A Generic Model for Negotiation Behaviors in Multi-Agent Engineering Applications

Moath Jarrah [1], and Bernard Zeigler [2]

[1] Department of Computer Engineering
Jordan University of Science and Technology
mjarrah@just.edu.jo

[2] Department of Electrical and Computer Engineering
The University of Arizona
zeigler@ece.arizona.edu

Abstract

In this paper, we will present a generic Domain-Independent Marketplace architecture that allows user agents to interact with service providers using two simple and yet powerful negotiation protocols. Service providers have different capabilities depending on the domain of interest. Hence, a dynamic message structuring capability is needed. A key role to support such an expressive power is to design an ontology that contains specializations between different domains. Integrating of the Domain-Dependent Ontology with the Domain-Independent marketplace gives the designer a powerful tool in which systems can be tailored based on the operational purposes.

Keywords: Negotiation Protocol, Marketplace, FD-DEVS, DEVS/SOA, SES, Ontology Design, Semantic Web, Language of Encounter, Oceanography

1. Introduction

The ability to manage and exploit geographically distributed systems of service providers is rather limited in today engineering solutions. The complexity results from the fact that there are many aspects and factors that represent the characteristics of these systems, such as node bandwidth, job processing deadline, the execution time. The user decision of whether to use a computing service or not is based on these factors. Many researchers and other parties have tried to provide solutions to exploit these resources efficiently [46] [47]. However, until now the development of methods to exploit geographically distributed information storages and computing resources has been very limited. Existing techniques [16] suffer from three main issues: first, current techniques cannot provide brokering in managing loosely coupled service providers. Second, the engineering design of existing management tools does not provide enough expressive capabilities for varying user behaviors or when different domains are encountered. Third, lack of interaction between different requestors and providers yields inefficient and very costly agreements. Also one main issue in collaborative distributed multi-agent environments is providing privacy and transparency to their agents.

Distributed environments are seldom static. Everyday more and more service providers are added to the system in order to provide more capabilities as the users grow in numbers and needs. This leads to the diversity in resources and data availability which adds new challenges to the management techniques that systems use. Hence, a manual management is not feasible in such a community because of the number of service providers and the heterogeneity in their information management. All of the above issues make discovering the "Best Match" for satisfying user requirements a tedious task.

Web Services developments are growing dramatically nowadays and millions of resources are being added every day to the World Wide Web. The success in e-commerce, e-learning, online auctions, online marketplaces, information discovery and retrieval has encouraged more and more companies to provide Web Services either to satisfy customer requirements or to manage their distributed computing resources. In order to reach to a successful framework design, the following issues must be supported:

- The system should provide brokering and negotiation services to its users.
- The system should provide transparency to its users.
- New service providers should be able to join the community in a simple and efficient way.
- The system should provide decision making capabilities on behalf of the agents whenever the user agents need it.
- The system should provide varying negotiation capabilities under different domains.
- The system must provide rich expressive negotiation primitives to its users to provide them with the capabilities to express their requirements to be able to use the system under different domains.
- The design of the system must be simple to shorten the development time on the system designer under a specific domain of interest.

In this paper, we will develop a generic negotiation model based on the marketplace concept that can be utilized by different engineering domains. The model defines different concepts and principles in the negotiation process. Our method consists of designing a Domain-Independent Marketplace architecture that allows user agents to interact with service providers using two simple and yet powerful negotiation protocols which define the rules of interactions in multi-agent environments. Having a

trusted third party marketplace supports privacy and transparency among collaborative agents and service providers. Service providers have different capabilities depending on the domain of interest. Such providers can be Radar sensors as in oceanography surveillance systems, print servers in distributed printing jobs community, or they can be online stores providing products on the Web in the e-commerce domain. In order to provide negotiation in different domains, a dynamic message structuring capability is needed. A key role to support such an expressive power is to design an Ontology that contains specialization relations between the different domains of interest.

The System Entity Structure (SES) methodology, which is a formalism to define hierarchical relations between entities, is used to build the required message structures Ontology. The architecture design of the Marketplace suggests different phases and functionalities which are mapped and implemented using the Discrete Event System Specifications (DEVS). DEVS/Service Oriented Architecture (DEVS/SOA) is used to validate our system and show a proof of the concept by deploying models of printing jobs in a web-services multi-server environment for printing server domain.

The paper is organized as follows. Section 2 gives a background and some research in the literature. In section 3, the system design method and purposes are introduced. Then the system implementation under different domains is shown in section 4. To provide a proof of the concept and validate the system, we provide the DEVS/SOA implementation in section 5. Finally section 6 concludes our work.

2. Background and Related Work

2.1 Negotiation Systems

The negotiation process is an interaction between two or more parties in an attempt to reach some agreement on a specific aspect. This aspect could be an idea as in e-learning, or a price of some goods as in e-commerce, or information availability and data provision. Hence, a multi-criterion negotiation system is needed that supports dynamic structures based on the domain of interest [40]. During the negotiation process, web-based agents exchange their capabilities, such as the services they provide, offers, counter offers, speed, bandwidth, goods, ideas, topics or computation power. The result can be an agreement or disagreement. In either case, the result depends on the interest of the agents and their achievement of profit. A negotiation agent needs to be flexible enough to act under different kinds of situations because negotiation is a dynamic activity by nature. The process is dynamic in the sense that it involves: asking for an item or service, discovering item/service providers, negotiating with sellers/service providers, proposing counter offers, decision making upon the receiving of some offers, and then acceptance or rejection of an offer. The agent needs also to make sure that he does not go into an infinite cycle of negotiation.

Negotiation activity in multi-agent environments is an iterative behavior in which agents negotiate by exchanging Offer-CounterOffer messages. G. O'Hare and N. Jennings in their book on Distributed Artificial Intelligence [8] classified the research in negotiation into three main categories: negotiation language, negotiation decision and negotiation process. Our research interests fall into the first category. The negotiation language category consists of negotiation protocols, negotiation primitives, semantics and object structure. Protocols refer to polices or rules that agents must follow during their interactions with other agents. Primitives refer to the messages that are exchanged between the agents. Negotiation primitives (messages) can be placed into three groups: *initiators* such as "request", *reactors* such as "respond", and *completers* such as "accept". The semantics give more explanation and meaning to the *language of encounter* (primitives) that is being used in negotiation protocols. The semantics capabilities are usually achieved by building an ontology which classifies primitives based on measurements of similarity. So, for example, one can consider the two primitives "Request" and "Query" to be equivalent. Some tools are used in order to help in computing measurements of similarities such as WordNet [41], which gives synonyms and acronyms of a given term based on the semantic meaning. In this work, the semantics are not part of our work because it is not a necessary factor for system completeness and methodology. The object structure refers to the structure of each of the primitives during the interaction between different agents, which defines what type of information a message can carry.

Current bidding and auction systems do not provide the flexibility to negotiate on parameters chosen by the users. They consider the price as the only parameter that in which users are interested. For example, eBay [28] and Amazon Auctions [33] require from the bidders that they locate an exact item and bid on it based only on its price. The bidding is a committed action, which means that if a bidder wins, he has to buy it. This discourages users of the system to bid on more than one item because they do not want to end up buying many items when they only need one [29].
Priceline.com [31] is an airline booking auction where a user selects his flight information (source, destination, traveling date, and returning date). And then the user bids by entering a specific price. Priceline searches its database to find a ticket price that is lower than the bidder price. If a ticket is found, then the bidder will get the ticket. This scenario of negotiation has drawbacks which are summarized as in follows:

1. If bidding is accepted, then the bidder is required to purchase the ticket.
2. The bidder cannot control other information on the flights such as waiting time in the airport, and number of stops on the way.
3. The system takes advantage of users who do not have the knowledge and experience about ticket prices. A bidder might enter a high bidding price for a cheap ticket.
4. It prevents the user from paying a little more money for a more comfortable flight.

Our objective in this research is to support negotiation capabilities over more than one dimension. We can have as many constraints as it needs. A user can choose different criteria to be considered in addition to the price; for instance, how many stops, Airline Company, period of the negotiation, and so on. The dynamic structure of the language of encounter makes this possible.

2.2 Negotiation in Multi-Agent Environments

In most of business and engineering distributed systems, managing the resources and services manually is impossible and autonomous agents are needed to act on behalf of the system users. Negotiation process is methodology that was applied to these systems to provide bargaining and brokering capabilities between different agents in multi-agent environments. Such agents are not just capable of making decisions in predictable situations, but also they need to be intelligent enough to act in any dynamic unpredictable interaction. The agents need to communicate with each other, share data and ontologies and negotiate with other agents to reach some agreements. For instance, a user can use search agents over the Web to search for a specific data or information and once the appropriate data provider is found, the data will be sent to the user.

Game theory is a branch of economics that is concerned with interactions between agents [21] [24] [25]. It imposes mathematical models (functions) that describe each agent utility function in multi-agent systems, and strategically try to maximize each individual preference. Under some domains, the mathematical function of an agent is formatted to take into account other opponents and coordinators utility functions. However, the game theory is limited by the assumption of having the knowledge about other players (agents) preferences. Negotiation probably is one of the most frequent domains in which game theory principles have been applied. Negotiation environments use game theory in order to model the decision making process in the negotiating agents; which can give insights into the computation of the search space in order to analyze different interaction strategies.

Martin J. Osborne and Ariel Rubinstein in [23] discussed variant sequential models in applying game theory to bargaining. The models have a sequential structure in the sense that each player makes decisions sequentially in a pre-specified order. The order reflects the rules or policies of the negotiation (negotiation protocol and rules of encounter). At all times, the negotiators care about time to protect themselves from going into an infinite cycle of offers and counter offers. No agreement is forced on any agent. This means that if all agents who are involved in a specific negotiation cycle chose to accept the terms in that negotiation cycle, then the agreement will take place, otherwise the agreement fails.

K. Binmore and N. Vulkan [26] used a simple mathematical formula to describe the decision making process. They modeled the agreement by using two real numbers a_1 and a_2. Player 1 (or buyer) and player 2 (or seller) keeps some reserved value for the item they are bargaining about [$r(1)$ and $r(2)$ respectively]. These values are kept hidden from each other (player 1 does not inform player 2 about his reserved value and visa versa). If player 1 proposes a price m (the amount of money) for the item, then $a_1 = r(1) - m$ and $a_2 = m - r(2).$ The agreement succeeds if both a_1 and a_2 are positive numbers (≥ 0). As mentioned earlier, both agents should approve the acceptance of the agreement terms and this model guarantees that.

