ISBN 978-93-5156-328-0
International Conference of Advance Research and Innovation (ICARI-2014)

Agent Unified Modeling Language in Agent Oriented Software Engineering: A Comprehensive Modeling Language Representing Agent Interection Protocol

Deependra Rastogi
Department of Computer Engineering, GRD, Institute of Management and Technology, Dehradun, India

Article Info

Article history:
Received 2 January 2014
Received in revised form
10 January 2014
Accepted 20 January 2014
Available online 1 February 2014

Keywords

Software Engineering
Agent
Object
UML
AUML
Intraction Protocol

Abstract

The concentrate of all earlier period programming experience and innovations for writing high-quality programs in cost effective and proficient ways have been systematically organized into body of knowledge. This comprehension forms the foundation of the software engineering principles. Software engineering discusses methodical and cost effective techniques to software growth. Agent Oriented Software Engineering techniques must be evaluated and compared to gain a better understanding of how Agent should be enginnered and evolved. Unified Modeling Language is a standardized, general purpose modeling language in the ground of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to produce visual models of Object Oreinted Software intensive system. An Agent Unified Modeling Language is an extension of the Unified Modeling Language, a de facto standard for Object—Oreinted analysis and design. AUML is not a language but it is only aproposal. In this paper, I am just presenting the mechanism to model protocol for multiagent interaction. Intraction is driven by interaction protocols.

1. Introduction

For the past decade, study on agent-oriented software engineering had suffered from a lack of handle with the world of engineering software growth. Newly, it has been documented that the use of software agents is improbable likely to gain wide receiving in industry except it relates to de facto standards (object-oriented software development) and supports the growth surroundings throughout the complete system lifecycle.

Fruitfully bringing agent knowledge to market requires techniques that diminish the apparent risk intrinsic in any new technology, by presenting the new technology as an incremental addition of known and trusted methods, and by providing explicit engineering tools to hold up proven methods of skill deployment.

Practical to agents, these insights entail come within reach of that:

- Introduces agents as an addition of dynamic objects*: an agent is an object that can say "go"*(flexible autonomy as the capability to begin action without outside incantation) *and "no"* (flexible independence as the capability to refuse or modify an exterior request)
- Promotes the use of ordinary representations for methods and tools to hold up the analysis, requirement, and design of agent software.

The earlier aspect of our approach leads us to center on quite fine-grained agents. More complicated capabilities can also be supplementary where needed, such as mobility, mechanisms for in lieu of and way of thinking about knowledge, and explicit modeling of other agents. Such capabilities are extensions to our essential agents—we do not reflect on them analytic of agenthood.

To attain the latter, three significant individuality of industrial software growth should be addressed:

1. The extent of engineering software projects is much better than typical educational research hard work, connecting many more populace crossways a longer period of time. Thus, communication is necessary;

Corresponding Author,
E-mail address: deependra.libra@gmail.com
All rights reserved: http://www.ijari.org

2. The skills of developers are listening carefully more on progress tactic than on tracking the latest agent techniques. Thus, codifying best put into practice is essential;
3. Industrial projects have clear achievement criteria. Thus, traceability between initial necessities and the final deliverable is necessary.

The Unified Modeling Language (UML) is ahead wide receipt for the demonstration of engineering artifacts in object-oriented software. Our view of agents as the next step beyond objects leads us to explore extensions to UML and idioms within UML to provide accommodation the individual requirements of agents. To pursue this objective, recently cooperation has been recognized sandwiched between the Foundation of Intelligent Physical Agents (FIPA) and the Object Management Group (OMG).

In this paper, we explain a core part within AGENT UML, i.e., mechanisms to representation protocols for multiagent interaction. This is achieved by introducing a innovative class of diagrams into UML: *protocol diagrams.* Protocol diagrams extend UML state and sequence diagrams in a variety of ways. Exacting extensions in this circumstance consist of agent roles, multithreaded lifelines, extended message semantics, parameterized nested protocols, and protocol templates.

2. Software Specification Technique

AGENT UML is an endeavor to bring jointly research on agent-based software methodologies and emerging principles for object-oriented software growth.

2.1 Methodologies for Agent Based Software Development

2.1.1 Gaia

Gaia is one of the first methodologies which are purposely modified to the analysis and design of agent based system. Its major purpose is to make available the designers with a modeling structure and several connected techniques to design agent leaning systems. Gaia separates the process of designing software into two different stages: analysis and design. Analysis involves construction the theoretical models of the target system, whereas the design point transform those abstract constructed to concrete entities which have direct mapping to implementing code. Figure 1 depicts the main artifacts of each stage: Role Model and Interaction Model (Analysis), and Agent Model, Services Model, and Acquaintance Model (Design). [1]

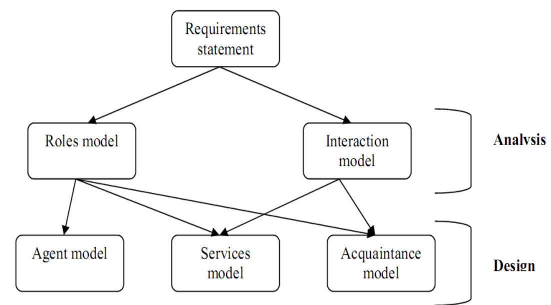

Fig: 1. Gaia Model

Gaia encourages the developers to view an agent based system as an organization. The software system association is similar to a real world organization. It has a certain number of entities in performance different roles. For instance, a university organization has several key roles such as organization, teaching, research, students, etc. These roles are played by dissimilar people in the university such as managers, lectures, students, etc. inspired by that analogy, Gaia guides the designers to the direction of building agent-based as a process of organizational design.[1]

