Building Holonic Supply Chain Management Systems: An e-Logistics Application for the Telephone Manufacturing Industry

Mihaela Ulieru, *Senior Member, IEEE,* and Mircea Cobzaru

***Abstract*—As an exercise in agent-based software engineering, this work proposes a holonic model for the domain of supply chain management. The supply chain system is a distributed infrastructure that enforces protocol rules and through which agents registered on a domain find each other, access the knowledge base, communicate (exchange messages), and negotiate with other agents, which are independent entities with specific goals and resources. It is considered that individual resources that belong to each agent are not sufficient to satisfy their goals; therefore, the agents must procure the needed resources from other agents present in the system through negotiation. Our approach is based on the holonic enterprise model with the Foundation for Intelligent Physical Agents (FIPA) Contract Net protocols applied within different levels of the supply chain holarchy. To accommodate differentiation of interests and provide an allocation of resources throughout the supply chain holarchy, we use nested protocols as interaction mechanisms among agents. Agents are interacting through a price system embedded into specific protocols. The negotiation on prices is made possible by the implementation of an XML rule-based system that is also flexible in terms of configuration and can provide portable data across networks.**

***Index Terms*—Holonic enterprise, supply chain holarchy, agent technology, nesting protocols, ontology, FIPA architecture.**

I. Motivation—Agent Technologies as Enablers of Today's Supply Chain Management Dynamics in the IT Driven Global Economy

A. Traditional Approaches

SUPPLY chain management (SCM) is the logical progression of developments in logistics management. Traditional supply chains begin at the point of manufacture and end with the sale to consumers. The focus of the supply chain professionals has been on the products that flow through the operations and logistics channels. SCM software applications provide analytical systems that manage the flow of product and information throughout the supply chain network of trading partners and customers. They are designed to improve SCM operations—supplier sourcing, production planning, inventory management, transportation, demand planning, and so on. Software producers for SCM compete in providing the most complete package of solutions for supply chain networks trying to integrate various models into a seamless, unitary system that brings maximum benefits for a company. The typical way of addressing these requirements is to employ a manufacturing resource planning (MRP) or enterprise resource planning (ERP) system that integrates a vast amount of information and makes this information available to a broader range of participants. The top software vendors and their product suites for SCM are SAP [15], Manugistics [11], J. P. Edwards [8], PeopleSoft [14], Oracle [13], and AppriseNet [1] all offer solutions that enable the transformation of the old linear supply chain mode into a global network through the integration of cross-department and cross-company processes. More recently e-Business solutions accompanying the SCM packages enable companies to manage the entire value chain across business networks. Most of the solutions proposed have a twofold focus: On one side, they enable enterprises to understand, predict, and manage customer demand effectively, leading to enhanced customer service, increased market share, and higher profitability, and on the other side, they provide more efficient ways to manufacture and distribute products.

Although the above-mentioned models can support a variety of business functions, there is no unified solution that is able to support all functions equally well. At present, SCM solutions are fragmented along functional applications into specific areas—for example, advanced planning and scheduling for the manufacturing plant, demand planning for the sales group, and transportation planning for the distribution center. The old way of delivering a product was to develop projections on demand, manufacture the product, and fill up warehouses with finished goods. Later in the 1960s, integrating warehousing and transportation functions provided inventory-reduction benefits from the use of faster and more reliable transportation. The next phase in SCM development was the "logistics stage" that incorporated manufacturing, procurement, and order management functions. The current stage is the "integrated supply chain management stage" [12]. This is viewed as a process-oriented, integrated approach to procuring, producing, and delivering products, and services to customers. It covers the management of materials, information, and funds flows.

B. Requirements for the Next Generation SCM Systems

The next generation of SCM systems will have to deal with the supply chain's growing complexity and should posses the following set of characteristics.

- *Integration:* This is the keyword that appears throughout the new generation of supply chain because it makes the difference between the old view of

Manuscript received June 12, 2004; revised December 16, 2004.

The authors are with the Electrical and Computer Engineering Department, The University of Calgary, Calgary, AB, T2N 1N4 Canada (e-mail: Ulieru@ucalgary.ca).

Digital Object Identifier 10.1109/TII.2005.843827

logistics as the discrete functions of transportation and distribution and the new vision of SCM that links all the participants and activities involved in converting raw materials into products and delivering them to consumers at the right time and at the right place.