V. Krishna and VC. Ramesh in their work on market games and their applications used a negotiation model based on coalition partners [21] [27]. The player agent chooses a set of agents to form a coalition. Then it uses probability profiles of the chosen agents to compute the payoffs resulting from using different strategies by simulating the actual bargaining. Next, it computes the probability distribution among the whole set of agent strategies. Using the payoff metrics, it arranges the strategies (solutions) on a priority basis where the solution that gives the maximum payoff has the highest priority.

Negotiation also enables coordination among agents to enhance performance in multi-agent systems where all agents aim to improve the overall system performance. In this context, Mahajan et al. [24] attempted to resolve selfishness routing in multi-hop networks, where Internet Service Providers try to lower the traffic they forward (route) by either dropping packets or sending them through the closest link which results in longer paths. The system sends anonymous messages in which the sender ID is hidden. If the recipient node cheated by not forwarding the messages correctly, all the neighbors isolate the cheating node from the network. The cost here for a cheating node is that it will be punished by disconnecting it so neighbors will not forward or receive message to or from it.

In competitive negotiation, each agent tries to maximize his own utility function (maximize his satisfaction) regardless of the other agents. However, in cooperative negotiation, an agent is concerned about other agents and he needs to compromise his own preferences for the good of community satisfaction. Archibald et al. [22] used a strategic-form game by evaluating the utilities of all players to reach a negotiation solution that is mutually acceptable. It does not have to be the maximum for each agent but good enough that all players are satisfied.

3. System Design

3.1 Ontology Design

An ontology is an information model that describes concepts and relations in some specific domain. Ontologies enable the processing and sharing of knowledge among different computing sites on the web [20]. Hence, ontologies are known to be the representation of a shared conceptualization of a specific domain. They provide a common understanding of a domain that can be communicated across people and applications. They have been also developed in Artificial Intelligence to facilitate knowledge representations and sharing. Ontology has a hierarchical structure of classes and concepts in the domain of interest and it describes different relations between concepts. Also, it provides a description of concepts through the use of an attribute-value mechanism. Many domains have started to develop and build their ontologies like VnHIES [43] and geographic applications [44]. In this work, we designed an ontology to define the language of encounter under different domains of interest.

3.2 Negotiation Process and the Protocols Design

We aim to provide a generic negotiation model that can be utilized under different engineering applications. The design of the negotiation protocols refers to rules that agents must follow during their interactions with other agents [8]. We will explain our design of the negotiation protocols and describe our approach for designing the negotiation primitives and the object structure. The semantics are not addressed as we mentioned earlier. Also, we will show how the marketplace architecture can help the negotiation parties in reaching agreements efficiently.

3.2.1 One-to-One Negotiation Protocol

Many system designers have applied the negotiation process to different domains. Based on the objectives of the systems, different types of negotiation are developed such as: collaborative environments, buyer and seller negotiation, negotiations for resources and data reservations and so on. Murugesan [11] discussed different issues concerning automated negotiation for electronic commerce. Feng et al [7] were trying to apply collaborative negotiation activity for e-commerce where different threads (parties) are independent from each other. They used a constraint network to measure conflict costs for collaborative negotiation and a state diagram to model the negotiation protocol. In all negotiation systems, agents must follow some rules of interaction known as "Negotiation Protocols". These protocols define how parties can interact with each other which in turn affect their decision and expressiveness capabilities. In most current e-commerce solutions, the conflict is related to the price of the items between the sellers and the buyers [65].

One important property of the negotiation process is a One-to-One protocol. In this protocol, negotiating parties can communicate with each other via offers and counter offers cycles. The process starts when the requestor sends a *Request*, then the provider replies with either, *Accept* where an agreement is established, *Reject* where no agreement has been reached or *Offer* where requestor needs to make evaluation upon receiving it whether to accept it or reject it. If the requestor response on the *Offer* was *Accept*, then an agreement has been reached; if he replies with *Reject* there is no agreement. The third choice is to reply with a *Counter-Offer* message. The cycles can go on forever. However, in real life and software developments, a predefined time is allowed before the termination of the process. In papers [6], [14] and [42] a simple negotiation protocol is used. It occurs between two agents to support a shared semantic ontology of the terms and primitives that can be used in the negotiation process. Figure 1 shows the One-to-One protocol nature.

Figure 1 shows some primitives along with a sequence of negotiation protocol (rules). However, it does not reveal details on the syntax involved in using these primitives nor does it show semantic specifications. Such a simple scenario is limited to the situation where the interacted agents know each other IDs. In many cases, however, more sophisticated protocols are needed to support dynamic behaviors during the negotiation process. For instance, the buyer might not know what services or products are available for him or if the buyer wants to complain about a transaction. Hence, a more flexible and comprehensive negotiation model is required. Transparency and privacy are other requirements that negotiation models need to support.

The objective of Bailin and Truszkowski's research [2] on scientific archives was to find relevant information on a specific topic. Again the One-to-One negotiation protocol has been used as one rule between agent A and agent B (two parties trying to negotiate on scientific archives information). Masvoula et al. [12] discuss the issue on how a negotiation model should be as close as possible to the real interactions in auctions and the bargaining behaviors in the stores. The protocol design is assumed to be One-to-One with offers, evaluation of offers and counter-offers. Research in e-learning needs different expressive requirements than other domains like the e-commerce. In a collaborative e-learning domain, the negotiators will ask questions, answer questions, confirm information, etc. [9]. The objective here is to reach an agreement and one understanding on topics or ideas. Although the negotiation primitives are different, they used a One-to-One negotiation protocol.

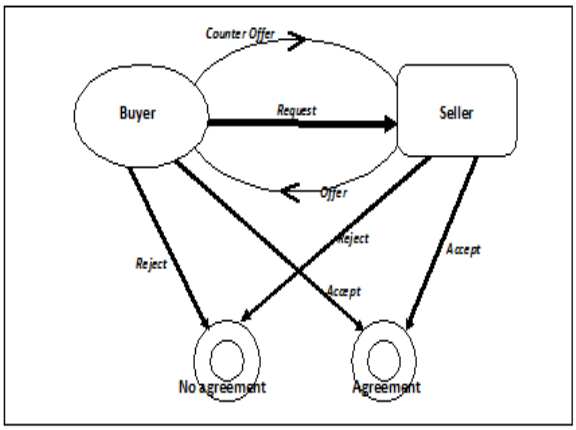

Figure 1: Simple sequence of negotiation activities

The work by M.Addis, P.Allen and M.Surridge on negotiation for software services used the One-to-One Negotiation protocol to support on-demand software and hardware resources sharing environments [16]. V. Krishna and VC Ramesh proposed a

model for competitive decision making agents where they used the One-to-One protocol to support the bargaining process when agent "a" calls agent "b" [10].

Because of the nature of the One-to-One negotiation model which models real life bargaining and negotiation behaviors, we are adopting this protocol for its simplicity and advantages. However, some modifications are needed concerning its definition to fit into our generic framework.

3.2.2 Service Discovery Protocol

Most of the current negotiation systems and distributed services management tools do not support brokering between agents. Distributed services environments do not interact with their users on different specifications. For example, if a user would like to use a service that is deployed on a distributed environment and that service is already being used, he will get usually a response that it is not available at this time. Then the user needs to request that service maybe every 1 minute. On the other hand and in some other extreme cases he might receive a decline that he cannot use this environment because of a simple error he made or because the service does not provide one of the job specifications he asked for in his request. In some case it might be that the user can ignore one of the specifications because the execution of his job will satisfy his needs. In such case brokering and negotiation is very essential to reach agreements in distributed computing environments.

Yilmaz and Paspuleti [19] used a *Broker* agent to support transparency, a *Matchmaker* to bring different views using relevance metrics that are independent of keyword matching, and a *Mediator* agent to convert contents to some common reference model (constructed as ontology) that negotiators understand. Tamma et al. [18] used a shared ontology to model the protocols that could be encountered or needed in supporting agent negotiation in e-commerce environment. According to their paper, the agents do not go into different states or decision making phases. However agents query the shared ontology for the next step in response of an event occurrence. Such an implementation is slow and lacks scalability requirement. Also, it is a single point of failure implementation with unmanageable size of ontology when the ontology grows up to handle more space of dynamic behaviors. Bailin and Truszkowski [2] used marketplace architecture to resolve semantic mismatches in real time without human intervention. The protocol they used is a One-to-One protocol between for example agent A and agent B. However, the definition and functionality is different because they exchange different types of primitives that need different ways of handling.

In order to support flexible generic negotiation protocols that can capture different user behaviors, we determined the following requirements that we believe the negotiation system should support them:

1. The ability to ask a service provider (might be a computing node) for a service or an offer.

2. The ability to negotiate with the service provider over jobs/products specifications (e.g. execution time for a job, bandwidth of data transformation after the job is finished).

3. The customer ability to respond with a counter offer that represents its interests.

4. The ability to complain to a third party that controls web services. This is important in order to support customer satisfaction over the web to build a trustable environment between requesters and providers.

5. The service provider ability to advertise their capabilities to be found later when customers search for services. This provides transparency and privacy between users and providers. For example a user might request a book with a specific ISBN number; the result will be all bookstores that have that specific book available. As a result, the user will negotiate with the best book provider that meets his interests.

6. Supporting decision making capabilities for customers to choose the best among different service providers.

7. Monitoring agreements which were achieved between customers/users and the service providers. For example, under the domain of information discovery and retrieval, a third party is needed to monitor the transfer of data from the data provider to the requester according to the agreement guidelines that both agreed upon during their negotiation.

8. The ability to reformat or add/remove some parameters to the messages being exchanged between customers and service providers to avoid misunderstanding or confusion and sometimes for future tracking purposes.

Supporting these requirements is not an easy task because of the dynamic nature in multi-agent environments. However, the choice of using marketplace architecture (a third party such as a controller or mediator agent) is useful in this regard, in addition to the choice of a new negotiation protocol namely "*Service Discovery*". The marketplace can act as a broker, a mediator, a controller and/or a database for service providers to advertise their products/services.

The service discovery protocol is needed when a customer is searching for a specific service or product. Customers then query the marketplace to find the best providers. The marketplace responds to the customers with a group of service providers who fulfill their requests. After the customers get the results back from the marketplace, they will have a list of the available service providers and their capabilities. Then the customers can decide on whether to proceed with the negotiation process or not. If the customers choose to proceed, they send a contract query to the marketplace searching for their preferable service provider.