2.1.2 Multiagent Systems Engineering (MaSE)

Multiagent System Engineering (MaSE) [2, 3] is an agent-oriented software engineering methodology which is a lean-to of the object-oriented. MaSE does not outlook agents as being essentially autonomous, proactive, etc; quite agents are "simple software processes that interact with each other to meet an overall system goal."[3]. In fact they view agents as specializations of objects which may have some of the individuality of weak agency. In addition, all the mechanism in the system are uniformly treated in spite of whether they posses cleverness or not. Because of this inherent point of view, MaSE is constructed according to the request of existing object-oriented method to the analysis and design of multiagent systems.[3]

As a software engineering methodology, the main goal of MsSE is to provide a complete—lifecycle methodology to assist system developers to design and develop a multi—agent system. Similar to Gaia, it also assumes the availability of an initial requirement prior specification to the start of software development under the methodology process. The process consists of seven steps, divided into two phases. The Analysis phase consists of three steps: Capturing Goals, Applying Use Cases, and Refining Roles. The remaining four process steps, creating Agent Classes, Constructing Conversations, Assembling Agent Classes and System Design, from the Design phase (Figure 2).

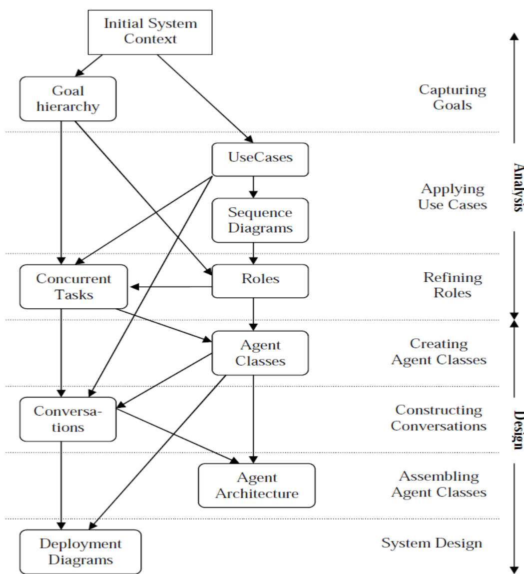

Fig: 2. MaSE's process step and artifacts [2]

The wide range of activity in this area is a sign of the increasing impact of agent-based systems, since the demand for methodologies and artifacts reflects the growing commercial importance of agent technology. Our objective is not to compete with any of these efforts, but rather to extend and apply a widely accepted modeling and representational formalism (UML) in a way that harnesses their insights and makes it useful in communicating across a wide range of research groups and development methodologies.

3. UML

Object-Oriented Software Engineering (OOSE) is a software design technique that is used in software design in object-oriented programming. OOSE is developed by **Ivar Jacobson** in 1992. OOSE is the first object-oriented design methodology that employs use cases in software design. OOSE is one of the precursors of the Unified Modeling Language (UML), such as Booch and OMT. It includes a requirement, an analysis, a design, an implementation and a testing model.[4]

UML support the following kind of models.

- **Use cases:** the measurement of actions that a system or class can execute by interacting with outside actors. They are frequently used to describe how a customer communicates with a software product.
- **Static models:** explain the static semantics of information and messages in a theoretical and implementational way (e.g., class and package diagrams).
- **Dynamic models***:* consist of interaction diagrams (i.e., sequence and collaboration diagrams), state charts, and activity diagrams.
- **Implementation models:** explain the component distribution on different platforms (e.g., component models and deployment diagrams).
- **Object constraint language (OCL):** a simple formal language to articulate more semantics within an UML specification. It can be used to describe constraints on the model, invariant, pre- and post-conditions of operations and direction-finding paths within an object net.

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose. Because other four diagrams (activity, sequence, collaboration and Statechart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. [5]

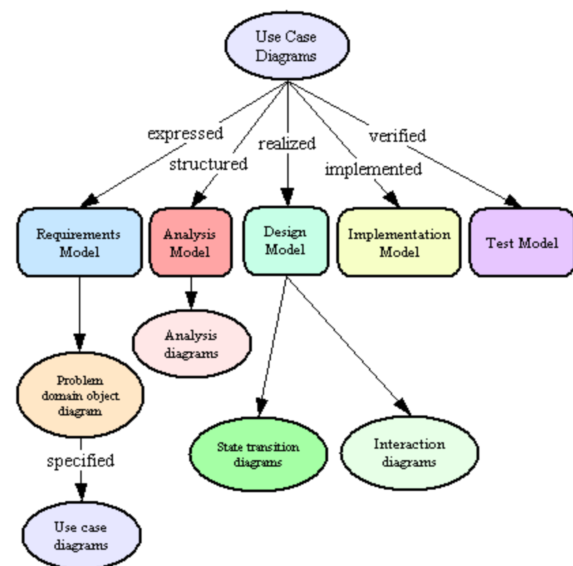

Fig: 3. UML Model Diagram [4]

In this paper, we offer agent-based extensions to three following UML representations: packages, templates, and sequence diagrams. This consequences in a new diagram type, called *protocol diagram*, and which will be measured for addition into UML

version 2.0 by OMG. The UML model semantics are represented by a meta-model the construction of which is also formally defined by OCL syntax. Extensions to this meta-model and its constraint language are not addressed by this paper.

