A Modeling and Simulation-based Methodology to Support Dynamic Negotiation for Web Service Applications

Moath Jarrah \(^1\), and Bernard Zeigler \(^2\)

\(^1\) Department of Computer Engineering
Jordan University of Science and Technology, Jordan
mjarrah@just.edu.jo

\(^2\) Arizona Center for Integrative Modeling and Simulation
Department of Electrical and Computer Engineering
The University of Arizona, USA
zeigler@ece.arizona.edu

ABSTRACT
Different negotiation engineering domains require from the system designer to tailor the negotiation framework according to the domain under which it will be used. This process of system design is timely consuming when supporting different geographically distributed and dynamic environments. Here we show a methodology to design negotiation systems by integrating domain-dependent message structure ontology with domain-independent marketplace architecture. The methodology gives the system designers a powerful modeling tool that can be used to tailor the framework in order to support different negotiation behaviors under different domains. System Entity Structure (SES) formalism is used to build the domain-dependent ontology while Finite Deterministic Discrete EVent System (FD-DEVS) formalism is used to build the marketplace model. The discrete event system with service oriented architecture (DEVS/SOA) simulation environment was employed to demonstrate a proof of concept of applicability to web service domains.

Keywords: SES, FD-DEVS, DEVS/SOA, Ontology, Negotiation Process, Language of Encounter

1. INTRODUCTION
The negotiation process is an essential activity that needs to be used widely and correctly in today’s complex distributed systems. The complexity comes in having many parameters that manage computing resources in geographically distributed systems. Such systems need to provide negotiation capabilities on different parameters in order to reach agreements and behaviors that are efficient and intelligent. For example, a programmer that needs to deploy a task on a busy computing resource might keep on rechecking the resource availability every 1 minute. However, if we let the programmer negotiate with the computing resource; the programmer might find out that the resource will be available after 1 hour. As a result, the programmer might wait and come back to deploy the desired task after 1 hour which is less costly and more efficient for both parties. The key point here is to allow negotiation on different dimensions selected by the system users while both share a common ontology.

The ability to reserve and utilize software and/or hardware services in current complex geographically distributed system has become increasingly difficult. The complexity results from the fact that there are many aspects and factors that represent the characteristics of these systems, such as a node bandwidth, job processing deadline, execution time, etc. The user’s decision of whether to use a computing service or not is based on these factors. Many researchers and collaboration parties have tried to provide solutions to exploit these resources efficiently [1] [2]. However, in spite of the importance of the negotiation process in this research, each of the negotiation environments was developed to be used under only one domain of interest such as ebay.com under e-commerce domain.

In addition, 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
marketplace, information discovery and retrieval has encouraged more and more companies to provide Web Services either to satisfy customer’s requirements or to manage their distributed computing resources. This leads to the diversity in resources and data availability which adds new challenges to the management techniques that systems use.

The contribution of this paper is to provide a modeling and simulation platform for designing negotiation systems under different domains for geographically distributed environments. Our method consists of integrating DEVS formalism, SES ontological framework and the underlying DEVS/SOA to develop web services for the negotiation system in hand. The DEVS simulation environment is used to simulate the generic negotiation protocols enforced by a marketplace model written in FD-DEVS. SES framework is used to develop a domain-dependent ontology that is understandable by different entities inside a specific domain. The ontology describes the language of encounter (messages) in a specific domain. DEVS/SOA tool provides the negotiation system’s designer with the capabilities to implement the system into non-simulation domain (Web Services).

Section 2 in the paper provides some overview on the literature and modeling technologies. In section 3, we provide a description of the modeling approach that we developed and how it can be utilized under different domains. A proof of the concept using web services is established in section 4 where we use the DEVS simulator under web services framework to present a simple negotiation system for distributed printing capabilities. Finally we conclude in section 5.

