A Multi-Agent Simulation Framework to Support Agent Interactions under Different Domains

Moath Jarrah*, Bernard P. Zeigler*, Chi Xu†, and Jie Zhang*

*School of Computer Engineering, Nanyang Technological University, Singapore
*Chief Scientist, RTSync Corporation, Arizona, USA
†Singapore Institute of Manufacturing Technology, 71 Nanyang Drive, Singapore
{hmoath,zhangj}@ntu.edu.sg
berniezeigler@rtsync.com
cxu@simtech.a-star.edu.sg

Abstract. The ability to study complex systems has become feasible with the new intensive computing resources such as GPU, multi-core, clusters, and Cloud infrastructures. Many companies and scientific applications use multi-agent modeling and simulation platforms to study complex processes where analytical approach is not feasible. In this paper, we use two negotiation protocols to generalize the interaction behaviors between agents in multi-agent environments. The negotiation protocols are enforced by a domain-independent marketplace agent. In order to provide the agents with flexible language structure, a domain-dependent ontology is used. The integration of the domain-independent marketplace with the domain-dependent language ontology is accomplished through an automatic code generation tool. The tool simplifies deploying the framework for a specific domain of interest. Our methodology is implemented in FD-DEVS simulation environment and SES ontological framework.

Keywords: Multi-agent modeling and simulation, business process, negotiation, ontology, automate, FD-DEVS, SES.

1 Introduction

Agent based simulation platforms have been in research in the last decades with the promising that they will give understandings to natural phenomena. However, the computing power was not feasible to tackle the ever-increasing complexity of systems. With the current development in high performance clusters, cloud environment, multi-core, and graphical processing unit, scientists and decision makers are even more ambitious that multi-agent modeling and simulation field will help them study complex systems such as in social sciences, command and control, business processes, chemical reactions, forecast, and many others. Simulating and understanding a system helps decision makers to be proactive and not reactive. Also, it helps companies to make proper decisions to increase their profit and lower their risks. A system consists of many components and

each component has its own behavior and interacts with other components in the system. A natural modeling for systems is to represent each component as an agent. Agents interact according to rules of interactions. Some agents are intelligent and go through evolution as in the artificial life proposed by Epstein and Axtell in their famous Sugarscape model of artificial societies [6]. Multi-agent based modeling and simulation is needed because systems are becoming more and more complex in their heterogeneity, intelligence, and interactions. In this regard, SciDAC and Argonne lab are collaborating to build a high end framework that demands huge processing power (where IBM Blue gene is to be used) to tackle the most challenging scientific phenomena in the universe through simulation, visualization, and analysis [12] [13] [18].

Researchers in the domain of industry have been using agent based modeling and simulating for their business processes in order to achieve better performance. One application of a business process is in stock markets. In this application, investors try to buy stocks at low prices and sell them when the stock price is high. Investors try to maximize their profits. Hence, taking risks and studying historical data profiles are very useful in making predictions. Examples of such models as in [2] and [16], where heterogeneous agents buy, sell, and hold stocks and bonds.

Supply chain is another suitable domain for applying the multi-agent simulation framework where a network of retailers, distributors, factories or warehouses, and suppliers interact to achieve profits. Every entity is represented by an agent in the simulation and the output results are used to analyze the supply chain performance to better plan and predict future rules. Many industrial companies have focused on this domain of research such as the work by Boeing on automation of their supply chain interactions[11].

The ability to manage and utilize software and/or hardware products and services in current complex 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. Many attempts exist in literature where solutions were proposed to exploit these resources such as in [4] and [5]. However, the development of methods to experiment and study complex systems is still far from being sophisticated.

Complex systems are dynamic and seldom static in the sense that new components emerge and some components disappear. An example on complex systems can be the Web services. Web Services developments are growing dramatically 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. Hence, a natural modeling and simulation solution is to map a complex system into a multi-agent simulation environment. More on multi-agent design issues and challenges can be found in [19]. Manual software management is not feasible in such dynamic behaviors because of the number of service providers and the heterogeneity in their information management. In

this work, we aim at providing a generic automated integration of negotiation protocols with application-specific messaging capabilities. The framework provides a flexible and easy to implement, design, tailor, and deploy a modeling and simulation platform under different applications.