4. A rationale for AGENT UML

UML provides an inadequate foundation for modeling agents and agent-based systems [Bauer, 1999]. Essentially, this is due to two reasons: *Firstly*, compared to objects, agents are dynamic because they can take the proposal and have organized over whether and how they progression external requests. *Secondly*, agents do not only act in separation but in collaboration or synchronization with other agents. Multiagent systems are social communities of inter-reliant members that act individually.

An application for a full life-cycle requirement of agent-based system growth is beyond the extent for this paper. In this paper, we will focus on a division of an agent-based UML lean-to for the requirement of *agent interaction protocols (AIP)*.

This subset was selected because AIPs are compound enough to exemplify the nontrivial use of and are used frequently enough to make this subset of AGENT UML helpful to other researchers. AIPs are a precise class of software design patterns in that they explain problems that occur regularly in multiagent systems and then explain the core of a reusable solution to that problem.

The explanation of interaction protocols is part of the requirement of the dynamical model of an agent system. In UML, this model is captured by interaction diagrams, state diagrams and activity diagrams.

- **Interaction diagrams** i.e. sequence diagrams and collaboration diagrams are used to describe the performance of groups of objects. Typically, one interaction diagram captures the performance of one use case. These diagrams are principally used to define basic communications between objects at the level of method incantation; they are not well-suited for recitation the types of complex social communication as they occur in multiagent systems.
- **State diagrams** are used to representation the performance of a absolute system. They define all probable states an object can arrive at and how an object's state changes depending on communication sent to the object. They are well suitable for defining the performance of one single object in dissimilar use cases. However, they are not suitable to describe the performance of a group of cooperating objects.
- **Activity diagrams** are used to describe courses of events/actions for more than a few objects and use cases. The work reported in this paper does not suggest modifications of activity diagrams.

5. A Layered Approach to Agent UML Intrection Protocol

The explanation of an agent interaction protocol (AIP) describes

- A communication pattern, with
- An allowed sequence of messages between agents having different roles,
- Constraints on the content of the messages, and
- A semantics that is consistent with the communicative acts (CAs) within a communication pattern.

Figure 4 depicts a protocol articulated as a UML *sequence diagram* for the agreement net protocol. When invoked, an Initiator agent sends a call for suggestion to an agent that is eager to contribute in providing application. The Participant agent can then choose to respond to the Initiator before a given time limit by: refusing to provide a suggestion, submitting a suggestion, or indicating that it did not recognize. (The diamond symbol indicates a decision that can result in zero or more transportation being sentdepending on the circumstances it contains; the "x" in the decision diamond indicates an *exclusive or* decision.) If a suggestion is obtainable, the initiator has a choice of either accepting or rejecting the proposal. When the contributor receives a suggestion acceptance, it will inform the initiator about the proposal's execution. Furthermore, the Initiator can cancel the execution of the proposal at any time.

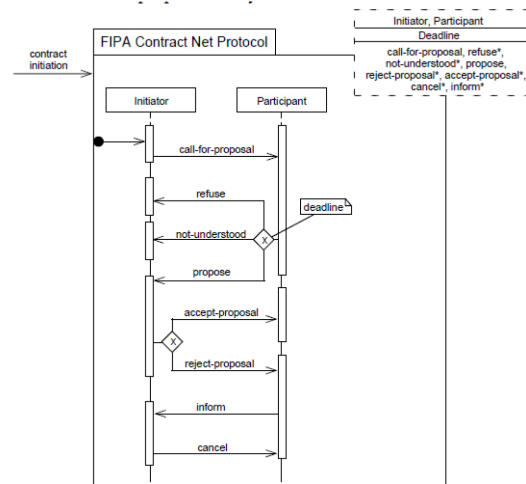

Fig: 4. A generic AIP expressed as a template package

This figure also articulte two more concepts presented at the top of the sequence chart. First, the protocol as a whole is treated as an entity in its own right. The tabbed folder notation at the upper left give direction that the protocol is a *package*, a theoretical aggregation of communication sequences. Second, the packaged protocol can be practics as a pattern that can be modified for analogous problem areas. The dashed box at the upper right hand corner denoted this pattern as a *template* requirement that identifies unbound entities within the tie together which need to be bound when the package template is individual instantiated.

The unique sequence diagram in Fig. 4 give a basic arrangement for a contract net protocol. More dealing out detail is often required. For example, an Initiator agent requirements a call for proposal (CFP) from a Participant agent. However, the diagram stipulates neither the process used by the Initiator to produce the CFP request, nor the process employed by the participant to respond to the CFP. Yet, such details are significant for developing detailed agent-based system stipulation.

Figure 5 explain how *leveling* can provide more detail for anyinteraction procedure. For example, the procedure that generated the communication act CA-1 could be compound enough to specify its dealing out in more detail using an activity diagram. The agent receiving CA-1 has a progression that prepares a response. In this example, the process being particular is depicted using a sequence diagram, though any modeling language could be selected to further identify an agent's fundamental process. In UML, the choice is an interaction diagram, an activity diagram, or a statechart.

