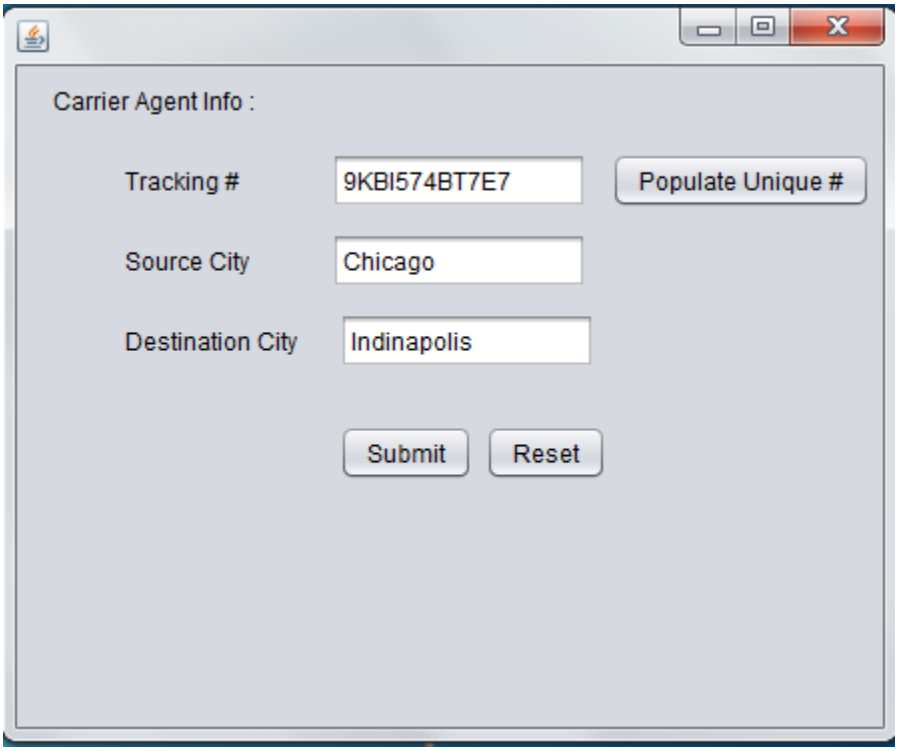

Figure 17. UI for the carrier agent.

```
    private void btnSubmitActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        ACLMessage msg = receive(); //receive the product information from buyer agent
        Message message = null;
        try {

            RFID obj = (RFID) msg.getContentObject();
            obj.setCode(txtTrackingID.getText());
            obj.setSourceAddress(txtSourceCity.getText());
            obj.setDestonationAddress(txtDestinationCity.getText());

        } catch (UnreadableException ex) {
            Logger.getLogger(AgentUI.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    private void btnResetActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void btnGenerateRFIDActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        txtTrackingID.setText(randomString(12));
        //String uuid = UUID.randomUUID().toString().replaceAll("-", "");
       //jTextField1.setText(uuid);
    }

    private void txtTrackingIDActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }


static final String AB = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
static Random rnd = new Random();

String randomString( int len )
{
   StringBuilder sb = new StringBuilder( len );
   for( int i = 0; i < len; i++ )
      sb.append( AB.charAt( rnd.nextInt(AB.length()) ) );
   return sb.toString();
}
```

Figure 18. Code for Java UI.

4.3.9. UI

This is the UI for the Java Application which will take the RFID (Tracking Number) as input or even you can generate a unique RFID. And also there is an ability to set up the source and destination to make this project more comfortable.

5. EXPERIMENTAL RESULTS

This user interface (UI) was designed using PHP. The following figure represents the home page for the package tracking application.