2. RELATED WORK AND BACKGROUND

The objective in this research is to support negotiation capabilities over more than one dimension while supporting a generic negotiation model integrated with ontology design. The process of integration both is automated with code generation capabilities to simplify the development effort. Many researchers attempted to apply the negotiation under different domains such as in current bidding and auction systems. In this domain, systems consider the price as the only parameter that in which users are interested as in eBay [3] and Amazon Auctions [4]. 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 [5]. Priceline.com [6] is an airline booking auction where a user selects 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 has to get the ticket. This type 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.

In our approach, we can have as many constraints as needed. 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.

The work by Addis, Allen and Surridge (or DISTAL project) [7] used one-to-one negotiation protocol to support on-demand pay-per-use software services; which makes it a special case of our negotiation methodology under a specific domain. The messages that are exchanged in DISTAL project are very similar to our offer and counter-offer messages.

Authors of [8] used a shared ontology to model the protocols that could be encountered in supporting agent negotiation in e-commerce environment. In their model, 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. In our model, we used generic protocols to support almost all negotiation behaviors with flexible messaging system via building different ontologies according to the domain of interest.

In multi-agent systems, the players (negotiators) can adopt a bottom-up approach where each player maximizes the profit. This approach usually results in a non-optimal overall solution (group performance). Alternatively, the
agents in the negotiation group can adopt a top-down approach, where the objective is to optimize the overall utility function of all players. This leads to better results in the coordination community, where agents need to compromise their utility for the group utility. In many situations, agents might refuse to announce their utility function to the public. Yilmaz and Paspuleti [9] in their model for interoperation of defense simulations 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. The Mediator agent resolves four types of conflicts: semantic, descriptive, heterogeneous and structural. In this regard, having a trusted third party that can manage and coordinate the cooperation between the agents is useful (the marketplace model).

Jennings, Parsons, Sierra, and Faratin [10] gave a description in their paper about automated negotiation to support autonomous agents with the infrastructure and capabilities to reach to agreements. Our paper discusses the negotiation infrastructure and does not focus on the decision methods of negotiators. Whether the negotiators are human interacts manually with the system through an interface, or autonomous agents who follow rules to reach to an agreement, our negotiation methodology provide them with the capabilities they need. The negotiation protocols we designed allow users to invoke one-to-one or one-to-many negotiation. In addition, they provide transparency and service providers’ discovery of the unknown ones. The structure of the semantics they use is dynamic and can be modified to support the dynamic structure that the authors in [10] discussed. Also the authors mentioned the dimensionality problem where negotiators search a space for agreements. In our system, we provide this capability by adding attributes and arguments as needed to the language of encounter ontology under the domain of interest.

2.1 Ontology Design

Our approach to market-place characterization also employs an ontological framework to capture its key behaviors in generic terms and allow them to be specialized for particular application domains. Knowledge representation using ontology structures is a relatively new research topic that emerged with the new requirements of the web. Tim Berners Lee [11] predicted in the late 1990s that the web will be changing to support data, information and knowledge exchange. In addition, Lee reasoned that the knowledge contained in web pages will be understandable by the machines. Since then, semantic web has become a hot research area in which many parties are cooperating to develop standards and rules to govern the interaction over the net such as the World Wide Web Consortium (W3C).

The success of semantic web highly depends on the success of ontology design and development. 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 [12]. 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 [13] and geographic applications [14]. Also, different ontology framework tools have been developed to help designers build their ontologies such as in the Protégé platform that was developed in Stanford [15]. Protégé supports two main ways of modeling ontologies via the Protégé-frames editor and the Protégé-OWL editor. OWL was introduced by the W3C consortium as the future ontology description language because it supports more features than other standards (such as XML, XML schema, RDF) [16] [17]. Here we are using SES/FDDEVS with DEVS/SOA since it allows our modeling methodology to be applied directly under DEVS simulation environment.