3.2.3 Language of Encounter Taxonomy and Structure

In our design of the negotiation model, we specified the language of encounter that our system users need to use for their interactions. O'Hare and Jennings [8] suggest three groups for language of encounter. However, such a classification is not enough to truly enable the negotiation process in loosely coupled distributed environments. More types are needed to support customer satisfaction and the dynamic in user behaviors. We have defined two new necessary classes for the messages to increase the expressiveness power and the negotiation capabilities.

Table 1 shows the language of encounter (messages) to which agents can use to express their needs. The difference between "*Decline*" and "*Reject*" is when the marketplace is too busy and cannot handle more requests. It might not choose to start ("*Decline*") the negotiation process. Note that the negotiation did not take place in this situation, which allows the requester to try to start the same negotiation later. However, "*Reject*" means that the negotiation process already took place and the result is no agreement, so there will be no point of trying again later to establish the same negotiation process under the same parameters. On the other hand, "*NotMet*" refers to situations where the two negotiation parties have come to an agreement and they started the transaction. However, one of the two parties has violated the agreement terms that both established before. In such a situation, the marketplace needs to stop or terminate the transaction. For example: if an information agent negotiates with a service provider to transfer some audio traffic with a minimum speed of 200KB and the service provider agrees on that, and subsequently, after establishing the link between them, the service provider was transferring the traffic with speed less than 200KB, then the information agent can ask the marketplace to terminate this contract by sending "*NotMet*". "*Terminate*" message means that the negotiation process has started but is not finished (still in progress and the result is not known yet). Then the requester has the right to stop the negotiation.

The last point to address in our research design of the negotiation language is the object structure. The structure of each message type depends on the domain under which it is being used. Hence, in order for our system to support negotiation services under different domains, a dynamic message structure is the key role to the success. In this implementation we used a shared Ontology that defines each message and its usage under different domains. Each domain would be a specialization in the message structure. Each Message type has a separate structural ontology defining its variables/fields. For example, Oceanography is a subdomain of the domain surveillance and has specific structure. Online store is a subdomain of the domain e-commerce and has a specific structure. Figure 2 shows the purpose of designing ontology for a specific primitive, "MessageX".

The alternatives here are that if there are two domains defined for MessageX ontology, then two cases might occur:

1. Given the input is "MessageX the subdomain Oceanography", then the system should automatically select the message structure to be sub SES 1 represented in the variables (Location, Altitude, Speed and Roughness).
2. Given the input is "MessageX and the subdomain Online Store", then the system should automatically select the message structure to be sub SES 2 represented in (SellerID, BuyerID, Price, S&H and Return).

Abort	Initiators	Reactors	Completers	Informative
Terminate	ContractQuery	Offer	Reject	Busy
NotMet	CapabilityQuery	CounterOffer	Accept	LinkEstablished
	ItemRequest	Decline		Item
		CapabilityStatement		ItemCheckResult
				BestProvider
				ProvidersChosen

Table 1: Classification of the language of encounter

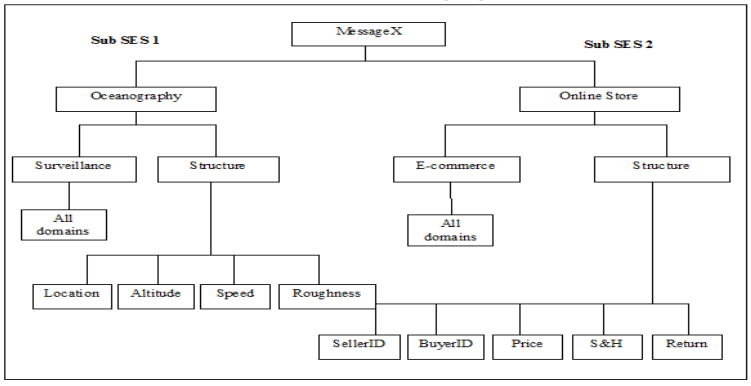

Figure 2: Ontology design for MessageX type

3.2.4 Domain-Independent Marketplace Architecture

The Marketplace controls the behavior of the interacting agents by enforcing our model rules and policies (negotiation protocols). Here we show the marketplace architecture design which is based on finite state model. Table 2 shows the different states of the marketplace along with their descriptions. Figure 3 shows the transitions between phases of the marketplace agent. This model of the marketplace can be translated easily into an FD-DEVS implementation. However, because of the specifications of FD-DEVS, some phases need to be reformatted according to the messages that cause the transition to these states. The marketplace enforces two negotiation protocols which are:

1. One-to-One negotiation when entities know each other's ID.
2. Service Discovery: When a customer is searching for a specific service or product, it usually looks for the best provider among the participants. The marketplace plays a role here on behalf of the requestors.

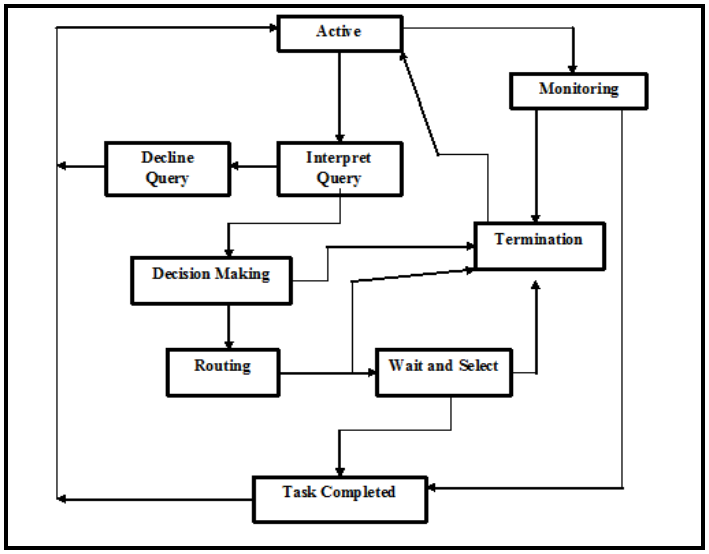

Figure 3: Marketplace state machine diagram

Phase	Description
Active	The marketplace is ready to receive different types of messages (language of encounter)
Routing	The marketplace acts as a router, its task is to forward the received messages to the appropriate receivers, this scenario occurs frequently when the two parties know each other (buyer and seller know each other their Id).
InterpretQuery	Upon receiving a contract query, the marketplace goes into this state to interpret the query based on messages structures.
DeclineQuery	If the marketplace is too busy and it cannot handle new requests, the marketplace goes into this state to send a "*Decline*" message to the customer.
DecisionMaking	After performing interpretation task on a received query and choose to serve it, the marketplace goes into this state to decide on the appropriate receivers, make some modifications on the query (such as reformatting it), accessing the database to find information about the service providers, and so on.
WaitAndSelect	After forwarding a request to service providers, the marketplace wait for responses from the specified providers to select the best that meets the requirements of the customer.
TaskCompleted	After finishing serving a query, the marketplace transit to this state to report that the request was completed successfully by reporting some information about the transaction that might be needed in the future.
Monitoring	While the marketplace in this state, it monitors the process of data transferring for the specified period of time.
TransactionReview	After a transaction is completed between a seller and a buyer and if the buyer complains about the item description, the marketplace transits to this state to resolve the issue.
Termination	This means that the transaction was terminated for some abnormal reasons.

Table 2: Marketplace states and their description

4. Implementation and Evaluation of the System

We used FD-DEVS formalisms to implement the marketplace model and we used SES formalisms to build the messages structure ontology.

4.1 FD-DEVS and the Marketplace Architecture

We have implemented the above negotiation protocols in FD-DEVS. Finite & Deterministic Discrete Event System Specification) is a formalism for modeling and analyzing discrete event systems in both simulation and verification ways. FD-DEVS is based on DEVS formalism [1] [3][4] . However, to implement it in FD-DEVS, we needed to do extra work by splitting some phases into multiple copies based on the message that causes the transition. To implement this in FD-DEVS, we needed to differentiate between the two states to convey two different messages. Our negotiation protocol consists of two scenarios:

1- One-to-One negotiation when entities know each other ID. In this protocol, the customer agent sends messages (such as request, contract query, counter offer, accept, reject) to the marketplace including the service provider ID. The marketplace reveals the service provider ID from the received message and forwards it to the specific service provider (receiver). On the other side, the service provider responds with replies (such as offer, accept, reject) with the customer ID included in the contents of the messages. The marketplace receives the messages, unmarshal them to find the customer ID, and then it forwards them to the specific customer. If the customer did not receive the correct item, it can choose to complain by sending "*Item*" message to the marketplace along with the transaction number. Then the marketplace searches its log files to find the transaction information in order to resolve the issue with the service provider. Figure 4 shows the protocol flow.

2- Service Discovery: When a customer is searching for a specific service or product, it usually looks for the best provider among the participants. Hence, customers query the marketplace to find the best providers, and since service providers advertise their services, information and products to the marketplace, the marketplace will have an updated database of the members of the service providers and their capabilities. The marketplace responds to the customers with a group of service providers who can fulfill their requests. Then the customers will decide on how to proceed with the negotiation process. The customers might choose to proceed with the negotiation by sending a contract query to the marketplace; then the marketplace will forward that to the selected providers that were chosen in the previous step, then it will wait in phase "Wait" to receive responses from the providers one by one. Once it finishes waiting in that phase, it will select the best offer from the list of responds. The best provider will be sent back to the customer. The customer now can choose whether to accept the offer, reject the offer or go to the one-to-one protocol and negotiate with the chosen provider. Once an agreement is reached. The customer establishes a link with the appropriate service provider to transfer data, information or products and then it informs the marketplace of the link establishment. The marketplace now enters a "*Monitoring*" phase to make sure that the agreement is fulfilled. Figure 5 shows the scenario

Notice from figure 4 that when agent **A** receives an "*Offer*" from agent **B**, then he can send back to agent **B** either "*Accept*", "*Reject*" or "*CounterOffer*" messages. When agent **B** receives "*CounterOffer*", then he can send back to agent **A** either "*Accept*", "*Reject*" or a modified "*Offer*" on the same product/service. In any case, one of the agents has to send "*Accept*" or "*Reject*" sometime to end the negotiation process, otherwise they might go into an infinite loop. This is easy to resolve when designing customers and providers agents. One way to solve this problem is by having a timing counter, once it expires, the agent sends a "*Terminate*" message rather than wasting the time with an endless negotiation.