Negotiation rules are mechanisms to allow agents' interaction in order to achieve their goals. However, in designing negotiation systems, the designer needs to address three main issues [1]. First, negotiation techniques should provide brokering in managing loosely coupled service providers. Secondly, the engineering design of management tools should provide enough expressive capabilities for various behaviors or when different domains are encountered. Thirdly, lack of interaction between different requesters and providers yields inefficient and very costly agreements. In summary, negotiation systems should provide an interaction environment where many parties (agents) can be engaged in the negotiation using flexible rules and powerful language capabilities.

In order to reach to a successful modeling and simulation framework design, we identified the following issues to be supported:

- The framework should provide flexible brokering and negotiation capabilities.
- The framework should provide transparency to its subscribers whenever they need.
- New resources should be able to subscribe in a simple and efficient way.
- The framework should provide decision making capabilities on behalf of the agents whenever a user's agent requires it.
- The framework must provide simple and rich expressive primitives and be able to specialize them under different domains in a simple and automated approach.
- The design of the framework must be easy to develop it for a specific domain to shorten the development time which is vital for adding and removing new heterogeneous components to the simulation environment.

Hence, in this work, we show how we developed a flexible, efficient, and automated agent-interaction model that can be utilized by different engineering domains [8]. The model defines different concepts and principles in the negotiation process. Our method consists of an automated domain-independent marketplace architecture that allows agents to interact using two simple and yet powerful negotiation protocols which define the rules of interactions in multi-agent environments. Having a third party "marketplace" supports privacy and transparency among interacted agents. The marketplace agent is implemented using FD-DEVS [7]. In order to provide negotiation in different domains, a dynamic message structuring capability is needed where a message can have different formats based on the selected application. We develop an ontology that contains specialization relations between the different domains of interest [9]. We focus in this paper on the automation and integration of the domain-dependent message structure ontology with the domain-independent marketplace architecture. This paper shows how a designer of a multi-agent platform will have a powerful tool where systems can be tailored based on the operational purpose and objectives.

This paper is organized as follows: section 2 provides a summary of the negotiation protocols for agents interactions. Section 3 provides a summary of the ontology design and message specialization. Section 4 discusses in details the automation phase in developing a tailored platform for a specific application. Section 5 shows a running example for digital photo and printing markets; and finally, we conclude our paper in section 6.

2 Agents Interactions

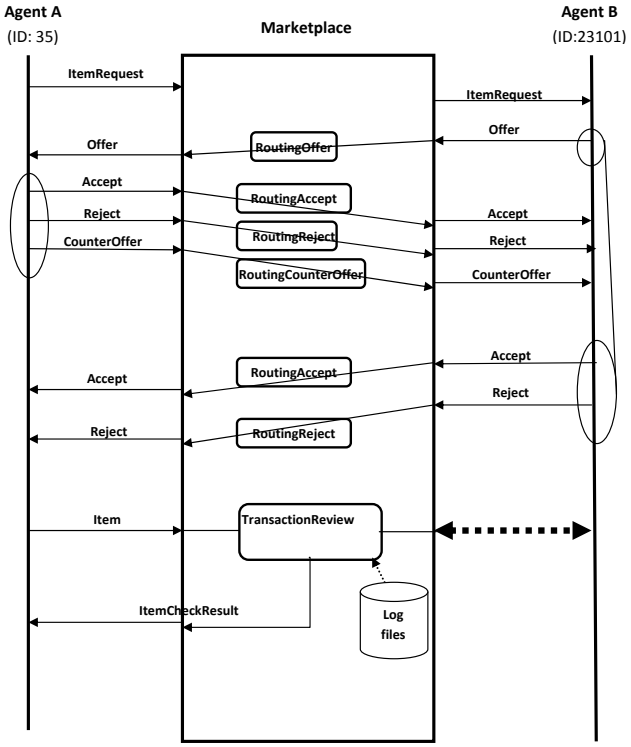

Fig. 1. One-to-One Negotiation Protocol

In most of business and distributed systems, managing the resources and services manually is impossible and autonomous agents are needed to act on behalf of system users. Agents interact using a negotiation process, which is a methodology that is applied to provide bargaining and brokering capabilities between the different agents in a multi-agent environment. In multi-agent domain, agents are not just capable of making decisions in predictable situations, but also they