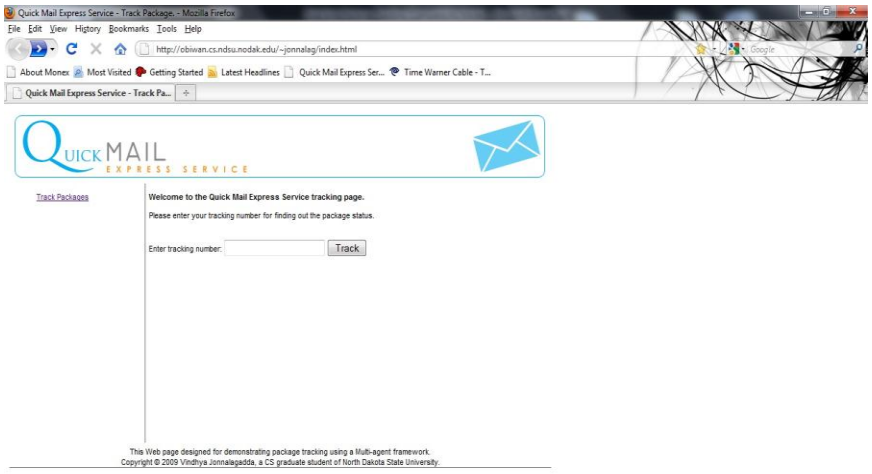

Figure 19. User interface for the tracking application – 1.

We need to enter tracking number in order to get the status of the package.

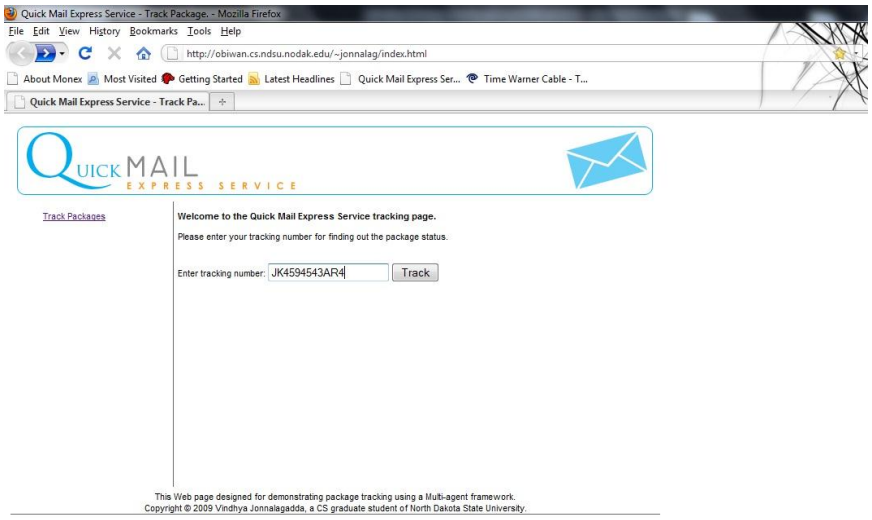

Figure 20. User interface for the tracking application – 2.

After clicking the tracking button, the details will be displayed, including transit center and time.

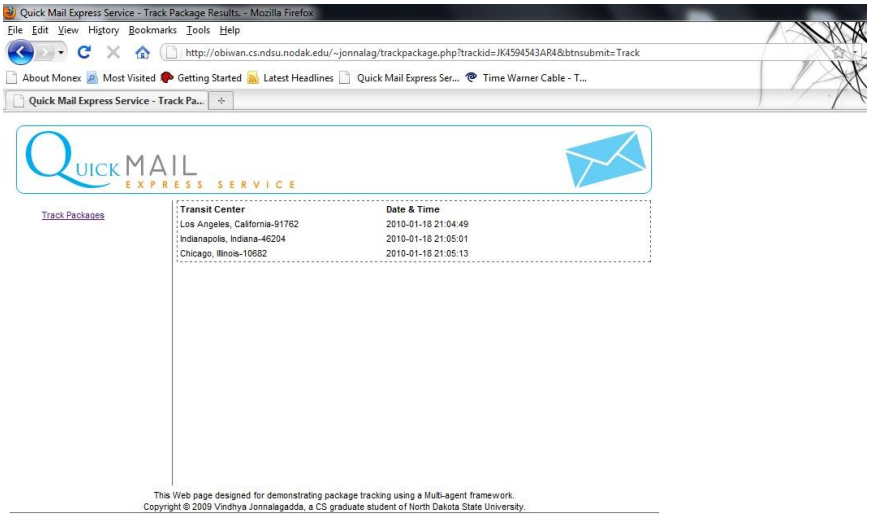

Figure 21. User interface for the tracking results.

We developed the code in order to track missing packages, along with the location, date, and time. A missed package information alert will be sent to the local warehouse facility. The missing information will be logged into the database along with package details.