When the marketplace receives complaints regarding a transaction ("*Item*"), it interacts with the item provider to resolve the issue (the two ways dotted arrow in figure 5). Such an interaction depends on the regulations of companies. In e-commerce, usually the product provider (seller) refunds the buyer the item price and the buyer returns the item. Some companies provide products exchange option. In figure 5, when the marketplace processes the "*ContractQuery*" message, it needs to decide on the appropriate receivers of the query (service providers who have the requested service available). Then it forwards the contract query to the chosen providers and waits for responses from them. After receiving the responses it selects the best that meets the requirements in the contract query and sends its ID back to the customer. At that time, the customer will choose whether to establish a link with that provider or maybe negotiate with the selected provider for a better deal using the One-to-One protocol (since now the customer agent knows the service provider ID).

We used the Finite Deterministic GUI tool version 0.6.0 to define the marketplace model. In using the tool, we need to specify the states table, internal transition function and the external transition function. The FD-DEVS tool generates an XML representation of the model. The in ports and out ports names of the model are generated based on the name of the messages; for example, the message "Accept" will be received on in port "inAccept" and will be sent on out port "outAccept".

4.2 SES and the Messages Structure Ontology

The messages structure should be dynamic based on the domain of interest. This is because the information that needs to be sent through messages is different. For example in E-commerce domain, the agents consider parameters such price, shipping and handling, return policy for their products and services. However, agents in Oceanography or software services will have different parameters that they care about such as execution time, bandwidth, latency. In some cases, even under the same domain, the designer can construct the structure of a message to have more than one meaning. Mathieu and Verrons [45] in their attempt to provide a flexible negotiation protocol, they had to add more stages on the One-to-One negotiation protocol to provide "modification request" and "propose modification". In order for our negotiation primitives to accommodate for varying capabilities under different domains, the messages structures must be dynamic and based on the domain under consideration. Hence, in our design we use an ontology structure for each type of messages. The design of the ontology is shown in figure 6 for message *ContractQuery* as an example. The message type (entity) has specialization relations to each of the domains defined

(*SubdomainOfIntrestSpec*). Each domain entity has a decomposition relation with its "domainMsgStructure" which refers to the message structure. Each domainMsgStructure has variable slots (fields) that contain the different parameters of the message structure such as (Price, SellerID, Location, Roughness).

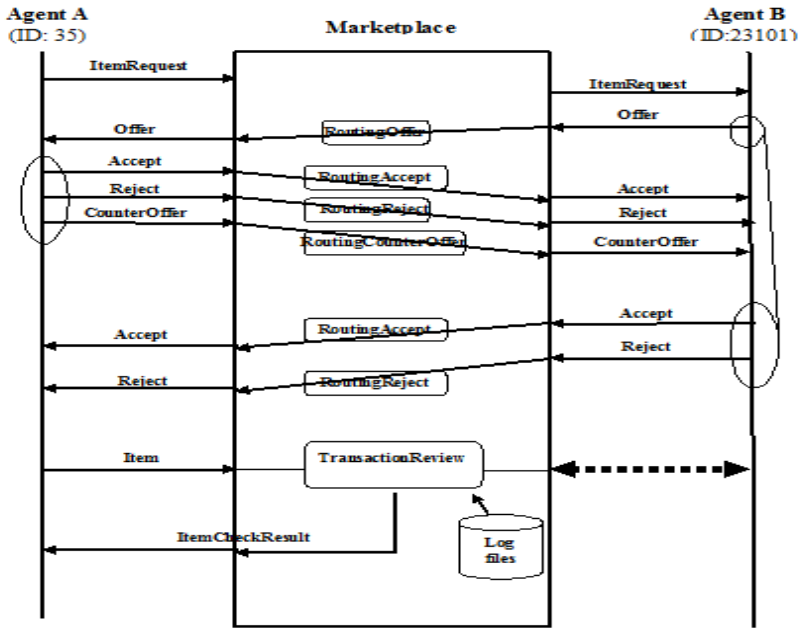

Figure 4: One-to-One negotiation protocol

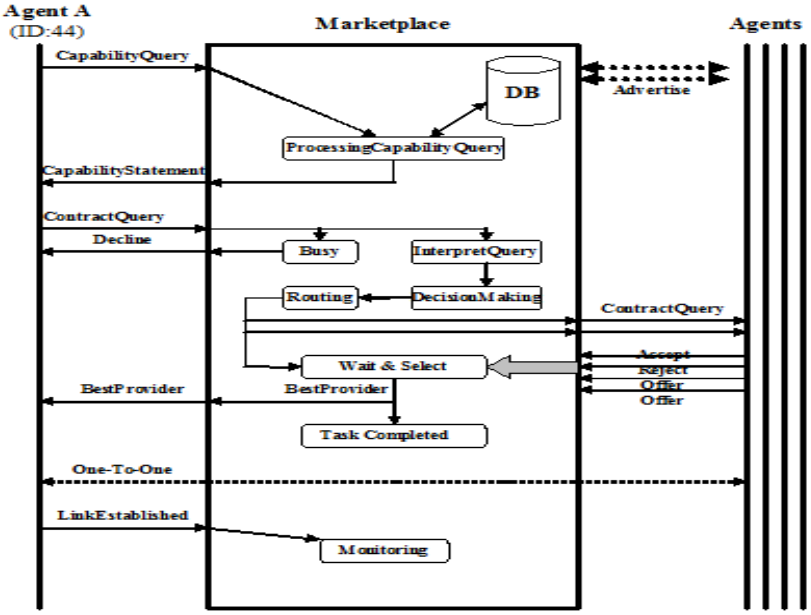

Figure 5: Service discovery negotiation protocol

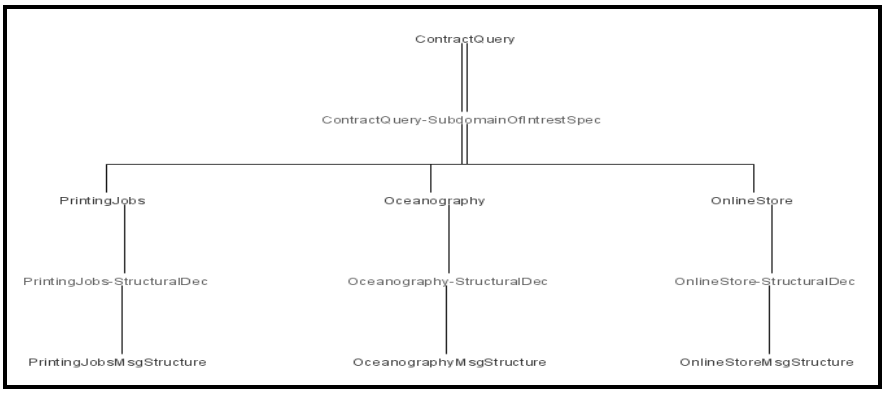

Figure 6: ContractQuery ontology tree

The message structure under PrintJobs domain consists of: PrintJob, TechnologyType, NoCopies, Deadline, Customer, PaperQuality, Duplex, PrintJobID, and Color. The message structure for Oceanography consists of: Speed, Roughness, Location, and Altitude. And for the OnlineStore we chose the structure to have: SandH, BuyerID, Price, Return, and SellerID. In order to implement the dynamic message structure ontology, we used System Entity Structure formalism. SES is a useful ontological framework to define data engineering ontologies. In SES, Entities represent things that exist in the real world or in the imagined world. Aspects represent ways of decomposing things into more fine-grained ones [5]. In our ontology tree, the message type is an entity as well as the domain. SES has been applied to many different areas as a classification tool such as in [13].

We used SES builder to design the message ontologies. The SES builder is an easy to use tool and provides many features. The input is a restricted natural language designed for the system entity structure framework purposes [5][30]. The natural language input that resulted in the above ontology for *ContratcQuery* message is shown in figure 7. For each of the messages types in the language of encounter, we have a text file similar to that in figure 7 that represents the message structure. It is obvious that the natural language interface gives very satisfactory options to the humans to express their ontological specifications. The book by B.Zeigler and P. Hammonds on simulation-based data engineering gives more insights into the natural languages and its usages [5][32].

```
ContractQuery can be PrintingJobs, Oceanography, or OnlineStore in SubdomainOfIntrest!
From the Structural perspective, the PrintingJobs is made of PrintingJobsMsgStructure!
From the Structural perspective, the Oceanography is made of OceanographyMsgStructure!
From the Structural perspective, the OnlineStore is made of OnlineStoreMsgStructure!
The PrintingJobsMsgStructure has PrintJob, TechnologyType, NoCopies, Deadline, Customer, PaperQuality, Duplex, PrintJobID,
and Color!
The OceanographyMsgStructure has Speed, Roughness, Location, and Altitude!
The OnlineStoreMsgStructure has SandH, BuyerID, Price, Return, and SellerID!
```

Figure 7: Natural language input for ContractQuery message

The marketplace architecture is a domain-independent design, where the language of encounter ontology is a domain-dependent structure. Combining both methodology results in a powerful negotiation model that provides enough expressiveness power while enforcing negotiation protocols to capture different user agents behaviors. Figure 8 depicts the big picture of the two methodologies.

4.3 Negotiation System Design Process Flow

The system designer starts the process by defining the domain-dependent message structure using a GUI tool that we implemented as shown in figure 8. The output of the GUI is a natural language for SES ontology structure where SES can be used to create the ontology representation in XML schema. The schema will be the input to the JAXB compiler which in turn results in Java classes defined for the domain of interest. Those Java packages are ready to use but carry no information or data yet. The second pipeline in bottom starts by implementing our negotiation protocols (rules and requirements) in FD-DEVS specifications, which results in a generic domain-independent marketplace model. The tailored marketplace is a result of the designer choice of the domain of interest. The marketplace receives messages, interpret them by unwrapping them (unmarshal) and it might need to marshal them with data and send them. On the service provider side, the same scenario occurs.

