Tender selection for electricity market

1. Abstract

As agent systems become more prevalent, methodologies are needed to help analysts and designers in the creation of new systems. Wood and DeLoach (2000) propose the Multiagent Systems Engineering Methodology (MaSE) as a practical, formal methodology for the creation of multiagent systems. In this project, the same methodology is used to design a software for tender selection for the energy supplier in electricity market.

2. Introduction

2.1 Topic Description

Today, with the modernization of the electricity market, more and more competitors are coming up in every sector of market such as distribution, transmission and generation. Electricity market being such a volatile and unpredictable in nature, the contracts and the necessary backups needed for the smooth operation of the system are decided way before the actual time.

2.2 Methodology and Software Platform

In this project, the methodology used is MaSE. Multi-agent Systems Engineering (MaSE) methodology [3] is a complete methodology for developing heterogeneous multi-agent systems. MaSE covers the complete lifecycle of the system, from the analysis to the design utilizing a number of graphically based models (uses UML – Unified Modelling Language – components). The models are transformed into diagrams in order to describe system agents, their communications, and the internal structure of each agent detailed in depth.

The software platform used for the project is Java. Java is a set of computer software and specifications developed by James Gosling at Sun Microsystems, which was later acquired by the Oracle Corporation, that provides a system for developing application software and deploying it in a cross-platform computing environment. Java is used in a wide variety of computing platforms from embedded devices and mobile phones to enterprise servers and supercomputers. Java applets, which are less common than standalone Java applications, were commonly run in secure, sandboxed environments to provide many features of native applications through being embedded in HTML pages.

Writing in the Java programming language is the primary way to produce code that will be deployed as byte code in a Java virtual machine (JVM); byte code compilers are also available for other languages, including Ada, JavaScript, Python, and Ruby. In addition, several languages have been designed to run natively on the JVM, including Clojure, Groovy, and Scala. Java syntax borrows heavily from C and C++, but object-oriented features are modeled after Smalltalk and Objective-C.[12] Java eschews certain low-level constructs such as pointers and has a very simple memory model where objects are allocated on the heap (while some implementations e.g. all currently supported by Oracle, may use escape analysis optimization to allocating on the stack instead) and all variables of object types are references. Memory management is handled through integrated automatic garbage collection performed by the JVM.

2.3 Integrated Development Environment

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools, and a debugger.

In this project, the IDE used is Eclipse. Eclipse is an integrated development environment (IDE) used in computer programming.[1] It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, JavaScript, Julia,[2] Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Rust, Scala, and Scheme. It can also be used to develop documents with LaTeX (via a TeXlipse plug-in) and packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others.

2.4 Framework

The framework used in the project is JADE. JADE is a proprietary object-oriented software development and deployment platform product from the New Zealand-based Jade Software Corporation, first released in 1996.[3] It consists of the JADE programming language, IDE and debugger, integrated application server and object database management system.

Designed as an end-to-end development environment to allow systems to be coded in one language from the database server down to the clients, it also provides APIs for other languages, including .NET Framework,[4] Java, C/C++ and Web services.

3. JADE

3.1 Introduction

Java Agent Development Framework (JADE) is an agent-oriented tool, implemented in JAVA and FIPA compliant. It is composed of an agent platform (execution environment) and a set of packages which provide the basic support for multi-agent systems construction. JADE [5] can be distributed among several computers and is customizable by means of a remote graphical interface. In addition, it provides the mandatory components (FIPA) for agents management:

- AMS (Agent Management System), which is responsible of providing the white page, agent lifecycle and agents directory services.
- DF (Directory Facilitator), that provides yellow page service
- ACC (Agent Communication Channel), which controls message passing.

In order to create a JADE agent, it is necessary to instantiate a class which extends the Agent class. Communication architecture offers a flexible and efficient message-passing mechanism; for each agent, the framework creates and manages a queue for private messages. The communication paradigm used is based on asynchronous message passing, which follows FIPA ACL standard. Each message is an object of ACLMessage class, and has methods for managing message parameters, for sending and receiving messages, for message filtering, etc.

3.2 WHY IS IT USED AND CRITICISM

Each task identified in role model and detailed in the concurrent task diagrams, has been used as basis for defining the details of each conversation. From the communication class diagrams of both the agent which initiates the conversation and the responder the interaction protocols that must be used were identified. For instance, when a meeting is to be fixed, the User- Interface Agent asks the Informer Agent to identify the users pertaining to a particular group. This communication is made by means of a FIPA-Query protocol. It is implemented by the AskUser/GroupsInitiator classes (used by UserInterface Agent) and AskUser/GroupsResponder (used by Informer Agent). Once this information is available, the UserInterface Agent makes a request for fixing a meeting to its MeetingManager Agent using a FIPA-Request. This is implemented adding a behaviour ProposeMeetingInitiator to UserInterface Agent and the corresponding ProposeMeetingResponder parameter in the MeetingManager Agent. This agent will communicate with the MeetingManager agents of all other users implied in the meeting, in order to negotiate a suitable date using the FIPAContractNet protocol. Next, all the users' personal agendas are updated automatically and the UserInterface agents are notified of these changes by means of a FIPA-Query protocol.