The System Entity Structure formalism [18] [19] provides a formal ontological framework for specifying real system composition with information about decomposition, specialization and taxonomy. The SES formalism has been applied to many engineering applications and proved its usefulness such as in data engineering [19] and network systems [20]. In real systems, objects are represented by entities in system entity structure framework. The SES represents the design space with various possible design configurations. To search for the best configuration, pruned SESs are constructed to reduce the search space into valid instances of the SES. For example, SES can have many specializations and multi-aspects relations. With pruned SES, a decision will be made on which of the entities and specialization should be chosen. The basic components of SES are:

- Entity: entities are representation of some real world objects, which in turn can be made of many other children entities.
• Aspect: represents the decomposition relation. An entity is composed of other entities. The relationship between the parent and the children is “aspect”.

• Specialization: represents alternative choices that a system entity can take. Each of the alternatives is also of type entity.

• Multi-Aspect: is a relation that expresses an all of one kind.

• Variables: are slots attached to some entities in the system. The slots can take values in a specific range. The slots define different contents of the associated entity.

The process of pruning an SES is to construct a desired structure to meet a particular domain specification. The pruning process chooses one entity out of many in specialization relations, which results in a completely pruned entity structure PES and variables take values in their ranges. At the implementation level, they are represented by XML schema or DTD and XML instances respectively. We are using SES to construct ontology for the domain of interest to support different messaging in multi-agent negotiation applications. When we have more than one domain, specialization components exist.

2.2 Finite Deterministic-DEVS

Since discrete event systems specifications formalism (DEVS) and its concepts were introduced in 1976, they have been regarded as a powerful tool in many engineering applications areas such as manufacturing [21], ecological disasters [22], computer [23], traffic [24] and command and control System of Systems (SoS) [25]. Finite Deterministic-DEVS (FD-DEVS), first introduced as Schedule-Controllable DEVS in 2005 [26], is based on the classical DEVS formalism concepts and relations with a simplification that restricts the states, inputs and outputs to finite sets while preserving the essential discrete event timing properties of the parent formalism. FD-DEVS serves as a specification language in our development of market-place behavior [27]. The difference between FD-DEVS and the standard DEVS is that, it enforces more rules in defining system states to solve the problem of (once it dies, it never returns) or ODNR that was not addressed before in the standard DEVS formalism [26] [28]. Moreover, DEVS still provides the simulation environment to execute models that are written in FD-DEVS tool.

In our approach we use the DEVS formalism to provide a rich environment in which phenomena are modeled by developing a DEVS model which in turn can be simulated under the DEVS simulation environment [29]. DEVS can model continuous systems as well as discrete event systems. Any real system (or proposed one) that goes through different states or phases, receive inputs from users or from other running entities, output messages to the interconnected properties, and has functions or algorithms that decide the transition from one state to another either concurrently with receiving inputs or after some phase period of time. In particular, a market-place can be modeled as a discrete event system with some specific parameters that need to be computed by observing the system under consideration of its behaviors. Once we decide on the different parameters of the system, we can model it using DEVS formalism and then execute the simulation for performance evaluation and/or exploring possible setups of the system until we find an acceptable system behavior [29].

In FD-DEVS, basic models or atomic models are connected into hierarchal models. The basic model consists of the following features [29]:

- The set of input ports through which external events (messages) are received.
- The set of output ports through which external events are sent and interact with other properties.
- Two distinct parameters for each state which are called “phase” and “sigma”. The phase represents the current state. Sigma defines the time period during which the model stays in the corresponding phase. For example, in ON-OFF model, for the active phase, sigma = ON T and for inactive phase, sigma = OFF T.
- The time advance function which keeps the time management of the model by monitoring the clock cycles and the sigma values of all models.
- The internal transition function (deltint) specifies the next state to which a model has to transit after some specified time.
- The external transition function (deltext) specifies how the model should alter its behavior by changing the state given some inputs have been received that affect the current model state.
- The confluent transition function specifies the next state a model has to transit if a transition to a state occurs at the same time when an input event is received.