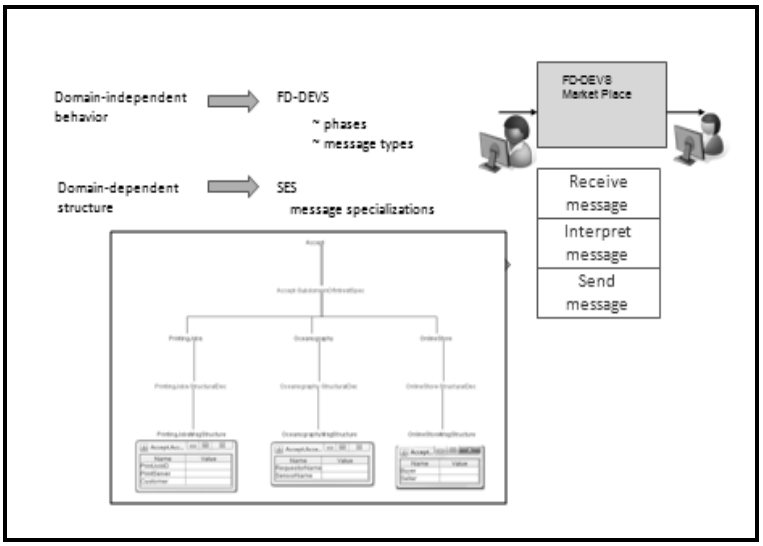

Figure 8: System negotiation modeling approach

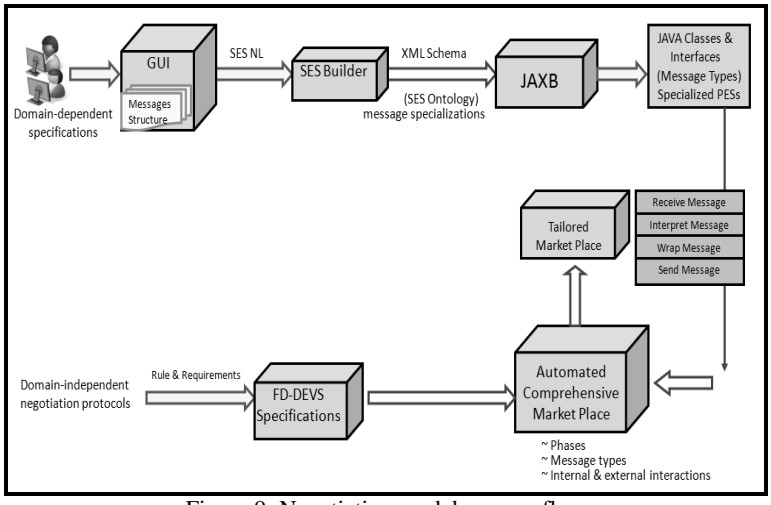

Figure 9: Negotiation model process flow

The process of unmarshalling and marshalling the language of encounter messages represented in Java classes between the requestors and the service providers is shown in Figure 10. On the service provider side, the data collections or services are represented in a pruned entity structures and XML instances. The pruned entity structures (PESs) are product descriptions such as (PrintJob = “Newspapers”, TechnologyType=”Digital”, NoCopies=”1”, Deadline=”20”, Customer”RequestorName”, PaperQuality=”High”, Duplex”yes”, PrintJobID=”15382”, and Color=”BlackandWhite”). These variables are encoded in XML instances in the same formats of the XML schema for ContractQuery message. When the service provider uses JAXB data binding “unmarshaller” of the PESs on an empty *ContractQuery* message class, the returned message will be a *ContractQuery* with the above data inserted in the corresponding slots of the *domainMsgStructure* entity; after that, the message can be exchanged between agents.

4.4 Experiments and Results

The application of the negotiation activity can be applied into many multi-agent disciplines where a user or an agent initiates the process by asking a query or a request to be fulfilled. The user seeks to find either the best provider for his request or a provider that can meet his requirements. In this section we will evaluate our system for two scenarios of interactions where the negotiation model is an essential to the success of requirements fulfillment. The first experiment is concerning surveillance systems in which observers negotiate with active or passive sensors to find the right sensor that can provide the right data and measurements over a specific region. The marketplace intermediates the interactions to find out the best data provider on behalf of the observer. The second experiment occurs very frequently in distributed engineering applications. A user or an engineer tries to find computing resources, where he can deploy his jobs and gets responses from the service provider within a specific deadline. The marketplace helps all negotiating party to reach an agreement. These examples show that the service provider can change dynamically.

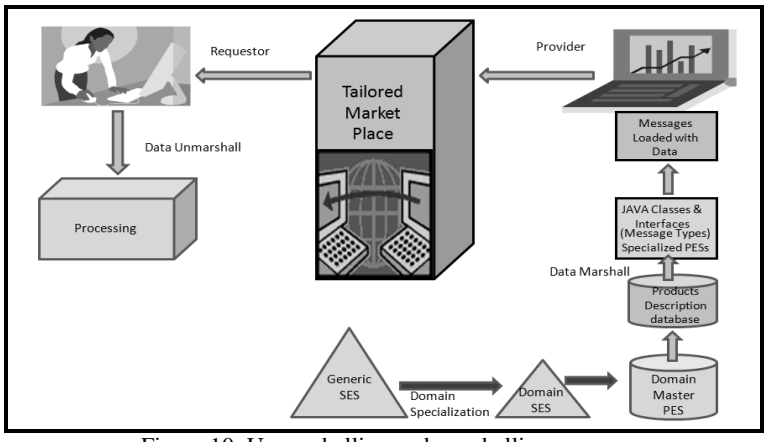

Figure 10: Unmarshalling and marshalling process

4.4.1 Oceanography in Surveillance domain

The problem of finding the best data source has been widely studied in the research. In this example, we will focus on how a requestor of data can find the right data provider for his specifications and how the data providers can be selected dynamically over time. The marketplace permits requestors to communicate with the appropriate data providers based on its database records. Also, the marketplace can decide on behalf of the requestor on who is the best provider. Such a situation occurs if the designer of the domain implements some decision making to compare different offers from the service providers to pick the best out of them. On the other hand, in most of the situations, the decision making is made by the user. However, in this example, the marketplace receives *Reject* and/or *Accept* messages, and then it chooses the best of them. In the next example on distributed services environments, we will show how the marketplace receives *Offer*s messages and routes them to the correct destination (requestor). No decision making will be made by the marketplace except in finding the appropriate service providers.

We applied our system to the Oceanography field in surveillance systems in which experts observe different kinds of nature phenomena that might occur in the ocean. Monitoring the sea level is critical in order to be prepared for any of destruction phenomenon that could affect our cities and maybe causing a terrible impact on our life such as in Tsunami effects. Many authorities and governments have radars and sensors collecting data above the oceans all day time trying to detect any Oil slicks, Tsunami, earthquakes, volcanoes activities, etc. Sensors are divided into two types: namely *active sensors* and *passive sensors* [36] [38]. Active sensors have the ability to measure the distances to objects [37]. Active sensors are capable of measuring sea level and can be used to detect the changes that Tsunami can cause on the ocean level [39].

4.4.1.1 Language of Encounter Structure

We have defined the message structure in the language of encounter ontology as shown in figure 11. We compiled the schemas of each of the message types into a Java package and we named it OceanographyMessages. The figure shows that some of the messages carry no information other than its type, which is all what it is needed for the marketplace to transit from one phase to another. Some messages carry information as needed by the experiment.

4.4.1.2 Observer Model

The Observer model starts the negotiation process in *ServiceDiscovery* phase causing the transmission of a *CapabilityQuery* message to the Marketplace asking if any of the sensors can provide a sea level altitude greater than a pre-defined threshold. The marketplace replies by sending the names of the sensors who can provide such data (need to be an active sensor type). After receiving the *CapabilityStatement* with the names of the sensor from the marketplace, the Observer model transits into *IssueContract* and marshals his specifications in a *ContractQuery* message and sends it to the marketplace. The *ContractQuery* will contain the different types of data that the Observer is interested in (namely Speed, Roughness, Location and Altitude). The marketplace then informs the Observer of the best provider sensor by sending a *BestProvider* message to it. After knowing the best provider, the Observer issues a *LinkEstablished* message asking the marketplace to setup a communication channel with the chosen data sensor. Then the sensor starts sending data periodically to the Observer until the collected data does not meet the specifications (this occurs when the altitude is less than the threshold). Once the dedicated sensor announces that he does not have the appropriate data. The Sensor will ask the marketplace to terminate the channel to the Observer, after that the Observer starts a new cycle looking for the next best provider. In this example, we assumed that there are three regions on the ocean, region A, region B and region C. in region A, the sea level is above the threshold, at that time, Sensor 1 is the best provider of the data and he can provide it for a while because he is covering a large region (A). In region B, Sensor 2 is the best provider. Since the waves move forward leaving the angle view of Sensor 2 at region C, then Sensor 3 becomes the next best provider. Figure 12 shows the state transition diagram.

4.4.1.3 Marketplace Model

The Marketplace receives a *CapabilityQuery* from the Observer to find out the sensors who are capable of measuring the sea level altitude. The Marketplace replies with the active sensors names since all of them can provide altitude measurements. Then it forwards the *ContractQuery* message to the same chosen sensors in the *CapabilityStatement* which is the output of *ProcessingCapability* phase. After that it waits to receive from the Sensors either: *Accept*, *Reject* or *Offer* messages. In this

experiment we have three active sensors, Active Sensor 1, Active Sensor 2 and Active Sensor 3. If it receives two *Reject*s and one *Accept*, then it will choose the one who responded with *Accept* as the best provider and sends its name in a BestProvider message to the Observer. If it receives all Reject, it will send an empty *BestProvider*. If it receives more than one *Accept*, then it will send the last one who replied with *Accep*t as the best provider. Once the Observer receives a best provider the Marketplace will establish a link between them. When one of the two communicated parties sends a *Terminate*, the Marketplace handles that by removing the communication link between them. Figure 13 shows the main state transitions for the Marketplace model for this experiment.

Message Type	Contents
Accept	SensorName, and RequestorName
BestProvider	SensorName
Busy	-
CapabilityQuery	AltitudeThreshold
CapabilityStatement	Sensors
ContractQuery	Speed, Roughness, Location, and Altitude
CounterOffer	-
Decline	-
Item	-
ItemCheckResult	-
ItemRequest	-
LinkEstablished	SensorName, and RequestorName
NotMet	-
Offer	-
ProvidersChosen	SensorsNames
Reject	-
Terminate	SensorName, and ObserverName

Figure 11: Language of encounter structure for Oceanography domain

4.4.1.4 Sensor Model

The Sensor model has database in the form of pruned SES files of the *ContractQuery*, which has fours variable (Speed, Roughness, Location, and Altitude). These data are a proposed data and not real, because of the lack of having real Radar sensors. However, the model gives useful insights on how collected datasets can be used; and no matter how the data is stored in the real sensors, it easily can be mapped into pruned SES files. When the sensors receive the *ContractQuery* message, they unmarshal their corresponding pruned XML files and check whether the variable "altitude >= Threshold". If the statement is true, the sensor will send *Accept*, otherwise it will send *Reject*.