- *Customer-centric service:* Providing online commitments for orders and schedules is a key factor to enhancing customer service and gaining a competitive advantage. Many companies envision this capability as part of the future SCM strategy. The next generation of SCM would solve customers' problems [3], [16] by
  - gathering and analyzing knowledge and data about customers' needs;
  - identifying partners to perform the functions needed in the demand chain;
  - moving the functions that need to be done to the chain member that can perform them most effectively and efficiently;
  - developing and executing the best logistics, transportation, and distribution methods to deliver products and services to consumers in a timely manner.
- *Synchronization:* The next generation of supply chain suites will have to synchronize supplier planning, production planning, logistics planning, and demand planning. These solutions will provide a comprehensive view of all supply chain activities and enable management to make more informed tradeoff decisions. Supply chain synchronization is the secret to improving customer service without increasing inventory investment.
- *Agility:* SCM systems must be able to process transactions rapidly and accurately. In today's business environment, organizations must focus on moving information and products quickly through the entire supply chain, distribution, assembly manufacture, and supply. The faster that parts, information, and decisions flow through an organization, the quicker it can respond to customer needs and orders.
- *Flexibility:* The next generation of SCM systems can create an advantage by being flexible enough to customize its services to meet the needs of distinct customer segments or individual accounts. The flexibility to meet diverse customer needs in a cost-effective way can distinguish a company and allow it to serve a wider customer category.
- *Information Protection:* In today's global economy, it is of the essence to ensure that information is securely protected while various companies (some of which may be competitors) collaborate within the same supply chain.

The typical way of building an SCM system to deal with the complex activities in the supply chain is to integrate several models, e.g., subsystems such as enterprise resource planning (ERP), active server pages (ASPs), product data management (PDM), etc., into a single system. The problem is that most of the subsystems mentioned above are proprietary, and their vendors provide interfaces to other types of subsystems (e.g., PDM to ERP) but seldom to another subsystem of the same type from a different vendor (e.g., ERP to ERP), which makes these components dependent on each other. This could make it difficult to combine different supply chain and enterprise systems due to lack of compatibility among them.

C. Case for Agents

Although autonomous agents have been used in enterprise integration and supply chain networks, they are still in the stage of experimentation and in course of being implemented as complete solutions. Agents are suitable for integrating supply chain functions because they can extend applications like production, distribution, and inventory management functions across supply chains spanning various organizations without the need for additional interfaces especially when a common infrastructure is used. Because supply chain management is concerned with coherence among multiple decision makers, a multi agent-modeling framework based on communications between constituent agents (such as manufacturers, suppliers, retailers, and customers) seems a natural choice to design and implement these dynamic environments.

All the traditional SCM systems are enormously complex by nature and it is quite difficult to optimize them as a whole, but using semiautonomous agents acting on a few simple rules can change the way the entire system is understood and optimized, just by modeling and changing the agents' behavior.

Agents can also expand the level for supply chain collaboration across multiple enterprises including resources, events, and the relationships between them, thus transforming closed trading partner networks into open markets.

An important aspect that deserves attention is order-tracking/reporting. Currently, this activity is done by direct communication between human personnel involved in the supply chain staff. Mobile agents can improve the current approach considerably by following the orders throughout the entire supply chain. This enhances the system with the capability to track "on the spot" the status of each individual order.

II. New Trends—From Supply Chain to Collaborative Cluster

Developments in artificial intelligence and multiagent systems have made it possible to apply agent technology not only to supply chain management but also to manufacturing planning, scheduling, and control.

Holonic manufacturing systems [4] have been explored in the past ten years as a new step toward distributed intelligent architectures for manufacturing [17]. They were also thought as an alternative solution to traditional architectures (e.g., computer integrated manufacturing systems) that have a low capacity to adapt and react to dynamic changes in the environment such as disturbances and market changes [17]. A holonic manufacturing system is composed of functional manufacturing units called holons—units that display the dual properties of autonomy and cooperativeness. A holon, as defined by the Holonic Manufacturing Systems (HMS) consortium [4], consists of an information processing part and often a physical processing part and can be part of another holon in a nested hierarchy. The activities