An important issue relating to the JADE platform and models is that some of the developed models introduce information which is not in fact necessary for implementation using the framework. For instance, communication class diagrams demand the achievement of a low abstraction level, specifying all the states the agent passes through during conversation. Meanwhile, conversations in JADE are defined at a higher level, because it adds patterns which implement the standard FIPA interaction protocols.

In the ontology development, the facilities of JADE relating Protégé integration using Bean Generator tool have been exploited. The basic elements of the defined ontology have been:

- Concepts: User, Group, Meeting, MeetingDate.
- Actions performed by agents: Meet, UpdateMeeting and UpdateUserGroups.
- Predicates: ConectedUser and SystemUsers.

4. MaSE Methodology

4.1 Introduction

Multi-agent Systems Engineering (MASE) methodology [6] is a complete methodology for developing heterogeneous multi-agent systems.

MaSE is a top-down methodology made up of seven steps, which covers the analysis and design portions of the software lifecycle. During the seven steps, a total of nine models are produced. The MaSE methodology begins with system requirements as input and ends with deployment as the last step.

1. Capturing Goals
2. Applying Use Cases
3. Refining Roles
4. Creating Agent Classes
5. Constructing Conversations
6. Assembling Agents
7. System Deployment

4.2 WHY IS IT USED AND CRITICISM

MASE provides a step-by-step to approach to multiagent system and design where work products from one step flow into the work products in successive steps. However, characteristics such as autonomy, pro-activity, and learning are virtually ignored by the methodology. As a result, while the objective of MASE was to create a complete agent system development methodology, it is rather limited in its definition of what an agent is and therefore is of limited benefit to actual multiagent system design. Further work and modifications to MaSE need to be done in order for it to be successful in designing agent systems.

As opposed to what normally might be expected, use cases are created in MaSE after the system requirements have been created and after the requirements have been broken down into a task hierarchy. This may lead to some difficulty for the analyst, as use cases are generally at a much higher level of abstraction and would normally precede the creation of goal hierarchy. As a result, use cases could be made to fit into the goal hierarchy. This approach would have a tendency to produce more of a procedural program than a system made up of autonomous agents that react to the environment around them and make decisions in order to achieve their goals. In order for agents to communicate successfully, a common ontology must be shared between them. DiLeo et al (2002) have addressed this issue and proposed the addition of an eighth step “Building Ontology” between the Applying Use Case step and Refining Roles Step, reasoning that the ontology, a necessary part of agent communication, can be defined using the Goal Hierarchy

Diagram, Use Cases, and the Sequence Diagram and is needed to create tasks in the Refining Roles step since tasks often involve parameter passing.

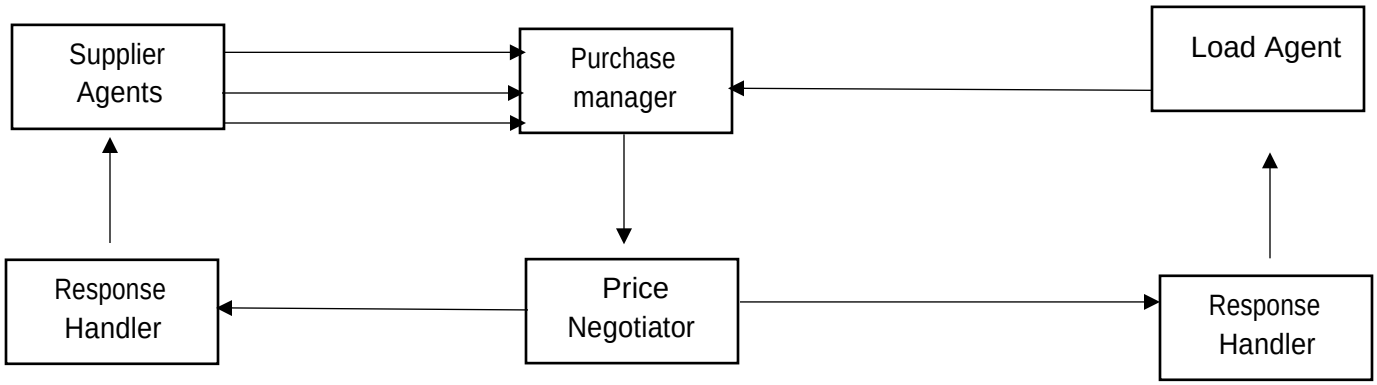
5. Project Description

5.1 Analysis:

In our project, we are creating an application for trading the electricity in the electricity market, where all the suppliers will be able to provide trading price along with a timeline for a particular tender and the buyer group will choose the supplier according to their demand. All the suppliers should communicate properly with the buyer agent and once the buyer opts for a particular supplier, the trading will end.

5.2 Design & Implementation:

In the third phase of the project, we decompose the system functionalities and identify all the required agents for this application.