These three functions: internal transition function (deltint), external transition function (deltext), and the confluent transition function provides a comprehensive tools to model thoroughly all system interactions that could
be possible between the components in a specific model. The output function generates and wraps a message just before an internal transition occurs and sends it through the interconnection links between the different models.

2.3 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.

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. For more details on DEVS/Service Oriented Architecture system specifications and services, refer to [20], [30], and [31].

3. SYSTEM DESIGN AND THE MODELING METHODOLOGY

3.1 Generic Domain-Independent Marketplace Architecture

The rules of negotiation are enforced by a trusted third party marketplace model which supervises the whole negotiation process while preserving privacy and transparency among the system users. Finite and Deterministic Discrete Event System Specification formalism (FD-DEVS) is used to implement the generic marketplace model. The domain-independent marketplace enforces two negotiation protocols:

1- One-to-One negotiation: This has been used widely in negotiation domains. In this protocol, the customer agent sends messages (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 messages and forwards them to the specified service provider. On the other side, the service provider responds by sending (offer, accept, reject) including the customer ID in the contents of the messages. The marketplace receives the messages, finds the customer ID, and then it forwards them to the specified 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 1 shows the protocol flow along with the marketplace states.

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 best providers for a specific service. Since service providers advertise their services, information and products to the marketplace, the marketplace keeps a database of the members of service providers and their capabilities. The marketplace responds to the customers with a group of service providers who can fulfill their requests. The customers then can 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, and then the marketplace forwards that to the selected providers that were chosen in the previous step. After that, it will wait in phase “Wait” to receive responses from the providers one by one. Once it finishes waiting, 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 2 shows the protocol rules along with the states of the marketplace scenario. Notice from figure 2 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 the agent can send back to agent A either “Accept”, “Reject” or a new “Offer”. 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 timer, once it expires, the agent sends a “Terminate” message. If the marketplace receives a complaint regarding a transaction (“Item”), it interacts with the item provider to resolve the issue (the two ways dotted arrow in figure 2). Such an interaction depends on the regulations of companies. 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.

Figure 1: One-to-One negotiation protocol
3.2 Language of Encounter Ontology Design

In our design of the generic negotiation model, we specified the language of encounter that system users need to use for their interactions. O’Hare and Jennings [32] suggested three groups for language of encounter. However, such a classification is not enough to truly enable the negotiation process in a flexible and generic way. We have defined two new necessary classes for messages to increase the expressiveness power and the negotiation capabilities as in table 1. The difference between “Decline” and “Reject” is that when the marketplace is too busy and cannot handle more requests. It might not choose to start the negotiation process by sending a “Decline” message. 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. “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 acknowledged that, but after establishing the link between them, the service provider was transferring the traffic with speed less than 200KB, then the requester can ask the marketplace to terminate the contract by sending a “NotMet” message. “Terminate” message means that the negotiation process has started but is not finished (still in progress and the result is not known yet).

The dynamic structure of the language of encounter is implemented in System Entity Structure (SES) ontological framework. Each negotiation message has a separate ontology that defines its structure under different domain specialization entities as shown in Figure 3 and 4.

<table>
<thead>
<tr>
<th>Abort</th>
<th>Initiators</th>
<th>Reactors</th>
<th>Completers</th>
<th>Informative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminate</td>
<td>ContractQuery</td>
<td>Offer</td>
<td>Reject</td>
<td>Busy</td>
</tr>
<tr>
<td>NotMet</td>
<td>CapabilityQuery</td>
<td>CounterOffer</td>
<td>Accept</td>
<td>LinkEstablished</td>
</tr>
<tr>
<td></td>
<td>ItemRequest</td>
<td>Decline</td>
<td></td>
<td>Item</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CapabilityStatement</td>
<td></td>
<td>ItemCheckResult</td>
</tr>
</tbody>
</table>

Table 1: Classification of the language of encounter

![Diagram of Accept message ontology for oceanography and OnlineStore](image)

Figure 3: Accept message ontology for oceanography and OnlineStore
The domain-independent marketplace integrated with the domain-dependent language of encounter ontology gives system designers a very powerful tool to benefit from as shown in figure 5. Our modeling approach that we developed in this research works as follows: given the language of encounter structures under a specific domain of interest and the domain name (both as inputs) produces a tailored integrated negotiation marketplace model that is ready to be used.