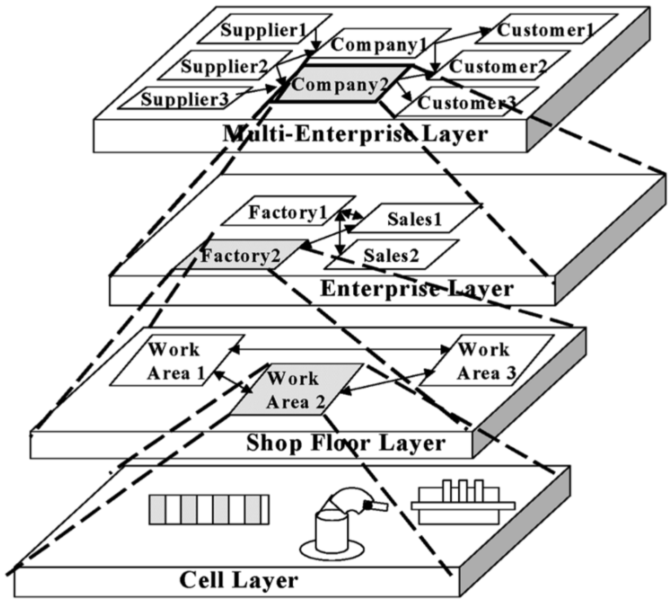


Fig. 1. Layers in Holonic manufacturing defining a manufacturing holarchy.

of each holon are determined through cooperation with other holons, as opposed to being determined by a centralized mechanism.

There is a close relationship between holons and agents [17] in that a holon can be mapped to groups of agents that together fulfill a certain functionality of the system. From this perspective, holons can be regarded as nested agents defining various levels of resolution within the system under discussion. As such, intelligent agents can be used to encapsulate existing legacy software systems and integrate manufacturing enterprises' activities such as design, planning, scheduling, simulation, execution, and product distribution, with those of their suppliers, customers and partners into an open, distributed environment [17]. The Holonic Enterprise can be viewed as an information ecosystem composed of collaborative but autonomous holons (see Fig. 1).

The Multi-Enterprise level models the interaction between distributed enterprises, which interact with their suppliers and customers. In the Enterprise level, we find the co-operation between geographically apart entities, the sales offices and the production sites. In the Shop Floor level, we find the distributed manufacturing control within a production site or shop floor. Here, the entities are distributed work areas working together and in co-operation, in order to fulfill all orders allocated to them. The basic level (the Cell) models the interactions between atomic holons (equipments and humans).

III. Implementation of a Holonic SCM System for a Global Enterprise—A Case Study for the Phone Manufacturing Industry

A. Agent-Based Application Development Methodology

To implement the agent-based application for supply chain management, we follow the methodology proposed by Kendall; see Fig. 2.

Agent systems analysis and design phases are primarily based on role modeling [2]; thus, an agent's behavior is modeled on the basis of the roles that it plays. During the analysis stage, relevant role models are identified by focusing on the agents and their interactions. During the design stage, roles that a given agent needs to play are refined and composed.

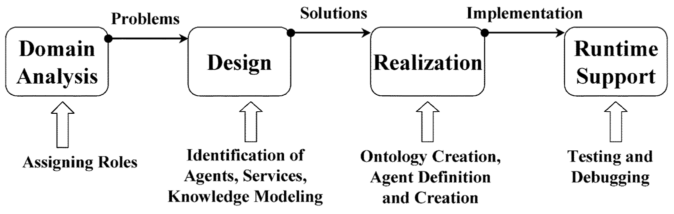


Fig. 2. Agent-based application development methodology [9].

The realization stage of an agent based application combines the creation of the agents through the implementation of role specific solutions identified in the previous stages, with the development of a specific ontology based on the knowledge modeling from the design phase.

Finally, the last phase in the process of building an agent-based application will deal with runtime issues related to deployment: testing and debugging.

In the sequel, we will illustrate the methodology (Fig. 2) on a supply chain scenario from the manufacturing and delivering of telephones and answering machines by modeling the roles corresponding to the specific entities' functions and assigning responsibilities to the agents that will perform these roles. Consider a manufacturing scenario in which company "A" produces telephones and answering machines, subcontracting the production of circuit boards to another company "B." In order to build a single system to automate (certain aspects of) the production process, the internals of both companies "A" and "B" have to be modeled. However, neither company is likely to want to give up information and/or control to a system designer representing the other company.

B. Application Analysis

The objective of the analysis phase is to understand the structure of the system to be developed without worrying about the implementation details and assign roles to agents in a consistent manner, following closely the system's requirements and definition. In our case, the system is viewed as an organization or collection of roles that relate to each other and form an interaction model. Roles in the system are descriptions of business entities, whose functions are modeled at an abstract level.

1) Business Description: Supply chain business processes usually deal with the production and transfer of resources on demand to those that need them.

The business domain entities and relationships are based on the purpose and functions of the supply chain system and are modeled in terms of objects representing user roles, businesses, and services. In our case, it covers four business domains (see Fig. 3).