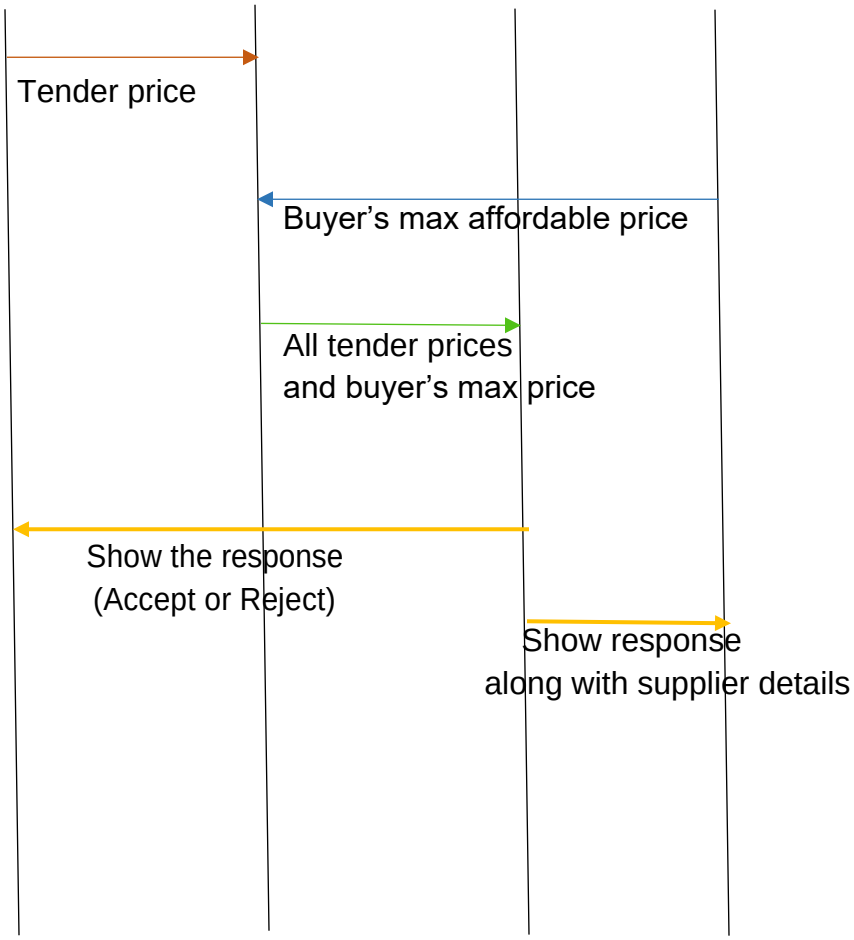
User Case1: Supplier Agent
Class name: ElectricitySupplierAgent
Overview: This agent represents the suppliers for a particular tender. In this class, SLCodec and Ontologies are registered, and GUI is created for each supplier using instance of ElectricitySupplierGuiImp class which implements ElectricitySupplierGui interface. Also, every agents is registered with yellow page service.

User Case2: Load Agent
Class Name: ElectricityLoadAgent
Overview: This agent represents the buyers for a particular tender. In this class, SLCodec and Ontologies are registered, and GUI is created for each supplier using instance of ElectricityLoadGuiImp class which implements ElectricityLoadGui interface. Also, every agents is registered with yellow page service.

User Case3: Purchase Manager
Class Name: PurchaseManager
Overview: This class collects the supplier names, their tender prices, buyer's name, affordable price for buyer, name of the tender and send all the details to Price Negotiator for comparison.

User Case4: Price Negotiator
Class Name: PriceNegotiator
Overview: In this class, we compare the tenders and select the lowest, if that is affordable by the buyer as well, else the buyer will be rejecting every supplier. In case of both supplier coming with same price, the first supplier will be given preference. Also, it handles all the responses and reply to each supplier properly.

5.3 Communication



6 Conclusion

In this project we used the MaSE Methodology and used three different classes to make the application for Tender selection in electricity market. We created an application for trading the electricity in the electricity market, where all the suppliers will be able to provide trading price along with a timeline for a particular tender and the buyer group will choose the supplier according to their demand. All the suppliers should communicate properly with the buyer agent and once the buyer opts for a particular supplier, the trading will end.

REFERENCE:

[1]. *"IDEs vs. Build Tools: How Eclipse, IntelliJ IDEA & NetBeans users work with Maven, Ant, SBT & Gradle". Zeroturnaround.com. Retrieved 28 December 2018.*

[2]. *"GitHub - JuliaComputing/JuliaDT: Julia Development Toolkit for Eclipse". Github.com. 10 October 2018. Retrieved 28 December 2018 – via GitHub.*

[3]. *"Jade – Who We Are". Jade Software Corporation.*

[4]. Scoop Independent News – JADE 6.3

[5]. S.A. DeLoach et al. Multiagent systems engineering. In The International Journal of Software Engineering and Knowledge Engineering, Vol. 11, No. 3, 2001.

[6]. F. Bellifemine et al. Jade Programmer's Guide (JADE 3.1). http://sharon.cselt.it/projects/jade/>, 2003.