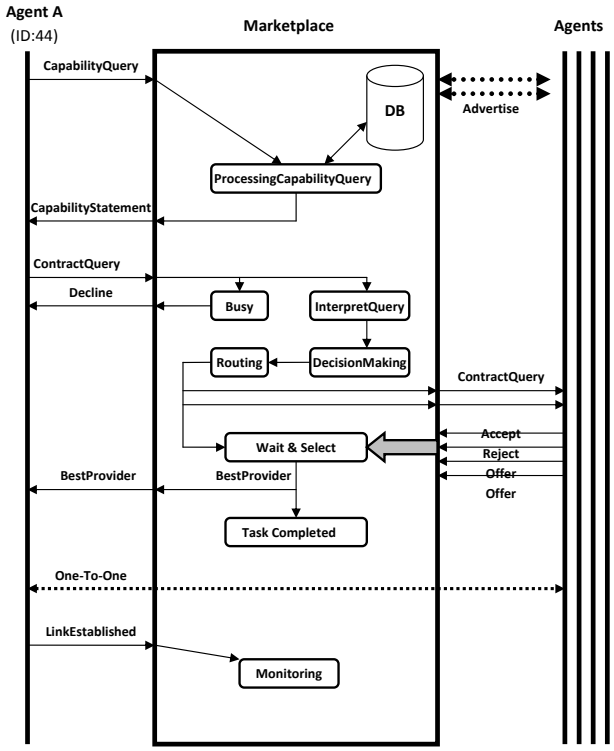

Fig. 2. Service Discovery Negotiation Protocol

can be intelligent to act in dynamic interaction. The agents need to communicate with each other, exchange data, and share same semantic language (can be an ontology). The objective of interactions is to reach to some agreements that are acceptable by those agents who are engaged in the negotiation. Game theory is a branch of economics that is concerned with interactions between agents to reach to decisions [10] [14] [17]. It imposes mathematical models (functions) that describe each agent's utility function in multi-agent systems in which strategically each agent tries to maximize its individual profit or preference. Autonomous agents have been used in many areas such as search engines, where they crawl the Web to find data or information.

In this framework, we have designed the interaction between agents through a marketplace that insures two negotiation scenarios: one-to-one rule, and service discovery rule. Figures 1 and 2 show the two negotiation scenarios. In one-to-one rule, the marketplace forwards a message received from an agent to the corresponding agent using the destination ID field. Hence, in the one-to-one rule, each agent should know the corresponding agent ID. An agent can send to many agents using their IDs. In the service discovery, an agent sends a query to the

marketplace which has access to the database of all subscribers, to find service providers. The marketplace responds to the request with a group of service providers or agents (IDs) who potentially can fulfill the request. An agents uses the list of the available service providers (agents) and their capabilities to engage in one-to-one negotiation to reach to an agreement on a specific contract. Any agent can decide on whether to proceed with the negotiation process or not. If the agent chooses to proceed with the interaction, it sends a contract query message to the marketplace agent, and then the marketplace in turn forwards that message to the appropriate agents and wait for responses from them. For more details on the negotiation protocols, refer to [8]. DEVS formalism was used to implement the states of the agents. Coupling and de-coupling is used between agents to reflect the interaction. The messages are sent through the in ports and out ports of the agents.

3 Language of Interaction

Agents interact with each other according to the rules of interaction (negotiation protocols). During an interaction, they send and receive different messages. Messages (for example, *Offer*) carry different information according to their structure (fields). For example, you can have the message *Offer* to contain three fields of information and have the message *Link Established* to have 7 fields, and so on. We have identified five groups of messages. The total number of messages are 17 divided into five groups as shown in table 1. Detailed description on the messages can be found in [8].

In order to support negotiation services under different domains, a dynamic message structure is implemented using shared ontology that defines each message and its structure under different domains [9]. Each domain would be a specialization of the message. Each message type has a separate structural ontology defining its variables/fields. System Entity Structure (SES) formalism is used to define the specialization and structuring of each of the messages [20]. Hence, the user of our platform decides the structure of each of the messages to be used in the interaction. The agents who engage in the interaction should be designed to utilize these information embedded in the messages.