If one of the sensors who responded with *Accept* is chosen as the best provider, the communication link will be established to it. After which it keeps retrieving the data from its own pruned XML files and sends the data to the Observer model. The process proceeds as long as the data he is collecting is greater than the Threshold. Once the Altitude is less than the Threshold, the sensor will send *Terminate*.

4.4.1.5 The System Simulation

The system consists of: Observer model, Marketplace model, Passive Sensor 1, Passive Sensor 2, Active Sensor 1, Active Sensor 2 and Active Sensor 3. The simulation of the negotiation process results in the same behavior as we expected. From simulation time 22 until 70, Active Sensor 1 is the best provider and is chosen to be the data source for the Observer. From simulation time 94 until 116, Active Sensor 2 is the best provider and is chosen as the data source for the Observer. And finally, from simulation time 142 until 172, Active Sensor 3 is the best provider and the appropriate data source.

4.4.2 Distributed Services Environment

Exploiting service providers in a distributed services environment has been a tedious task to achieve. That is because of the fact that service providers are geographically distributed and loosely coupled [16]. Users or engineers have been always try to share computing resources because many of distributed systems are costly and expensive to design and maintain. Hence, whenever it is possible, different companies and other parties prefer to have software services that are optimally utilized where they can deploy their models and jobs on the grid on demand. In these environments, the users concern about different parameters such as the execution time, deadline until they get responses, the quality of the data they need, the solution efficiency.

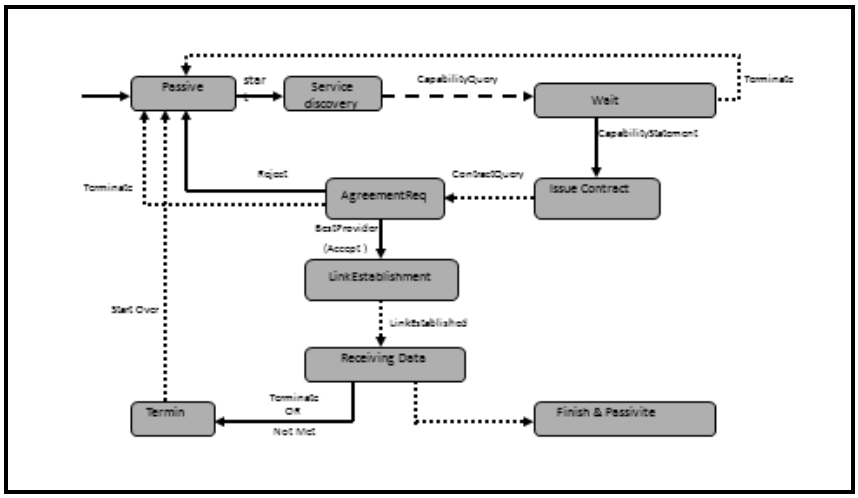

Figure 12: Observer state transition diagram

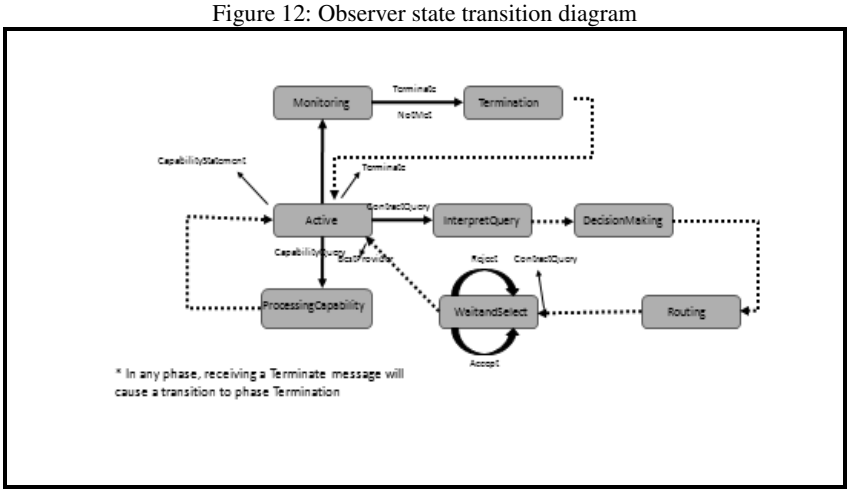

Figure 13: Marketplace main state transitions

Having the services distributed brings the following challenges into systems management techniques. First, users will need help from a third party to locate and find out the appropriate providers among many of them. Second, privacy and transparency where users do not like to publish their interests to every provider registered in a multi agent environment. Third, users do not want to waste time and money to discover their candidates. For example, in [15] an investment banking system based on web services have been discussed where semantic ontologies were developed to represent services in an attempt to close the gap and match between requesters and providers. The point here is that you have many distributed and loosely coupled investment systems and the users cannot locate the provider who can meet their requirements. As a result, a service model based on the semantics is used to make the users understand and choose their best match. In this example, we will show printing jobs scenarios in which users sends different kinds of printing jobs and negotiate on different aspects of the job specifications until they reach an acceptable agreement within their range. The problem is very close into its definition to the problem of deploying computing jobs (or programs) into distributed computing grid. This scenario captures most of the issues that could be found in such engineering service environments.

4.4.2.1 Language of Encounter Structure

We used the GUI that we developed to define the structure of each of the messages in the language of encounter. The result of the tool is a Java package that we gave it the name PrintingJobsMessages. In designing the message structures for this domain, we chose some selections of the types and technologies in current printing servers. For instance, if a customer is concerning with printing business cards, he might choose thermography, Engraving or Letterpress technology. Also, we defined different aspects for paper quality, deadline, color and duplex. Figure 14 shows each message type and the contents/information that it carries. We assumed also that if a new printing server would like to join the printing services community, he should send a "*MyCapability*" message to the Marketplace to register himself. *MyCapability* message should contain at least the provider ID and name along with what printing capabilities he can provide such as: Business Cards, Wedding Cards.

4.4.2.2 User/Customer Model

The user of the printing services system starts the negotiation process by sending a service discovery request to the marketplace asking whether his job can be serviced by any of the printing servers. The marketplace replies with whoever can provide the service for that specific job, for instance, print server 3 provides Business Cards printing. After discovering the

appropriate service providers, the user starts to negotiate with the selected providers by exchanging offers and counter offers on different printing attributes such as paper quality, color, the deadline to finish printing. Once an agreement is reached, the user sends *Accept*. In modeling such an interaction behavior, A DEVS Java model is developed with the following decision making rules.

- The User model is searching for a provider who has the Business Cards printing capability.
- The user would accept an offer if one of the following conditions is satisfied:
    1. If the paper quality is medium or high, the color is full HD and the deadline is less than 80.
    2. If the paper quality is medium or high, the color is RGB and deadline is less than 30.
    3. If the paper quality is medium or high, the color is grayscale and the deadline is less than 20.
- If the offer does not match any of its acceptable ranges, the user sends back a counter offer asking either his first preference or a modified one based on the history of the offers he was receiving. In our model, we chose that the user sends his first preference.

Figure 15 shows the state transitions. At the beginning a start message is injected into the User model causing it to transit into *ServiceDiscover* phase. In this phase, the User puts its printing job type and its name into a *CapabilityQuery* message and sends it to the Marketplace model at the end of the phase (internal transition). Then the User waits for a *CapabilityStatement* in phase *Wait*. After receiving the *CapabilityStatement*, it gets the selected providers for his job and transits to state *IssueContract*, where a *ContractQuery* message is prepared with different printing job specifications and attributes to be sent to the selected providers. Note here that if *CapabilityStatement* that the user has just received from the Marketplace does not contain any providers, then even if the User sends a *ContractQuery* message to the Marketplace it will not be routed to any of the providers since none of them supports the User requirements. The internal transition from *IssueContract* outputs *ContractQuery* to the Marketplace and the User goes into state *Agreement* waiting for an agreement with any of the appropriate providers. While the User in the *Agreement* phase, he will be receiving different *Offers* from the selected providers. It will wait in the *Agreement* state for a specific time (we selected it to be enough until all the providers complete sending their offers). The internal transition function causes the User to transit into *DecisionMaking* phase, in which it starts pulling each *Offer* he received and decide whether it meets his acceptable range or not. In this state, the User unmarshals the data he needs from the *Offer* message to decide on that offer, this include the different fields in the message such as: PaperQuality, Color, Deadline, TechnologyType. If the *Offer* does not meet his interests, the User goes into *IssueCounterOffer* state where the internal function cause a *CounterOffer* message to be sent at the end of that phase to the source of that specific *Offer*. After sending all *CounterOffers* to the providers involved in the negotiation process, the User waits in state *Wait*. The internal transition function takes the User from *Wait* into *Agreement* again and the same cycles of *Offers –CounterOffers* proceeds until an acceptable *Offer* is detected. If the User receives an *Offer* that is acceptable to him, then during the *DecisionMaking* state the User will decide to transit to phase *Acceptance*. The internal transition from *Acceptance* causes a message *Accept* to be sent to the Marketplace and then to the provider who owns that *Offer*. Immediately after that, a transition to phase *LinkEstablishment* occurs. The internal transition from *LinkEstablishment* causes an output of message *LinkEstablished* to be sent to the Marketplace and the appropriate provider in order to inform them that the user is ready to receive the service. The User transits into *Receiving Data* until the provider processes his job and send him back an acknowledgment (*DataOut*) that he finished processing his job. Once the User is informed that his job is finished, he goes into *Termination* state causing message *Terminate* to be transmitted to the Marketplace.

4.4.2.3 Marketplace Model

The generic Marketplace model is used here. However, we added two more functionalities to permit the Marketplace to intermediate the negotiation to enhance the performance and efficiency. The two functionalities are:

1. Dynamic coupling and decoupling to setup channels between the User model and the service providers based on the message source and destination.
2. When receiving a *ContractQuery* message from the User to be forwarded to the appropriate providers. The Marketplace unmarshals it and adds a unique *PrintingJobID* field. The purpose of this field is to enable the Marketplace to keep track of all the jobs that goes between users and providers, and to differentiate between all of the jobs, it will be helpful to have the Marketplace adding a unique ID for each job in order for future purposes such as resolving an a agreement. For example, when a User complains about an agreement violation, the Marketplace can access its own database and find out the job that needs to be resolved.