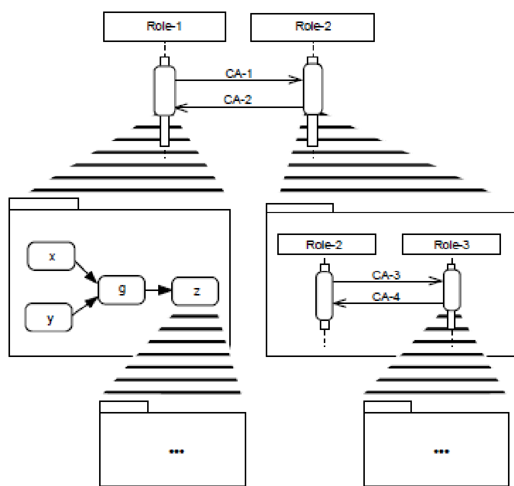

Fig: 5. Interaction protocols can be specified in more detail (i.e., leveled) using a combination of diagrams.

Finally, leveling can persist "down" until the difficulty has been specified sufficiently to develop or construct code. So in Fig. 5, the interaction protocol at the top of the diagram has a level of point below, which in turn has another level of detail. Each level can express *intraagent* or *interagent* activity.

6. Level 1: Representing The Overall ProtocoL

Patterns are thoughts that have been establish useful in one realistic background and can almost certainly be useful in others. As such, they give us examples or analogies that we strength use as solutions to problems in system analysis and design. Agent interaction protocols, then, make available us with reusable solutions that can be applied to various kinds of message sequencing we come across between agents. There are two UML techniques that best articulate protocol solutions for reuse: packages and templates.

6.1 Packages

Since interaction protocols are patterns, they can be treated as reusable aggregates of dealing out. UML describes two ways of expressing aggregation for OO structure and behavior: *components* and *packages*. Mechanisms are physical aggregations that make up classes for completion purposes. Packages aggregate modeling elements into theoretical wholes. Here, classes can be theoretically grouped for any subjective purpose, such as a subsystem grouping of classes. Since AIPs can be viewed in conceptual terms, the package notation of a tabbed folder was employed in Fig.

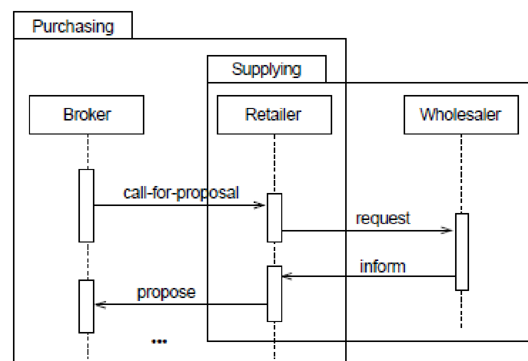

Fig: 6. Using packages to express nested protocol

Because protocols can be codified as identifiable patterns of agent communication, they become reusable modules of processing that can be treated as first-class notions. For example, Fig. 3 depicts two packages. The Purchasing package expresses a simple protocol between a Broker and a Retailer. Here, the

Broker sends a call for proposal to a Retailer and the Retail responds with a proposal. For certain products, the Retailer might also place a demand with a Wholesaler regarding accessibility and cost. Based on the return in order, the Retailer can provide a more precise proposal. All of this could have been put into a single Purchasing Protocol package. Though, many businesses or departments may not need the supplementary protocol connecting the Wholesaler. Therefore, two packages can be defined: one for Purchasing and one for Supplying. When a exacting scenario requires the Wholesaler protocol, it can be nested as a divide and distinct package. However, when a Purchasing scenario does not require it, the package is thriftier.

6.2 Templates

Figure 4 illustrates a universal kind of performance that can serve as explanation in analogous problem domains. In Fig. 6, the Supplying performance is reused precisely as distinct by the Supplying package. However, to be really a pattern—instead of just a reusable component—package customization must be supported. For example, Fig. 4 applies the FIPA agreement Net Protocol to a particular scenario connecting buyers and sellers. Notice that the Initiator and Participant agents have become Buyer and Seller agents and the call-for-proposal has developed into the seller-rfp. Also in this situation are two forms of rejection by the Seller: Refuse-1 and Refuse-2. Lastly, an actual time limit has been supplied for a response by the seller.

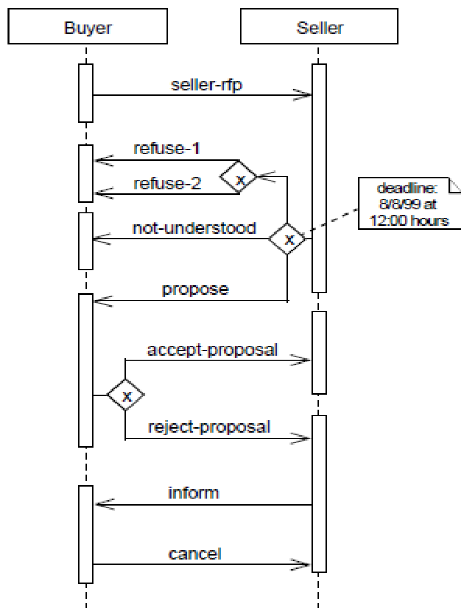

Fig: 7. Applying the template in Fig. 1 to a particular scenario involving buyers and sellers.