Table 1. Classification of the Messages

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

3.1 Specialization and Structuring

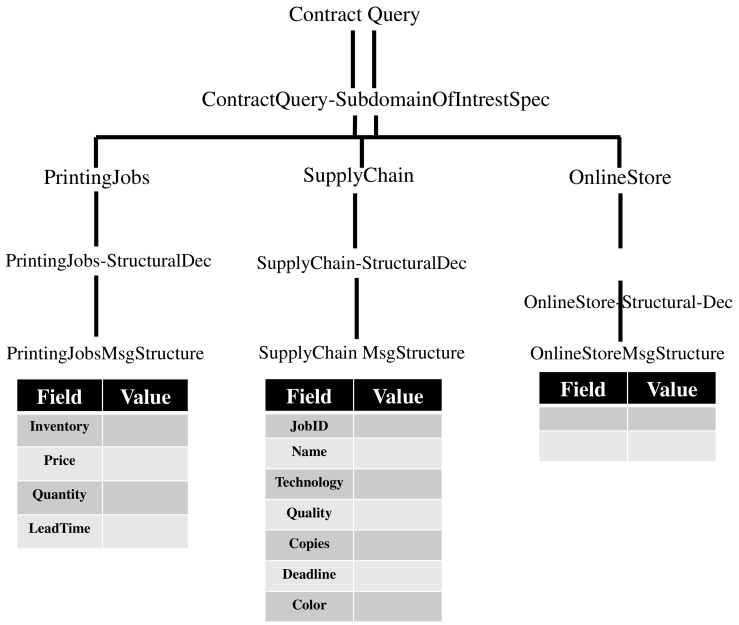

Fig. 3. Contract Query Specialization and Decomposition under Different Domains

In this subsection, we discuss how the message (language of encounter) is designed and implemented in order to support many application in a simple and automated way. Each message has a SES ontology that defines its structure under different domains. In this paper, we consider three applications as examples which are: Supply chain, photo development market [3], and a generic online store (e-commerce site). We will explain how to build the ontology for one message (*ContractQuery*) and the rest follow the same methodology. In supply chain domain, supplier and demand agents interact on contract variables that usually include: inventory level, price, product quantity, and lead time. Hence, a natural *ContractQuery* message will contain the aforementioned fields. In the domain of photo printing/development market, a contract usually consists of variables such as: print job ID, customer name, technology type, paper quality, number of copies, deadline, color. Hence, autonomous agents in a multi-agent environment exchange (interact) offers and counter offers in order to reach to an agreement that is acceptable by the engaged agents. Figure 3 shows how the ontology of the *ContractQuery* looks like.

4 Automatic Tailored Framework Generation For a Specific Application

In this section, we show the integration details and the automatic code generation capabilities. The integration is required to attach the interaction rules with the message structure in order to result in a working platform where agents can communicate, understand, and negotiate to accomplish their tasks. The implementation and proof-of-the-concept are done in DEVS environment along with the SES ontological framework.

4.1 Integration of the Domain-Independent Negotiation Protocols with the Domain-Dependent Ontology

The marketplace that enforces generic (domain-independent) negotiation rules is created using FD-DEVS GUI tool [9], which is a useful tool to generate Java templates [21]. The output of the tool is a Java file which is a domain-independent generic marketplace template that enforces the negotiation protocols.

In order to integrate the message structures into the domain-independent marketplace, the system designer must create Java classes for each message type consisting of the fields that are defined under the domain of interest. For example, if the designer is developing a framework for supply chain, then all messages specialization for supply chain must be imported. If another designer wants to develop the system for printing jobs, then the corresponding Java classes must be imported into the marketplace Java code. Hence, based on the domain of interest, the designer must manually import the same domain message packages, and that requires plenty of coding since there are 17 different messages and each has different structure according to the application.
Also, it requires the designer to manually unwrap message classes and wrap them in the marketplace code and the database where all subscribers information exist. For more details, refer to [8].