Message Type	Contents
Accept	Customer, PrintServer, and PrintJobID
BestProvider	-
Busy	-
CapabilityQuery	PrintJob, and Customer
CapabilityStatement	PrintJob, and PrintServer
ContractQuery	PrintJob, TechnologyType, NoCopies, Deadline, Customer, PaperQuality, Duplex, PrintJobID, and Color
CounterOffer	PrintJob, TechnologyType, NoCopies, Deadline, Customer, PaperQuality, PrintServer, Duplex, PrintJobID, and Color
Decline	-
Item	-
ItemCheckResult	-
ItemRequest	-
LinkEstablished	Customer, and PrintServer
NotMet	-
Offer	PrintJob, TechnologyType, NoCopies, Deadline, Customer, PaperQuality, PrintServer, Duplex, PrintJobID, and Color
ProvidersChosen	-
Reject	Customer, PrintServer, and PrintJobID
Terminate	-

Figure 14: Language of encounter structure for PrintingJobs domain

The rest of the Marketplace behaviors follow the same rules and specifications as mentioned previously when we discussed the Marketplace architecture and its functionalities. We will point out here that when the Marketplace receives a *ContractQuery* from users, it will forward it to the appropriate providers based on their capabilities that were published in the past. After which the Marketplace waits for responses from the providers. When it receives offers from the providers, it routes them back to the destination of the Offer messages. The Marketplace database consists of XML files in the project path under directory "MarketplacePrunedDB". These XML files contain the printing job type name and the names of the providers who can provide that printing type.

4.4.2.4 Service Provider Model

The Print Server model or Service Provider accesses its own XML files database in the same way the Marketplace accesses its database. Each of the Print servers has different printing capabilities that are stored in the XML files. For example, Print Server 1 has the capability to print Business Cards, Brochures, Newspapers and Posters. The specifications of each of these printing capabilities will be stored under the corresponding PES file for that printing capability; for example, "Business Cards.xml". We assume that each of the print servers can update or change on these specifications such as Deadline in order to match user requirements. The modifications process of the aspects follows some rules which were defined for each of the Print Servers models. The scope of this research is not on how the decision making occurs on the Print Server side or the user side. It could be a manual user interaction, or an automated mathematical model that captures the user objective function. Hence, in our implementation we have assumed some random updates on different printing jobs specifications, for instance, we used that CurrentDeadline = PreviousDeadline – Update.

If a new Print Server would like to join the printing services community, he sends a "MyCapability" message including his name and the printing capabilities he provides. Then the Marketplace will add him to its database along with the printing capabilities. When the Print Server model receives a *ContractQuery* message, he transits into *DecisionMaking* state, where a decision will be made on whether he can meet the requirements of the printing job in the *ContractQuery* message or not. If he can, then he will send *Accept* and an agreement will be reached. However, if he cannot meet the customer specifications, he will send an *Offer* message to the Marketplace including his current offer and his name. The Marketplace receives the message, find out the customer name by unmarshalling the message, and then routes it to the appropriate receiver. The way we designed the decision making rules in this experiment is to show how negotiation cycles of *Offer-CounterOffer* occur. On the other hand, in the previous experiment as we explained, the decision making was direct with best provider chosen.

The internal transition function causes the transition from *DecisionMaking* to *Offering* phase, the output of *Offering* phase is an *Offer* message. After that, the Print Server model holds in *WaitonOffer* phase; in which the Print Server waits to receive *Accept*, *Reject* or *CounterOffer*. If he receives a *CounterOffer*, he goes into the same cycle of *DecisionMaking->Offering->WaitonOffer*, or he can accept and goes into *Acceptance* state which results into sending *Accept* message. In this implementation, we assumed that if a Print Server sends an *Offer* to a Customer and the customer accepts the offer, then an agreement is reached. No need to go back to the Print Server and asking him whether he accepts or no.

If the Print Server receives *Accept*, he will hold in state *ProvideService* for the time defined in the *Offer* Deadline. Internal transition causes the model to transits from *ProvideService* to *Passive* and an output of *DataOut* will be sent to the Customer informing him that the processing of his job has finished. The state transition diagram of the Print Server is shown in Figure 16.

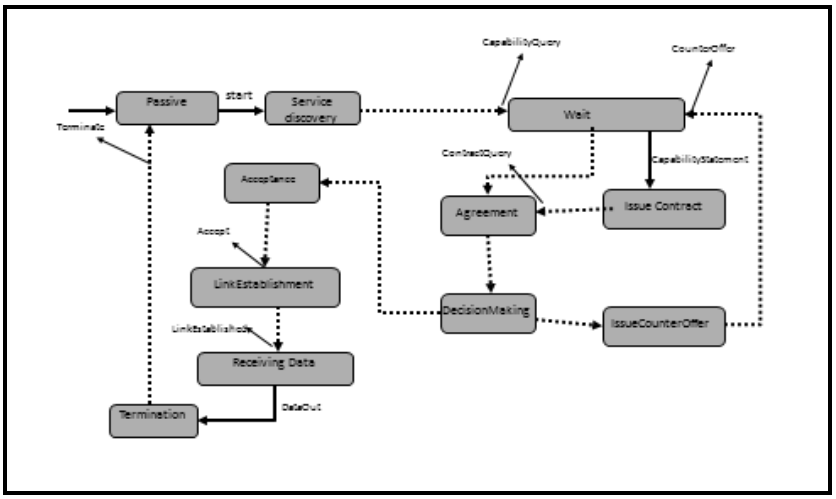

Figure 15: State diagram for User/Customer model

4.4.2.5 The System Simulation

We have seven Print Servers each of which has its own printing capabilities which are defined in his own PESs database. When the Marketplace needs to send a message to Print Server X, it adds coupling to it, sends him that message and removes the coupling unless its needed in the next step of the simulation. The Print Servers can easily send their messages to the Marketplace because it is connected to the Marketplace model. Print Servers models and the User models exchange their *Offers-CounterOffers* through the Marketplace model. If a Print Server model and the User model needs to communicate, they inform the Marketplace and then the Marketplace add the required coupling permitting them to negotiate. This situation occurs when they reach an agreement, the customer will ask for a link to be established resulting in the Marketplace adding a link between the two parties of the agreement. The link will be removed once the job processing is done. The output of the negotiation activity is an agreement as shown below. When we started the simulation, we did not know the result ahead of time. After the simulation is done, we compared the negotiation result with the customer decision making options mentioned earlier and The terms of the agreement match the first condition of the User model decision making:

1. If the paper quality is medium or high, the color is full HD and the deadline is less than 80.

Agreement Offer information is:

Customer : Customer Job Type : Business Cards Print Server : Print Server 6 Color : FullHDColor
Paper Quality : High Deadline : 78 Duplex : Yes Number of Copies : 1 Technology Type : Thermography

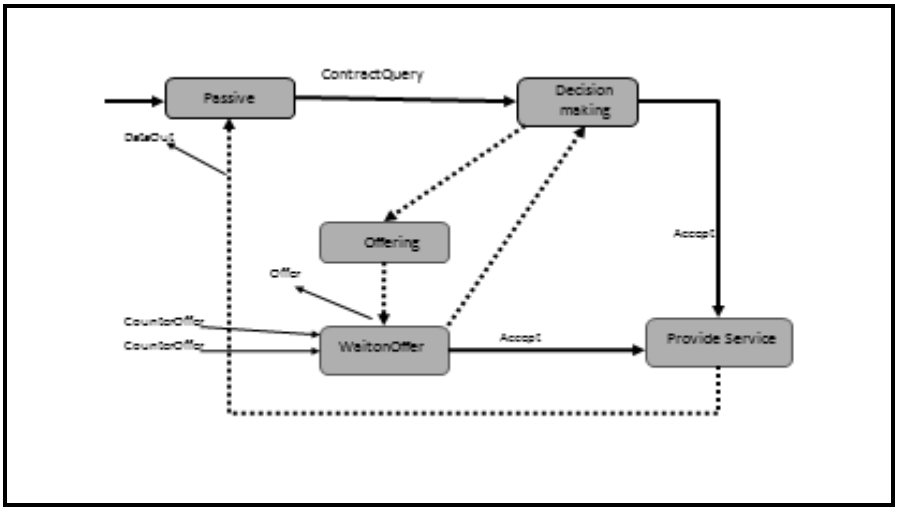

Figure 16: Print server state diagram

5. PROOF OF CONCEPT (DEVS/SOA)

5.1 DEVS/SOA Environment

DEVS Service Oriented Architecture is a web services multi-server environment to support DEVS simulator. The system consists of two services, namely *MainService* and *Simulation* Service. Our concern in this section is the *MainService* and how can we deploy our models in the system. The *MainService* has four functionalities, *Upload* DEVS models, *Compile* DEVS models, *Simulate* DEVS models and *Get* results of the simulation[17][34][35]. DEVS Service Oriented Architecture was designed to support interoperability between different platforms and for heterogynous servers. In order to support that, the system nodes exchange messages among each other as strings in XML formats. For us to use such capability, we created a new class type of each of the language of encounter that has a String local variable where we send the pruned XML structure of a specific message as a string.

The DEVS/SOA system we used is a centralized distributed simulation, which means, a coordinator controls the time for the next event t_N. The coordinator asks each node in the distributed environment for their local next time event t_N and collects them all. Then the coordinator calculates the minimum t_N, and informs each of the servers to change their next time event to the minimum t_N that was just computed. The following section shows the steps in deploying our models in DEVS/SOA and the output results of the distributed simulation.

5.2 Printing Jobs Deployment in DEVS/SOA Environment

After preparing the Print Jobs experiment to run on DEVS/SOA environment, we chose five different machine servers to deploy the models. The first step of the models deployment is the IP assignments of each of the models. The second step is to upload the models to the servers, where a copy of each of the models (client) will be sent to the appropriate machine that has the IP address assigned to. The third step is to compile the models and then the last step is to run the simulation. We assigned Print Server 1, Print Server 3 and Print Server 6 to different machines dedicated to run their models because we knew from the beginning that those three print servers are the only ones capable to provide the customer request. Hence, in order to show that the negotiation occurs between separate machines, we chose this assignment.