In UML jargon, the AIP package serves as a *template*. A template is a parameterized model building block whose parameters are bound at model time (i.e., when the new customized model is produced). In Fig. 4, the dotted box in the upper right indicates that the package is a pattern. The unbound parameters in the box are alienated by horizontal lines into three categories: role parameters, constraints, and communication acts. Figure 8 illustrates how the latest package in Fig. 4 is twisted using the template definition in Fig. 4 Wooldridge et al. suggest a similar form of definition with their *protocol definitions* [8]. In their packaged templates "a pattern of interaction . . . has been officially defined and abstracted away from any exacting sequence of implementation steps. Presentation interactions in this way mean that concentration is concentrating on the necessary nature and purpose of interaction rather than the accurate ordering of particular message exchanges." Instead of the notation illustrated by Wooldridge et al., our graphical move toward more closely resembles UML, while expressing the same semantics.

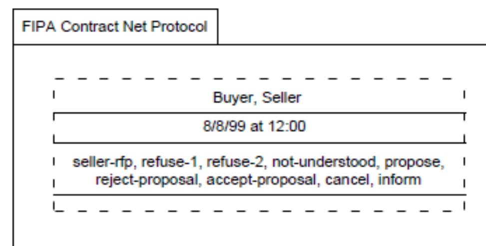

Fig: 8. Producing a new package using the Fig. 4 template; Fig. 4 is the resulting model.

7. Level 2: Representing Interection Among Agents

UML's *dynamic models* are useful for expressing communications among agents. *Interaction diagrams* capture the structural patterns of communications among objects. Sequence diagrams are one affiliate of this family; collaboration diagrams are another. The two diagrams contain the same in sequence. The graphical layout of the sequence diagram emphasizes the chronological sequence of communications, while that of the teamwork diagram emphasizes the relations among agents. *Activity diagrams* and *statecharts* capture the flow of processing in the agent area.

7.1 Sequence Diagram

A brief description of sequence diagrams using the example in Fig. 4 appeared above. (For a more detailed discussion of sequence diagrams, see

ISBN 978-93-5156-328-0
International Conference of Advance Research and Innovation (ICARI-2014)

Rumbaugh [24] and Booch [3].) In this section, we discuss some possible extensions to UML that can also model agent-based interaction protocols. Figure 6 depicts some basic elements for agent communication. The rectangle can express individual agents or sets (i.e., roles or classes) of agents. For example, an individual agent could be labeled Bob/Customer. Here Bob is an instance of agent playing the role of Customer. Bob could also play the role of Supplier, Employee, and Pet Owner. To indicate that Bob is a Person—independent of any role he plays—Bob could be expressed as Bob: Person. The basic format for the box label is agent-name/role: class. Therefore, we could express all the various situations for Bob, such as Bob/Customer: Person and Bob/Employee: Person.

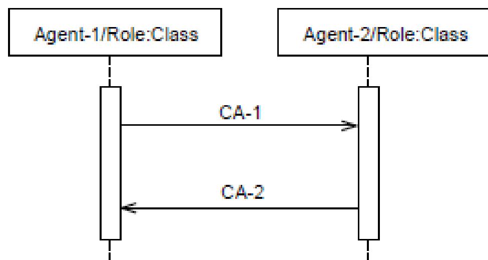

Fig: 9. Basic formats for agent communication

The rectangular box can also indicate a universal set of agents playing a precise role. Here, just the word Customer or Supplier would come into view. To identify that the role is to be played by a definite class of agent, the class name would be appended (e.g., Employee: Person, Supplier: Party). In other words, the agentname/ role: class sentence structure is used without specifying an individual agent-name.

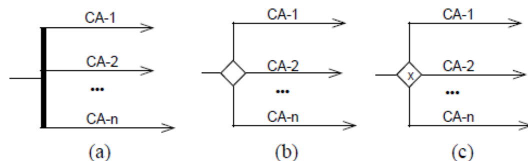

Fig: 10. Some recommended extensions that support concurrent threads of interaction.

The agent-name/role: class syntax is before now part of UML (apart from that the UML syntax indicates an object name in its place of an agent name). Figure 8 extends UML by classification the arrowed line with an agent *communication act* (CA), as an alternative of an OO-style *message*.

Another optional extension to UML supports synchronized threads of interaction. While synchronized threads are not forbidden in OO, they are not frequently employed. Figure 9 depicts three ways of expressing multiple threads. Figure 9(a) indicates that all threads CA-1 to CA-n are sent at the same time as. Figure 9(b) includes a decision box representative that a decision box will make a decision which CAs (zero or more) will be sent. If more than one CA is sent, the interaction is concurrent. In short, it indicates an *inclusive or*. Fig. 9(c) indicates an *exclusive or*, so that accurately one CA will be sent. Figure 7(a) indicates and communication.