In summary, the manual steps require writing SES natural language for each message and domain and care must be taken in not making syntax errors. After SES natural language, it is required to run the SES builder on each of the 17 SES text files to create SES XML schema (ontologies). Afterwards, a conversion from SES schemas to Java packages using JAXB compiler (requires 17 different commands) is needed. Deceleration and adding many lines of codes to the header file of the marketplace Java file requires tedious work from the designer whenever a modification or a new application is to be supported. The following subsection shows how we automated all the tedious and time consuming steps by developing an automatic code generation tool that does the work on behalf of the designer. The tool reduces the human interactions into two very simple inputs from the designer using GUI interface.

4.2 Automatic Generation and Integration of the Negotiation Marketplace

In order to help the designer in defining the message structures, we have developed a simple and easy to use Graphical User Interface which is shown in figure 4. The user of the GUI can add any domain (for example supply chain) to a message ontology; and define the structure of that message under the domain. For example, The user inputs how many fields the message *ContractQuery* has, and the name of each field in the structure. The user or the designer might select four fields with the name: Inventory, Price, Quantity, and Lead time. If a message type is not used in agent interactions, then the designer can select that the number of fields for that specific message is zero.

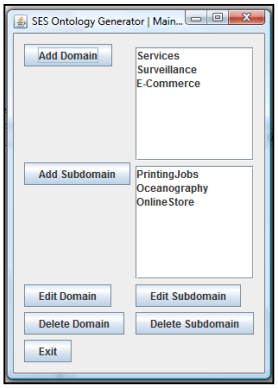

Fig. 4. Domain-Dependent Ontology Creation GUI

The output of the GUI tool is a collection of SES natural language text files (17 files), one for each message type. Those files are automatically converted into XML representation of the SES ontology in the automatic code generation tool. The tool then converts the XML representation (String sesinxml) into an XML schema by executing the line of code:

String schema = XmlToSes.getSchema(sesinxml);

The returned value of the above code is a schema, where the tool writes automatically 17 schema files for each of the messages into files with the extension .xsd to prepare them for the JAXB compiler. The SES schema is the representation of a master ontology that contains all domains that were defined. A Java method (*ExecJAXBSchemaCompiler*) is called to execute the JAXB compiler to translate the schema files into Java classes using the compiler command (with the appropriate parameters):

xjc schema.xsd d dirName p PackageName

The method iterates on each file and executes the command. Then another Java method named (*PostProcessingJavaClasses*) is called to extend (derived class) of type (*entity*) which is the base class for exchanging messages in DEVSJAVA. Also, the Java method imports the package (*import GenCol.*;*). At this point, the Java packages are complete and can be used by the marketplace generic model to declare the appropriate messages for the specific domain of interest.

4.3 Tailoring of the Marketplace Agent for a Specific Application

The first step in designing the framework is simple as mentioned in the previous subsection, which is to use the GUI to define the message structures. The second step that needs human interaction is very simple as well and all what it needs is to call a Java method (namely *CreateFDDEVSModelFor*) with the domain of interest as a string input such as Supply Chain" or "PrintingJobs. The Java method *CreateFDDEVSModelFor* uses the marketplace XML file that was created in FD-DEVS which defines the negotiation protocols, the in ports, and out ports of the marketplace agents. The in ports and out ports are through which messages are received and sent respectively. Hence, if the designer executes the code:

CreateFDDEVSModelFor("Supply Chain");

Then it generates a tailored marketplace agent model for supply chain domain. The model enforces the negotiation protocols and the message structures defined using the GUI interface for supply chain. The Java method *CreateFDDEVSModelFor* is responsible of performing the following code generation steps:

1. Import the required Java classes for the negotiation process to take place, and the appropriate message package for a specific application.
2. Declare an instance of each of the negotiation primitives (messages).
3. When receiving a message, it stores it in the corresponding local instance produced in step 2. Then it generates the appropriate code to unwrap the message to get the domain message structure class that has the *get* and *set* methods to allow the designers to access the data received or to set variables to be sent in the message. The objective of storing the messages into local variables provides the capabilities for future data access and processing.
4. Create the JAXB Unmarshaller code to provide the marketplace agent to access its database when search for subscribers services.
5. Prepare domain message structure classes and wrapping them into the corresponding primitive; and then send it through the appropriate out port. This step provides the designer with the flexibility of adding *setV* methods to marshal the messages with data as needed.

In summary, this section showed how we automated the process of generating the marketplace agent given the language of interaction and the domain of interest. For example, if the message is ContractQuery and the domain is PrintingJobs, the tool will select the pruned SES of the ContractQuery ontology that defines the message structure under the domain "PrintingJobs".