We have evaluated our methodology under two different domains (applying it to other domains follows the same approach). The first example shows how brokering can lead to a data transformation contract from a data collector (such as a sensor) to a data requestor under the oceanography domain. The data collector can be changed dynamically through the use of the negotiation protocols. In the second experiment, we apply our system to the
domain of distributed software services environment in which, services providers can do different job capabilities. In this context, we used print servers as our service providers. Since some print servers provide similar capabilities as others, and some provide services that none of the other can provide, negotiation over the capabilities is necessary. We deployed our negotiation framework under web services environment (DEVS/SOA). Our modeling methodology provides the infrastructure that supports different domains with different negotiation requirements.

3.3 Steps in the Marketplace Generation

Designing the negotiation system is a very time consuming task which consists of many steps. We divided the steps here into two groups. The first group is regarding defining the dynamic message structure ontology. The second group is for designing the marketplace phases, transitions and output in FD-DEVS formalism. To design the language of encounter ontology for a specific domain, the system designer needs to follow the following steps:

1. Writing an SES natural language that describes the language of encounter’ ontologies. This requires from the designer to write each message structure for each specific domain.
2. Using SES builder tool [33], to create the ontology structure in SES XML schemas. The SES builder is an efficient tool for Knowledge Representation and data engineering and ontology design [34]. SES builder is also useful to prune SES XML files [18].
3. The result of the second step associates each negotiation primitive with a SES schema. Java Architecture for XML Binding (JAXB) allows users to map Java classes into XML representations and vice versa [35]. JAXB compiler takes XML schemas as inputs and produces Java classes and interfaces [36]. The negotiation system designer can use the JAXB compiler to create negotiation messages packages that can be plugged directly into Java files (our objective is to use them in the marketplace implementation).
4. The output packages of the JAXB compiler can be used now in the marketplace Java file.

To create the marketplace negotiation protocols in FD-DEVs, the designer can use the FD-DEVS GUI tool [37], which is a useful tool to generate Java templates, to create the marketplace states and transition specifications. The following steps are to be performed by the designer:

1. Use FD-DEVs GUI to define the marketplace phases, the internal function and the external function tables. The tool will result in two files. One is an XML representation of the model and the second is a Java file.
2. Take the Java file which is a domain-independent generic marketplace template for the negotiation protocols.

In order to integrate the language of encounter Java packages with the domain-independent marketplace, the following steps must be carried out:

1. Importing the specific domain message classes into the marketplace model. For example, if the designer is developing an oceanography domain negotiation system, then the designer must import the specific messages for the oceanography domain. If another designer wants to develop online store negotiation system, then the new designer must import the negotiation messages for OnlineStore domain. As a result, based on the domain of interest, the designer must import the same domain messages packages.
2. Remove the messages definitions of the generic marketplace model and define new messages classes based on step 1.
3. Unwrap messages classes and wrap them in the deltext method and the out method in order to provide the capabilities of sending data or receiving data. For example, under the oceanography domain, the user needs to access the structure of the “Accept” message using its class (specialization and decomposition properties) which includes: RequestorName and SensorName.
4. The phase ProcessingCapability shown in figure 2, suggests that the marketplace receives a “CapabilityQuery” to find the appropriate providers for a specific job. Hence, the marketplace needs to access its database (in the form of pruned XML files) to unmarshal data in order to send them back to the requestor via a “CapabilityStatement” message. The designer must handle this process by adding the correct JAXB unmarshalling code.