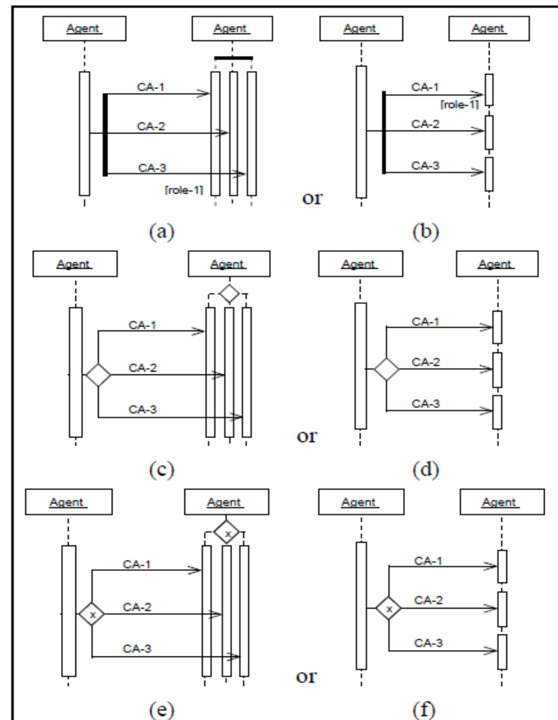

Fig: 11. multiple techniques to express concurrent communication with an agent playing multiple roles or responding to different CAs.

Figure 11 illustrates one way of using the synchronized threads of interaction depicted in Fig. 10. Figures 11(a) and (b) portray half each of expressing synchronized threads sent from agent-1 to agent-2. The numerous vertical, or *activation*, bars point to that the in receipt of agent is dispensation the various communication threadsconcurrently. Figure 10(a) displays parallel foundation bars and Fig. 11(b) establishment bars that appear on top of each other. A few things should be noted about these two variations:

- The semantic connotation is equivalent; the choice is based on ease and clarity of visual manifestation.
- Each commencement bar can indicate either that the agent is using a dissimilar role or that it is simply employing a diverse processing thread to

ISBN 978-93-5156-328-0
International Conference of Advance Research and Innovation (ICARI-2014)

support the communication act. If the agent is using a different role, the commencement bar can be annotated suitably. For example in Figs. 11(a) and (b), Can is handled by the agent under its role-1 processing.

- These figures indicate that a single agent is at the same time as dispensation the multiple CAs. However, the simultaneous CAs could *each* have been sent to a *different* agent, e.g., CA-1 to agent-2, CA-2 to agent-3, and so on. Such protocol performance is already supported by UML; the notation in Fig. 9, on the other hand, is a recommended extension to UML.

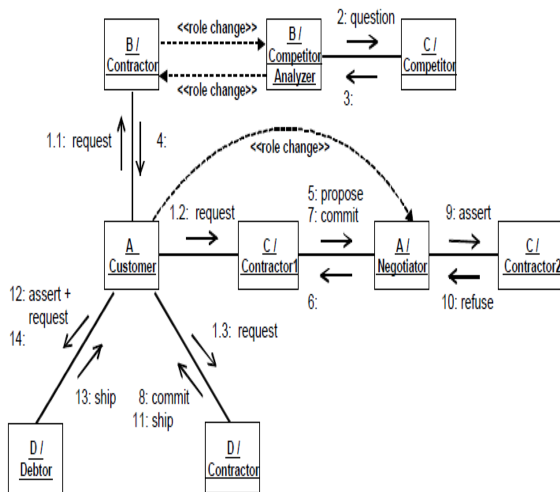

Fig: 12. An example of a collaboration diagram depicting an interaction among agents playing multiple roles.

7.2 Collaboration Diagrams

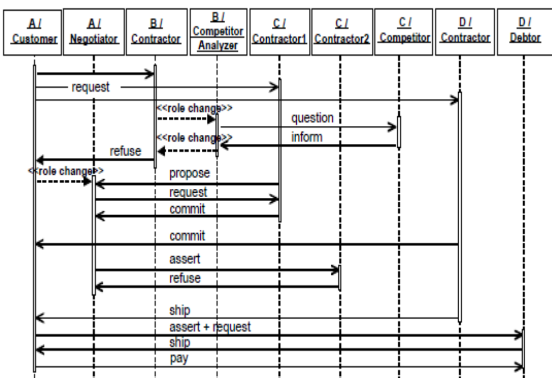

Fig: 13. A sequence diagram version of Fig. 11

Figure 12 is an example of a collaboration diagram and depicts a pattern of communication among agents. One of the primary distinctions of the collaboration diagram is that the agents (the rectangles) can be placed anyplace on the diagram; while in a sequence diagram, the agents are situated in a horizontal row at the diagram's top. The sequences of communications are numbered on the Collaboration diagram; whereas the communication diagram is basically read from the top down. If the two interaction diagrams are so similar, why have both? The answer lies principally in how clear and comprehensible the appearance is. Depending on the person and communication protocol being described, one diagram type might provide a clearer, more comprehensible representation over another. Semantically, they are corresponding; graphically they are similar. For example, Fig. 11 expresses the same fundamental meaning as Fig. 10 using the sequence diagram. Experience has established that agent-based modelers can find both types of diagrams useful.