5 Experiments and Proof of the Concept

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 the request or just any provider that can meet the requirements. In this section we show how agents interact to reach an agreement on a contract for online printing jobs; similar to the photo development market where an agent tries to get the service, within specific deadline, price, quality, and color. The marketplace agent helps all negotiating parties to reach to an agreement. For example, if a customer is concerning with printing business cards, the customer might choose thermography, Engraving or Letterpress technology. Also, other aspects for paper could be quality, deadline, color and duplex.

We have designed a user agent model that is searching for a provider who has Business Cards printing capabilities. The user would accept an offer if a contract with the following conditions occurs:

1. If the paper quality is medium or high, the color is RGB and deadline is less than 30.
2. If the paper quality is medium or high, the color is full HD and the deadline is less than 80.
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 the previous conditions, the user sends back a counter offer asking for the first contract or a modified one based on the history of the offers that were received. In our model, we chose that the user sends the first preference back again.

For the service provider agents, we have arbitrary assigned different photo and printing capabilities using the following pool of technologies: Digital printing can be : Brochures, Journals, Booklets, photos. Embossing Printing can be: Greeting Cards, Metals, Garments. Flexography Printing can be: Milk and Beverage Cartons, Disposable Cups, Containers, Adhesive Tapes, Envelopes, Newspapers, Food and Candy Wrappers. Letterpress Printing can be: Business Cards, Company Letterhead, Proofs, Billheads, Forms, Posters, Embossing, Hot-leaf Stamping. Engraving Printing can be: Stationery, Wedding Cards, Business Cards, Letterhead. Gravure Printing can be: Label, Flexible Packaging, Cartoning. Thermography Printing can be: Fax Printers, Business Cards, Letter Head, Invitation.

Some of the service agents are designed to update their deadline to accomplish the job using some constant value. Others do not and stick to the same deadline value through the whole simulation. The marketplace agent enforces the rules of interaction via reacting and routing the messages accordingly. We have used one agent to represent the user demand and seven agents to represent the service providers. After we ran the simulation in DEVS environment on a single machine, we found that an agreement has been reached with the following contract terms:
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.
The contract terms that was reached actually satisfy the conditions in the second item of the user agent decision making which is:

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

In order to provide a proof of the concept and deploy the framework in a distributed cluster; we have created the model agents and uploaded them on five machines in a DEVS/Service Oriented Architecture cluster. 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. For more details on DEVS/SOA system specifications and services, refer to [15].

The same logical behaviors as in the single machine simulation is used for DEVS/SOA. One user agent is used and seven service provider agents were used, and distributed on five machines in the cluster according to table 2. As expected, agent (Print Server 6) established the agreement with the user agent.

Table 2. Agent Distribution in DEVS/SOA Cluster

Machine IP	Agent
150.135.218.200	Customer, Print Server 2, Print Server 4, Print Server 5, and Print Server 7
150.135.218.201	Print Server 1
150.135.218.203	Print Server 3
150.135.218.204	SOAMarketPlace and the Coupled model (ServicesSOAEnv)
150.135.218.206	Print Server 6

6 Conclusion

Two powerful and yet flexible negotiation protocols are used to enforce the rules of interactions in multi-agent modeling and simulation platform. The rules are implemented through a third party marketplace agent, which supervises the negotiation process while preserving privacy and transparency among the system users. Discrete event modeling and simulation environment (DEVS formalism) is used to implement the generic marketplace model. In order to accompany the negotiation protocols with flexible messaging capabilities to handle different complex domains and applications, a dynamic structure of the language of interaction is implemented in SES ontological framework. Each negotiation message has a separate ontology that defines its structure under different domain specialization.

The domain-independent marketplace model integrated with the domain-dependent language of interaction ontology gives system designers a very powerful and easy to use tool. Given the language of interaction structures under a specific domain of interest and the domain name as inputs to the automated code generation tool, it produces a tailored negotiation marketplace agent that is ready to be used. The automated marketplace code generation deceases significantly the time spent to tailor the platform for a specific domain. Different applications in industry and academia can utilize our framework to study their processes and phenomenon while reducing the development time and changing the messages contents in a flexible and simple way.