Figure 6 shows the flow of the manual steps that the negotiation system designer needs to follow. The figure shows five time-consuming and tedious human interaction tasks that each designer needs to go through before starting to tune up dynamic coupling and decoupling in the hierarchical model. Writing SES natural language needs a lot of care to avoid syntax errors, in addition to that, each message in the language of encounter needs a separate SES natural language, which results in 17 different text files. The second step needs to import each of the 17 SES
text files into the SES builder and create the SES XML schema. The third step will need a 17 system commands for each of the SES schemas to convert them into Java packages using JAXB compiler. In order to import a domain specific message structure, we need to write in the header file of the marketplace Java file many lines of codes to import the correct messages. In the last step, a lot of work needs to be done. Unwrapping each of the messages in the deltext method and wrapping each message in the out method consumes a lot of time and effort.

We have overcome all these tedious and time consuming steps by developing a tool that does most of the work on behalf of the designer. The tool reduces the human interactions into two very simple inputs from the designer. The overall automated pipeline of the marketplace generation is shown in figure 7.
4. SIMULATION AND PROOF OF CONCEPT

4.1 Printing Jobs Models Deployment in DEVS/SOA Environment

We applied our negotiation methodology to a distributed environment using DEVS/SOA. We called it PrintingJobs domain. Also we applied the methodology to tailor the system for oceanography domain. Oceanography is part of 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 to act before destruction can happen. Many authorities and governments have radars and sensors collecting data about the oceans all day time, trying to detect any oil slicks, tsunami, earthquakes, and volcanoes activities. Negotiation over data providers is important to keep track of changes on the sea level. Brokering can lead to a data transformation contract from a data collector (such as a sensor) to a data requestor. We applied the negotiation system to show how the data collector can be changed dynamically through the use of the negotiation protocols.

Here we will discuss the PrintingJobs domain. In this environment different sites provide different printing capabilities services such as business cards, journals, embossing. We deployed ten models on five different machines. The ten models are divided into: seven printing servers, a customer or user, marketplace, and a coupled model.

The customer starts the negotiation by sending a “ContractQuery” message to the marketplace. The message contains its requirements. The marketplace uses the service discovery protocol and replies with the appropriate service provider. After the customer receives a set of offers, a decision will be made on whether to accept an offer or negotiate with a specific provider using one-to-one negotiation. At the end of the simulation we analyzed the offer that was accepted by the customer and it falls into the customer criterion. The customer established a link with the provider and received the service. This scenario is very similar to negotiation on tasks’ computing time in Grid computing environments.

This section ends our objective of the DEVS/SOA implementation which is a proof of the concept that our system can be used in different web-based 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 methodology to support different interaction behaviors. This can be done by using the 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.

5. CONCLUSION

We have constructed a negotiation system that supports brokering between service providers and requestors under different domains. Two powerful and yet flexible negotiation protocols are used to enforce the rules of interactions. The rules are implemented in a trusted third party marketplace model which supervises the whole negotiation process while preserving privacy and transparency among the system users. Discrete event modeling and simulation environment (DEVS) is used to implement the generic marketplace model. In order to accompany the negotiation protocols with flexible expressive primitives to handle negotiation behaviors in complex distributed systems, a dynamic structure of the language of encounter is implemented in SES ontological framework. Each negotiation message has a separate ontology that defines its structure under different domain specialization entities. The domain-independent marketplace design integrated with the domain-dependent language of encounter ontology gives system designers a very powerful tool to benefit from. We are not aware of any tool or methodology that tries to model negotiation systems under different domains.

The negotiation system is evaluated by applying our system to the domain of distributed software services environment in which, service providers can do different job capabilities. In this context, we used print servers as our services providers. Since some print servers provide similar capabilities as others, and some provide services...
that none of the other can provide, negotiation over the capabilities is necessary. We deployed our negotiation framework in Web Services environment (DEVS/SOA). Each one of the nodes has its own data pruned entity structure and running one or more of the print server capabilities.

We aim to develop a Web Services Description Language (WSDL) interface for the services of the marketplace and the service providers.

REFERENCES

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