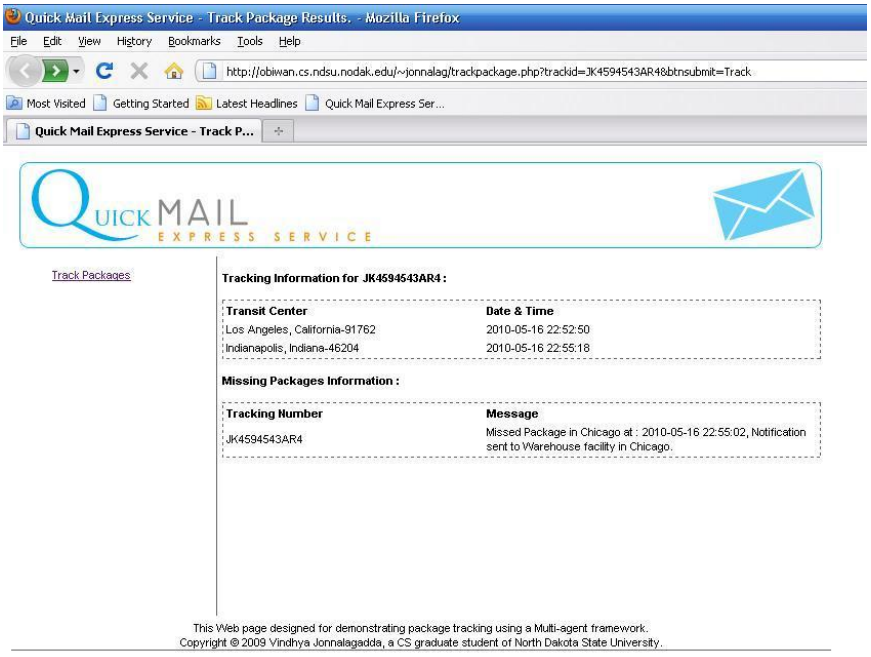

Figure 22. User interface for the missing results.

6. CONCLUSION AND FUTURE WORK

In this paper, it is clearly shown that the software agents can be utilized to manage the communication between various devices by deploying software agents in them. Also, using RFIDs – package tracking can be taken to a different level as it can reduce time and cost by reducing the involvement of human in every step and streamlining the package sensing process through RFID sensors. Moreover having a package travel through a series of conveyer belts which can increase time – just for the sake of tracking purposes. Installation of conveyer belts and human labor can also increase cost.

On the other hand, use of RFIDs can increase the costs, as there is a cost involved in employing each RFID chip; this can be reduced by recycling the RFIDs as the information in each RFID can be erased and rewritten. Package tracking has become increasingly vital as tracking a package with utmost accuracy is needed to ensure there is no delay in delivering packages/goods. The tracking principles can make a difference in customer experience, as the accuracy and ease of tracking can help build trust.

In future, the use of RFIDs for tagging a package can also be used in automatic sorting, by which can reduce time, and which could easily be extended as part of this paper.

7. REFERENCES

[1] Russell, M. “GPS and RFID Package Tracking.” Web, 2011. http://ezinearticles.com/?GPS-and-RFID-Package-Tracking&id=513239.

[2] RFID Report – Barcoding, Inc. Barcodes, RFID, Wireless LANs, Mobile Computer Systems. Web, 2012. http://www.barcoding.com/rfid/.

[3] “Benefits of RFID-Enable d Supply Chain,” in rfid4u. Web, 2012. http://www.rfid4u.com/downloads/Benefits of RFID-Enabled Supply Chain.pdf.

[4] Reed, K, "Featured markets: Packaging / RFID / Barcodes / QR Codes," Graphic Arts Magazine, 2010. http://graphicartsmag.com/articles/2010/07/featured-markets-packaging-rfid-barcodes-qr-codes.

[5] Hassanien, T.H. Ella, and K. Ragab, Developing advanced web services through P2P computing and autonomous agent’s trends and innovations. Hershey, PA: IGI Global, 2010.

[6] H. Wang, Manufacturing intelligence for industrial engineering methods for system self-organization, learning, and adaptation. Hershey, PA: IGI Global, 2010.

[7] F. Bellifemine, C. Giovanni, T. Trucco, and G. Rimassa, Jade programmer’s guide: A white paper, 2010.

[8] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, Java agent development framework: A white paper, 2010.