2) Product Description: Building products like telephones or answering machines involves the acquisition and integration of different components and materials, from various manufacturers and suppliers, thus providing a suitable example of a supply chain.

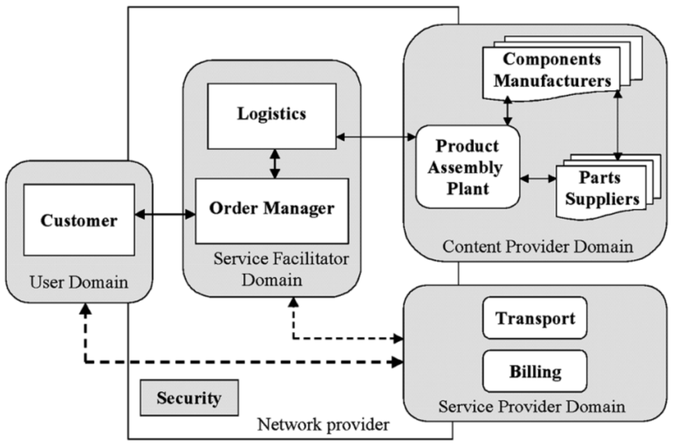


Fig. 3. Business domains and relationships in an agent-based supply chain application.

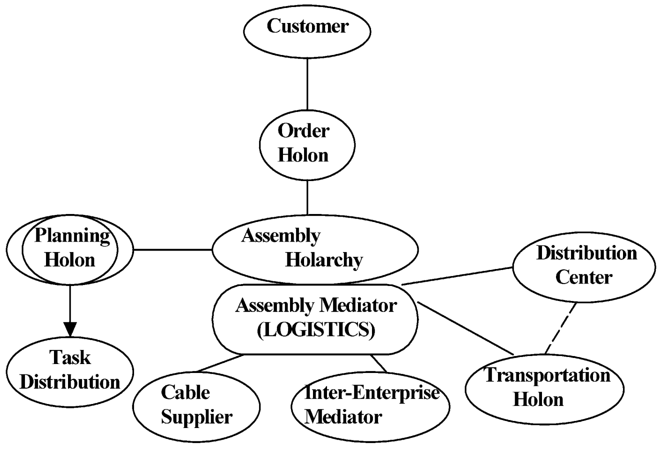


Fig. 4. Order-management holarchy.

We will consider the general case of a product consisting of components provided by different manufacturers and parts or subcomponents provided by suppliers. For the sake of simplicity, we assume the following.

- Telephones consist of a printed circuit board, a molded case, and transmitter-receiver equipment as components and cables as parts or subcomponents.
- Answering machines will be made of a printed circuit board (different than the one used for telephones), a molded case, and power adapter as components and cables as parts.
- Printed circuit boards are manufactured from electronic parts as subcomponents.

3) Order Management Holarchy: At this level, the following entities interact (see Fig. 4):

- *the Customer*—the end user of a product who triggers the production and transfer of resources on demand;
- *the Order Manager*—responsible for acquiring orders and handling customers' requests;
- *the (e)Logistics*—attached to the manufacturer (assembler) of the final product (the Assembly Plant), being responsible for coordinating manufacturers and suppliers and negotiating the production and delivery of needed resources (components, parts);

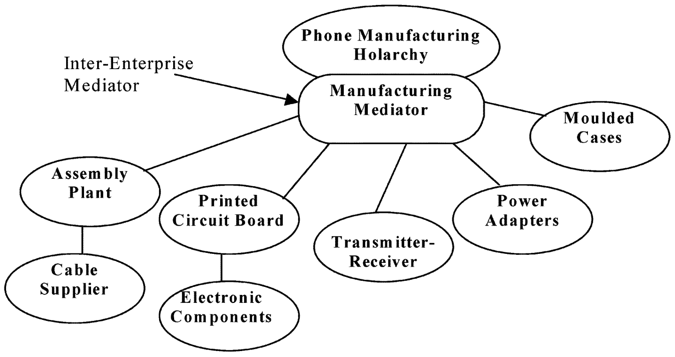


Fig. 5. Manufacturing holarchy.

- *the Planning Holon*—responsible for the coordination of assembly tasks;
- *the Transport Unit*—responsible for the management of transportation resources;
- *the Inter-Enterprise Mediator*—the link with the other holarchies involved in the supply chain (such as the manufacturing holarchy).