7.3 Activity Diagrams

Agent interaction protocols can occasionally require stipulation with very clear processing-thread semantics. The *activity diagram* expresses operations and the events that trigger them. The example in Fig. 14 depicts an order dispensation protocol among several agents. Here, a Customer agent places an order. This procedure results in an Order placed event that triggers the broker to place the order, which is then conservative by an Electronic Commerce Network (ECN) agent. The ECN can only connect an order with a citation when both the order and the Market Maker's quote have been recognized. Once this occurs, the Market Maker and the Broker are at the same time as notified that the trade has been completed. The activity diagram differs from interaction diagrams because it provides an unambiguous thread of control. This is predominantly useful for complex interaction protocols that involve simultaneous processing.

Activity diagrams are alike in nature to colored Petri nets in more than a few ways. First, activity diagrams make available a graphical demonstration that makes it possible to create in your mind processes simply, thereby facilitating the design and communication of behavioral models. Second, activity diagrams can represent concurrent, asynchronous processing. Lastly, they can express concurrent communications with several correspondents. The primary dissimilarity between the two approaches is that activity diagrams are officially based on the comprehensive state-machine model defined by UML [24]. Ferber's BRIC formalism [8] extends Petri nets for agents-based systems; this paper extends UML activity diagrams for the same purpose.

ISBN 978-93-5156-328-0
International Conference of Advance Research and Innovation (ICARI-2014)

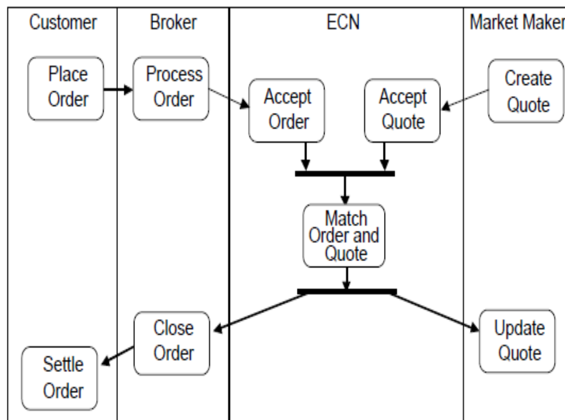

Fig: 14. An activity diagram that depicts a stock sale protocol among several agents.

7.4 Statecharts

Another process-related UML diagram is the *statechart*. A statechart is a graph that represents a state machine. States are represented as round-cornered rectangles, while transitions are normally rendered by heading for arcs that be linked the states. Figure 15 depicts an example of a statechart that governs an Order protocol. Here, if a given Order is in a Requested state, a supplier agent may entrust to the requested negotiation—resulting in a transition to a dedicated negotiation state. Furthermore, this diagram indicates that an agent's Commit action may occur only if the Order is in a Requested state. The Requested state has two other likely actions besides the Commit: the supplier may refuse and the customer may back out. Notice that the supplier may refuse with the order in either the Proposed or the Requested states.

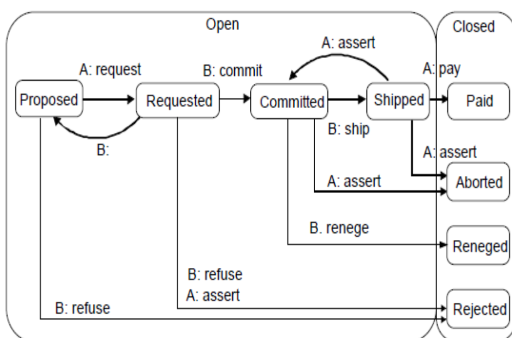

Fig: 15. A statechart indicating the valid states and transitions governing an Order protocol.

The statechart is not normally used to express interaction protocol for the reason that it is a state-centric view, rather than an agent- or process-centered view. The agent-centric outlook portrayed by interaction diagrams emphasizes the agent first and the interaction second. The process-centric view emphasizes the process flow (by agent) first and the consequential state change (i.e., event) second. The state-centric view emphasizes the allowable states more significantly than the changeover agent processing. The most important strength of the statechart in agent interaction protocols is as a restriction mechanism for the protocol. The statechart and its states are characteristically not implemented in a straight line as agents. However, an Order agent could embody the state-transition constraints, thereby ensuring that the overall interaction protocol constraints are met. Alternatively, the constraints could be personified in the supplier and customer roles played by the agents involved in the order process.

8. Level 3: Representing Internal Agent Processing

At the lowest level, requirement of an agent protocol used spelling out the thorough giving out that takes place within an agent in order to put into practice the protocol. In a holarchic representation, higher-level agents (holons) consist of aggregations of lower-level agents. The internal performance of a holon can thus be described by means of any of the Level 2 representations recursively. In addition, state charts and activity diagrams can also identify the internal dispensation of agents that are not aggregates, as illustrated in this section.