Acknowledgment. This work is supported by the SIMTech-NTU Joint Lab on Complex Systems.

References

1. Addis, M. J., Allen, P. J., Surridge, M.: Negotiating for Software Services. In: Eleventh International Workshop on Database and Expert Systems Applications (DEXA2000), September (2000).
2. Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R., Tayler, P.: Asset pricing under endogenous expectations in an artificial stock market. In: The economy as an evolving, complex system II, pp. 15-44. Addison Wesley, Redwood City, CA (1997).
3. Chan, Y., Chen, X., Chou, M., Goh, B. H., Haw, C. S., Koh, S., Lee, h. K., Ye, h. Q., Yuan, X. M.: Analysis of a Software Focused Supply Chain in Photo Development Market. In: IEEE International Conference on Industrial Informatics, pp. 759-764, August (2006).
4. Cooper, T.: Case studies of four industrial meta-applications. Lecture Notes in Computer Science, 1593, pp. 1077–1086, (1999).
5. DISTAL, Distributed Software On-Demand For Large Scale Engineering Applications, `http://cordis.europa.eu/esprit/src/26386.htm`.
6. Epstein, J. M., Axtell, R.: Growing artificial societies: social science from the bottom up. Brookings Institution Press, (1996).

7. Hwang, M. H., Zeigler, B. P.: Reachability Graph of Finite and Deterministic DEVS Networks. IEEE Transactions on Automation Science and Engineering, 6, pp. 468–478, July (2009).
8. Jarrah, M, Zeigler, B. P.: A Modeling and Simulation-based Methodology to Support Dynamic Negotiation for Web Service Applications. journal Simulation. 88, pp. 315–328, March (2012).
9. Jarrah, M., Zeigler, B. P.: Ontology-based marketplace for supporting negotiation in different scientific applications. In: IEEE Conference on Systems, Man, and Cybernetics (SMC), pp. 667-672, October (2012).
10. Krishna V., Ramesh VC.: Intelligent Agents for Negotiations and Market Games, Part 1: Model. IEEE transaction on Power Systems. 13, pp. 1103–1108, August (1998).
11. Kruse, S., Brintrup, A., McFarlane, D., Sanchez, L. T., Owens, K., Krechel, W.E. :Designing Automated Allocation Mechanisms for Service Procurement of Imperfectly Substitutable Services. IEEE Transactions on Computational Intelligence and AI in Games, 5, pp. 15–32, March, (2013).
12. Macal, C. M., North, M. J.: Agent-Based Modeling and Simulation: Desktop ABMS. In: 39th Conference on Winter Simulation, WSC07, pp. 95-106. IEEE Press, NJ, USA (2007).
13. Macal, C. M., North, M. J.: Agent-based modeling and simulation. In: Conference on Winter Simulation, WSC'09, pp. 86-98. (2009).
14. Mahajan R., Rodrig M., Wetherall D., Zahorjan J.: Experiences Applying Game Theory to System Design. In: Proceeding SIGCOMM PINS Workshop, pp. 183–190, (2004).
15. Mittal, S., Risco-Mart J. L., Zeigler, B. P.: DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process. Simulation Journal, 85, pp. 419–450, July (2009).
16. Palmer, R. G., Arthur, W. B., Holland, J. H., LeBaron, B., Tayler, P.: Artificial economic life: a simple model of a stock market. J. Physica D. 75, 264-274 (1994).
17. Persons S., Wooldridge M.: Game Theory and Decisions Theory in Multi-Agent Systems. Journal on Autonomous Agents and Multi-agent Systems. 5, pp. 243–254, September (2002).
18. Scientific Discovery through Advanced Computing, agent-based modeling and simulation for exascale computing, `http://www.scidacreview.org/0802/html/abms.html` (2014).
19. Susan E. L.: Issues in Multi agent Design Systems. Journal IEEE Expert: Intelligent Systems and Their Applications, 12, pp. 18–26, March (1997).
20. System Entity Structure, SES, `http://www.ms4systems.com/pages/devs/ses.php` (2014).
21. W3C XML Schema for Finite Deterministic(FD) DEVS Models, `http://www.duniptechnologies.com/research/xfddevs/` (2014).