[9] Nwana, HS, Lee, LC & Jennings, NR, Co-ordination in software agent systems. BT Technology Journal, 14(4), 1996.

[10] "UPS: About UPS." Shipping, Freight, Logistics and Supply Chain Management from UPS. http://www.ups.com/content/us/en/about/index.html.

[11] Nedev D, V. Nedeva, (2008): Aspects of Multi-Agent System Application In E-Learning, International Scientific Conference Computer Science, 2008.

[12] K. Yeh, R. Chen, and C. Chen, Intelligent service-integrated platform based on the RFID technology and software agent system, in ACM Digital Library. 2011.

[13] C.A. Duffy, "UPS toes the line with its package-tracking technologies," PC Week, vol. 10, Jun. 1993.

[14] B. Hermans, "Intelligent software agents on the Internet: An inventory of currently offered functionality in the information society & a prediction of (near-) future 19 developments," Tilburg University, Tilburg, The Netherlands, July 1996.

[15] R.A. Brooks, (1991b), Intelligence Without Reason. AI Memo No. 1293, MIT. AI Lab. In Proceedings of the 12th International Joint Conference on Artificial Intelligence. Morgan Kauffman., San Mateo, CA, 1991.

[16] Brusilovsky, Peter. Methods and techniques of adaptive hypermedia, User Modeling and User Adapted Interaction, v6, n 2-3, 1996.

[17] BA. El Fallah Seghrouchni, M. Dastani, J. Dix, and R. H. Bordini, Multi-Agent Programming Languages, Tools and Applications. Boston, MA: Springer-Verlag US, 2009.

[18] R. M. Carro, A. M. Breda, G. Castillo, and A. L. Bajuelos, "A Methodology for Developing Adaptive Educational-Game Environments," in AH 2002: Adaptive Hypermedia and Adaptive Web-Based Systems Malaga, Spain: Springer, 2002.

[19] "Jade - java agent development framework," 2012, http://jade.tilab.com/.

[20] C. Larman, Applying UML and patterns : an introduction to object-oriented analysis and design and interative development. Upper Saddle River, NJ: Prentice Hall PTR, 2007.

[21] M. Berger, S. Rusitschka, D. Toropov, M.Watzke, and M. Schlichte, “The Development of the Lightweight Extensible Agent Platform,” EXP in Search of Innovation, vol. 3, no. 3, 2003.

[22] K. Inoue, K. Satoh, F. Toni, CLIMA VII, and International Workshop on Computational Logic in Multi-Agent Systems, Computational logic in multi-agent systems : 7th international workshop, CLIMA VII, Hakodate, Japan, May 8-9, 2006.

[23] L. Braubach, J.-P. Briot, and J. Thangarajah, “Programming multi-agent systems 7th International Workshop, ProMAS 2009, Budapest, Hungary, May 10-15, 2009.

[24] M. Pezzini, Do MOM, ORBs and data access middleware suit mobile? Gartner Research Note Number: T-14-3936, 20 September 2001.

[25] D.Lange, M.Oshima. Programming and deploying Java mobile agents with Aglets. Reading, Mass: Addison-Wesley, 1998.

[26] E.Cortese, F.Quarta, and G.Vitaglione, "Scalability and Performance of JADE Message Transport System," presented at AAMAS Workshop on AgentCities, Bologna, 2002.

[27] W.A. Jansen, "Countermeasures for mobile agent security" Computer Communications. Special Issue on Advanced Security Techniques for Network Protection, Elsevier Science, 2000.

[28] J.J. Tan, J.P. Pimentão, S. Poslad, M. Calisti, A.Yip, N. Foukia, R. Jurca, D. Khadraoui, S. Vitabile (2004), Agentcities/Opennet Forum, Security Working Group: Security Requirements for the Agentcities Network, URL:http://www.agentcities.org/out/00023/actf-out-00023a.pdf.

[29] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework" London, New York, Wiley Interscience, vol. 31, 2001.

[30] L. Welling and L. Thomson, PHP and MySQL Web development. Upper Saddle River, NJ: Addison-Wesley, 2009.

[31] A. Sturm, and O. Shehory (2003), A Framework for Evaluating Agent-Oriented Methodologies, Paolo Giorgini, Brian Henderson-Sellers, and Michael Winikoff, The Fifth International Bi-Conference Workshop Agent-Oriented Information Systems, AOIS 2003.