8.1 Activity Diagrams

Figure 16 depicts the detailed dispensation that takes place within an Order Processor agent. Here, a succession diagram indicated that the agent's process is triggered by a Place Order CA and ends with the order finished. The internal dispensation by the

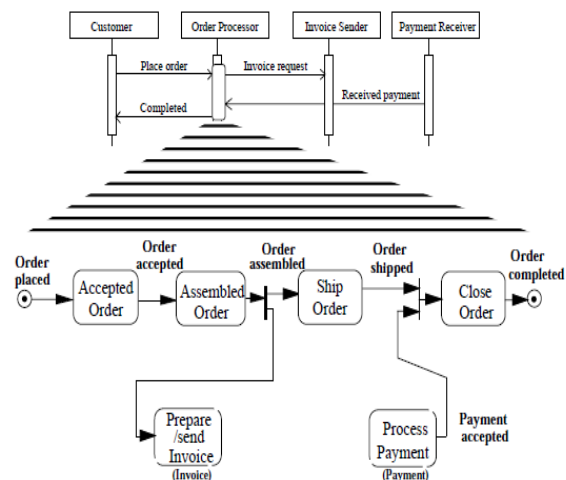

Fig: 16. An activity diagram that specifies order processing behavior for an Order agent

OrderProcessor is articulated as an activity diagram, where the OrderProcessor accepts, assembles, ships, and closes the order. The dotted operation boxes correspond to interfaces to processes carried out by external agents—as also illustrated in the sequence diagram. For example, the diagram indicates that when the order has been assembled, both Assemble Order and Prepare/send Invoice actions are triggered concurrently. Additionally, when both the payment has been established and the order has been shipped, the Close Order process can only then be invoked.

8.2 Statecharts

The internal dispensation of a single agent can also be articulated as statecharts. Figure 17 depicts the interior states and transitions for Order Processor, Invoice Sender, and Payment Receiver agents. As with the activity diagram above, these agents interface with each other—as indicated by the dashed lines. This intra-agent use of UML statecharts supports Singh's notion of agent skeletons [10].

9. Conclusions

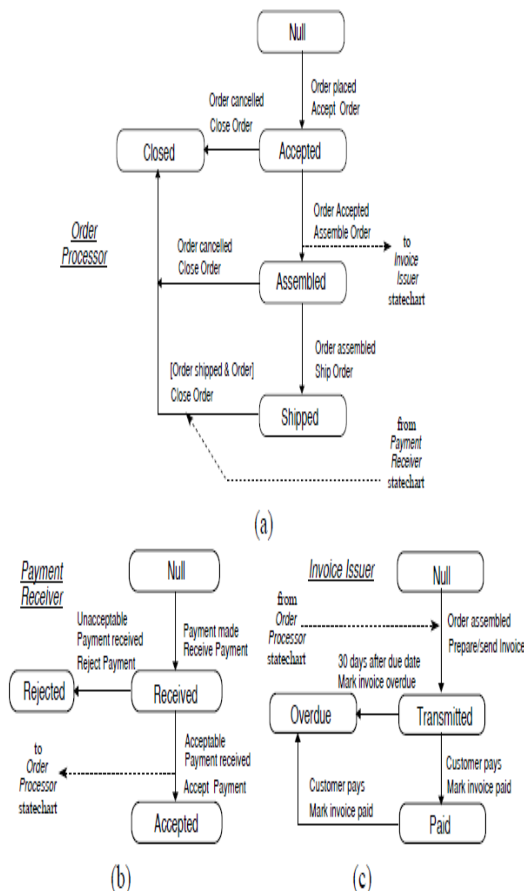

Fig: 17. Statechart that specifies order processing behavior for the three agents

UML provides tools for specifying agent interaction protocols at multiple levels:

- Specifying a protocol as a whole, as in [9];
- Expressing the interaction pattern among agents within a protocol, as in [1, 8, 11]; and
- The internal behavior of an agent, as in [10].

Some of these tools can be sensible directly to agent-based systems by adopting simple idioms and conventions. In other cases, we suggest several clear-cut UML extensions that support the additional functionality that agents offer over the current UML version 1.4. Many of these proposed extensions are before now being measured by the OO community as useful extensions to OO growth on UML version 2.0. Furthermore, many of the AUML notions presented here were developed.

Agent researchers can be gratified at the increasing attention that industrial and business users are paying to their results. The transfer of these results to practical application will be more rapid and accurate if the research community can communicate its insights in forms consistent with modern industrial software practice. AUML builds on the acknowledged success of UML in supporting industrial-strength software engineering. The idioms and extensions proposed here for AIP's—as well as others that we are developing—are a contribution to this objective.

References

[1] M. Wooldridge, N. R. Jenning, D. Kenny, “The Gaia Methodology for Agent-Oriented Analysis and Design”

[2] Mark F. Wood, Scott A. Deloach, “An Overview of multiagent systems Engineering Methodology”

[3] Scott A Deloach, “Analysis and Design using MaSE and AgentTool”

[4] http://cs-exhibitions.uni klu.ac.at/index.php?id=448

[5] http://www.tutorialspoint.com/uml/uml_use_case_diagram.htm

[6] Rumbaugh, James, Ivar Jacobson, and Grady Booch, *TheUnified Modeling Language Reference Manual*, Addison-Wesley, Reading, MA, 1999

[7] Parunak, H. Van Dyke, and James Odell, Engineering Artifacts for Multi-Agent Systems, ERIM CEC, 1999.

[8] Burmeister, B., ed., Models and Methodology for Agent-Oriented Analysis and Design 1996

[9] Wooldridge, Michael, Nicholas R. Jennings, and David Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," International Journal of Autonomous Agents and Multi-Agent Systems, 3:Forthcoming, 2000

[10] Singh, Munindar P., ed., Developing Formal Specifications to Coordinate Heterogeneous Autonomous Agents IEEE Computer Society, Paris, FR, 1998.

[11] Parunak, H. Van Dyke, ed., Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis 1996