After the simulation is over, we got the results as we expected. The output of the Customer model on machine 150.135.218.200 shows that the Customer sent a *ContractQuery* message to the SOAMarketplace asking for Business Cards Printing Job. Then he starts getting *Offers* which transits him to *DecisionMaking->IssueCounterOffer->DecisionMaking-> IssueCounterOffer* …and so on, until he receives an *Offer* that is acceptable to his decision making rules. After that, he establishes a link with the provider. Also, we implement the Customer to print out any Offer he accepts. Notice here, that although the rest of the other Print Servers (2, 4, 5, 7) are deployed and running, but none of them produced output that is because they are not part of the negotiation since they do not provide Business Cards printing capability. Print Server 1 and Print Server 3 outputs are almost the same except that each one of them outputs whatever *Offers* they are sending to the *Customer*. The offers information is for Business Cards printing. Notice here also that the Deadline does change from time to time since we designed them to update their Deadline such as:

$$CurrentDeadline = PreviousDeadline - Update$$

Print Server 6 is the winner provider of the negotiation process since he replied to the customer with an *Offer* that is acceptable to the Customer satisfaction. Hence, we can see in the output of the Print Server 6 that it goes into phase *ProvideService*. Print Server 1 and Print Server 3 outputs do not show that they provided any service to the Customer. The output on machine 150.135.218.204 shows that after the marketplace received a *CapabilityQuery* for Buisness Cards job, it accessed it's XML files database and found that Print Server 1, Print Server 3 and Print Server 6 are the only providers for Business Cards. Then it received a *ContractQuery* and transits to *InterpretQuery* to interpret the message. Then the market place went through *RoutingOffer-> RoutingCounterOffer->* and so on, until it received an *Accept* message, it forwarded it to the appropriate provider (Print Server 6) and then it transited into *Monitoring* after receiving *LinkEstablished* message. After Print Server 6 finished processing the Customer printing job, the Customer sends *Terminate* to the Marketplace causing its transition from *Monitoring* phase into *Active* phase.

6. Conclusion

This ends our objective of the DEVS/SOA implementation which is a proof of the concept that our system can be used in distributed engineering applications. Whether the distributed nodes are sensors who collect data and information, computing resources who provides an environment for software and hardware resources, print servers who provides different printing capabilities or online stores who provide products; all these and other domains can use the system to support different interaction behaviors. This can be done by using flexible negotiation protocols that are enforced by the trusted third party marketplace architecture we developed. The language of encounter, which was designed to be dynamic in structure, gives the domains enough expressive tools and capabilities to define their own messaging system so that users of the domain under consideration can simply understand and use them in the correct manner. Negotiation with service providers can take couple of minutes at the beginning to find the best (or an appropriate) provider; but once it is found, it could save hours and even days of data transformations or jobs processing.

7. REFERENCES

[1] Bernard P. Zeigler, Herbert Praehofer and Tag Gon Kim, "Theory of Modeling and Simulation", 2nd Ed, Academic Press, 2000.

[2] Bailin, S. and Truszkowski, W. "Ontology negotiation between scientific archives", *Proceedings of the Thirteenth International Conference on Scientific and Statistical Database Management* (SSDBM 2001), IEEE Press, July 2001.

[3] M.H. Hwang and B.P. Zeigler, ``Reachability Graph of Finite & Deterministic DEVS``, *Proceedings of 2006 DEVS Symposium*, pp48-56, Huntsville, Alabama, 2006 .

[4] Bernard P. Zeigler, "DEVS Today: Recent Advances in Discrete Event-Based Information Technology", *11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp.148-161, 2003.

[5] Bernard P. Zeigler and Phillip E. Hammonds, "Modeling and Simulation-Based Data Engineering. Introducing Pragmatics into Ontologies for Net-Centric Information Exchange", Academic Press, 2007.

[6] J. Kim and A. Segev, **"**A web services-enabled marketplace architecture for negotiation process management". *Decision Support Systems*, Vol. 40, pp.71-87, July 2005.

[7] Y. Feng et al., "Research on Collaborative Negotiation for E-Commerce"". *Proceeding of the 2nd international conference on Machine Learning and Cybernetics*, Nov. 2003.

[8] Greg O'Hare and Nick Jennings, "Foundations of Distributed Artificial Intelligence", *Sixth-Generation Computer Technology Series, John Wiley & Sons, Inc*. 1996.

[9] Inaba and T. Okamoto, "Negotiation Process Model to Support Collaborative Learning". *Systems and Computers in Japan*, Vol.28, No. 14, 1997.

[10] Krishna, V.; Ramesh, V.C., "Intelligent agents for negotiations in market games. I. Model", *IEEE Transactions on Power Systems*, Vol. 13, Issue 3, Aug 1998.

[11] Murugesan, S., "Negotiation by software agents in electronic marketplace", *TENCON Proceedings*, Vol.2, 2000.

[12] Masvoula, M. et al., "Design and development of an anthropocentric negotiation model"**,** *Seventh IEEE International Conference on E-Commerce Technology*, 2005.

[13] Park, H.C. and Kim, T.G., "Relational algebraic system entity structure for models management", *IEE Preceedings on Computers and Digital Techniques*, Vol.143, Iss.1, pp.49-54, 1996.

[14] M. Contreras and J. Hernández, "Ontology Solution for Communicating Heterogeneous Negotiation Agents in a Web-based Environment". *Proceedings of the Fourth Latin American Web Congress (LA-WEB'06) IEEE*, pp.59-66, 2006.

[15] D. Bell, S. A. Ludwig, and M. Lycett, "Enterprise Application Reuse: Semantic Discovery of Business Grid Services", *Journal of Information Technology and Management*, vol. 8, no. 3, pp. 223-239, 2007.

[16] Addis, M. J., Allen, P. J. and Surridge, M., "Negotiating for Software Services". *Eleventh International Workshop on Database and Expert Systems Applications (DEXA2000)*, September 2000.

[17] Taekyu Kim, "Ontology/Data Engineering Based Distributed Simulation over Service Oriented Architecture for Network Behavior Analysis", *Ph. D. Dissertation, Electrical and Computer Engineering Dept., University of Arizona*, Spring 2008.

[18] V. Tamma et al., "Ontologies for supporting negotiation in e-commerce". *Engineering Applications of Artificial Intelligence*, 18:223–236, 2005.

[19] L. Yilmaz and S. Paspuleti, "Toward a Meta-Level Framework for Agent-Supported Interoperation of Defense Simulations". *The Society for Modeling and Simulation International, JDMS*, vol.2, pp.161-175, July 2005.

[20] S. Decker et al., "The Semantic Web – on the respective roles of XML and RDF". *IEEE Internet Computing*, September-October 2000.

[21] Krishna V. and Ramesh VC., "Intelligent Agents for Negotiations and Market Games, Part 1: Model". *IEEE transaction on Power Systems*, Vol.13, pp.1103-1108, 1998.

[22] Archibald J. K. et al., "Satisfying Negotiations". *IEEE Transaction on Systems, Man and Cybernetics, Part C*, Vol. 36, Issue 1, pp.4-18, Jan. 2006.

[23] Osborne M. J., and Rubinstein A., "Bargaining and Markets". Academic Press, San Diego, 1990.

[24] Mahajan R. et al., "Experiences Applying Game Theory to System Design", *proc. SIGCOMM PINS Workshop*, 2004.

[25] Persons S. and Wooldridge M., “Game Theory and Decisions Theory in Multi-Agent Systems”. *Autonomous Agents and Multi-agent Systems* Vol.5, No.3, pp.243-254, 2002.

[26] Binmore K., and Vulkan N., “Applying Game Theory to Automated Negotiation”, *Netnomics*, Vol.1, No.1, pp.1-10, 1999.

[27] Krishna V. and Ramesh VC., “Intelligent Agents for Negotiations and Market Games, Part 2: Application”. *IEEE transaction on Power Systems*, Vol.13, pp.1109-1113, 1998.

[28] eBay. http://www.ebay.com

[29] Morris, J. and P. Maes. "Negotiating Beyond the Bid Price.", *Workshop Proceedings of the Conference on Human Factors in Computing Systems (CHI 2000),* April, 2000.

[30] “Modeling and Simulation-Based Data Engineering” Online Site. http://www.devsworld.org/

[31] Priceline. http://www.priceline.com/

[32] RTSync Tutorials. http://www.sesbuilder.com/ses_tutorial.html

[33] Amazon Auctions. http://www.amazon.com/auctions

[34] Mittal, S., Risco-Martin, J.L., and Zeigler, B.P., "DEVS-Based Simulation Web Services for Net-centric T&E", *Summer Computer Simulation Conference SCSC'07*, July 2007.

[35] Cheon, S., and B.P. Zeigler., “Web Service Oriented Architecture for DEVS Model Retrieval by System Entity Structure and Segment Decomposition.” *Paper presented at the DEVS Integrative M&S Symposium*, Huntsville, AL 2006.

[36] S. Rodriguez et al., “Complementarity of Radar and Infrared Remote Sensing for the Study of Titan Surface”, *Workshop on Radar Investigations of Planetary and Terrestrial Environments, 2005*.

[37] Nancy Gordon, Cliff Ogleby, Remote Sensing Centre for Environmental Applied Hydrology, Department of Civil and Agriculture Engineering, University of Melbourne.

[38] Henderson F. M. and Lewis A. J. *Manual of Remote Sensing*, *vol. 2*, 1999

[39] The European Southern Observatory (EOS) - http://www.eso.org/public/

[40] Hoh In, Olson, D. and Rodgers, T., “A Requirements Negotiation Model Based on Multi-Criteria Analysis Source**”,** *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, 2001.

[41] WordNet A lexical database for the English Language. http://wordnet.princeton.edu/

[42] Bravo et al. , “Ontology support for communicating agents in negotiation processes”, *Proceedings of the Fifth International Conference on Hybrid Intelligent Systems*, Nov. 2005.

[43] Dung et al., “A Proposal of Ontology-based Health Care Information Extraction System: VnHIES**”,** *IEEE International Conference on Innovation and Vision for the Future*, March 2007.

[44] Tomai & M. Spanaki, "From ontology design to ontology implementation: A web tool for building geographic ontologies", *In Proceedings of the 8th 8th AGILE Conference on Geographic Information Science*, Estoril, Portugal, May 2005.

[45] Mathieu, P. and Verrons, M.H., “A generic model for contract negotiation.”, *In AISB'02 Symposium on Intelligent Agents in Virtual Markets,* April 2002.

[46] DISTAL – Distributed Software On-Demand For Large Scale Engineering Applications. EC project EP26386.

[47] Cooper, T.P., “Case studies of four industrial meta-applications”. *High Performance Computing and Networking*, Springer Lecture Notes in Computer Science, 1999.