4) Manufacturing Holarchy: The participants in the manufacturing holarchy and their capabilities are the following (see Fig. 5):

- *the Assembly Plant (the main participant)*—a telephone and answering machine manufacturer that acquires the necessary components for building these products and assembles them;
- *the Cable Supplier*—direct supplier for the assembly plant.

The collaborators of the assembly plant are

- transmitter receiver plant;
- printed circuit board plant, having one supplier of electronic parts;
- power adapters plant;
- molded cases plant.

Each of the above-described roles is assigned to an agent, as will be detailed later (in Section III). Fig. 6 presents the overall holarchy integrating both the order—management and manufacturing levels. (At the order-management level, another important holon is illustrated: *the Bank*—responsible for transferring funds between accounts upon demand when resources have been transferred as part of a transaction.)

5) System Requirements: The requirements are divided into three categories as follows.

- *Support for automation of ordering:* This system requirement aims to facilitate and automate the process of ordering a product in the supply chain market and delivery of resources on demand among the participants present on the Enterprise level.
- *Support for multiagent negotiation:* Negotiations require a particular protocol and an inference engine to help the agents reach a decision, mainly based on their financial goals. In addition, the negotiated transactions

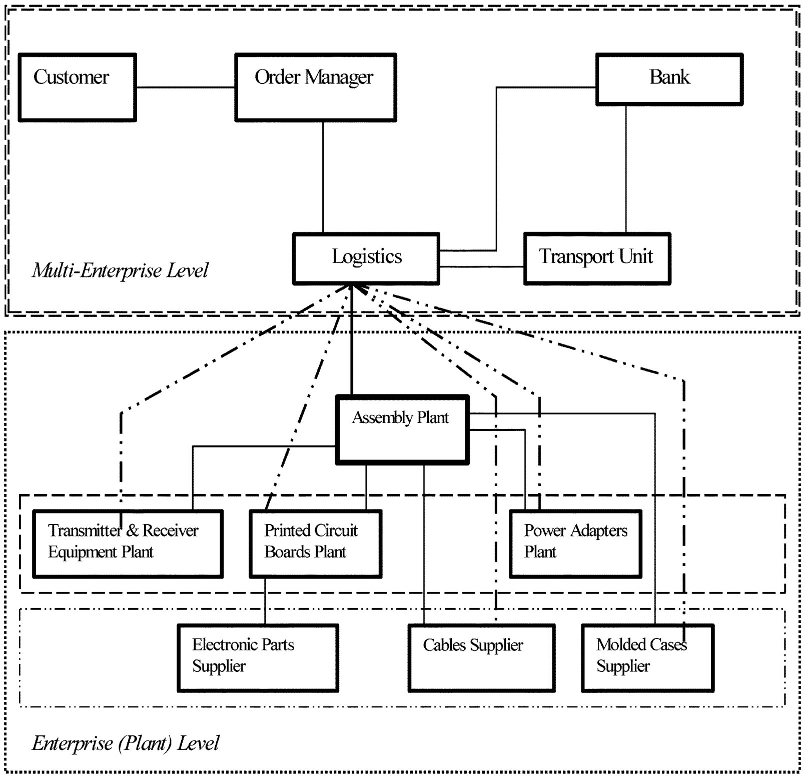


Fig. 6. Supply chain holarchy.

need a certain amount of time to complete, during which a context must be maintained.

- *Support for added services:* The system has to provide a number of added services that may be used in common by the participants in the supply chain such as 1) directory facilitator, which is a registry of service names that provides matchmaking between providers of services and those who request them, 2) transportation facilities shared by manufacturers, suppliers, and assemblers, and 3) bank facilities accessed by customers, logistics, and plants in the process of closing a transaction.

6) Supply Chain Agents Role Modeling: Having defined the entities involved in the overall holarchy (see Fig. 6) and established the roles and their interactions within the supply chain application, we can create a network of agents based on the responsibilities that come from these roles and the resources that need to be produced or consumed. Thus, we identify and design several types of agents (see Fig. 7).

- *Plant Agents that play the roles of the assembler and manufacturers and are capable of performing application specific tasks:* For example, the Molded Case Plant Agent will perform two operations (molding or casting and finishing) as part of a process of molding cases. The specific process pertinent to a plant will be embedded in an agent behavior called at the right time during the protocol interactions. The Assembly Plant Agent does not encapsulate all of negotiation, production, and supply. The Logistics Agent handles the orders and the negotiations. Although the Logistics entity is on the order-management level, it will be acting as part and on behalf of the Assembly Plant (which is also part of the manufacturing level, thus bridging the two holarchies). The other plant agents will have their own handling of production, supply, and negotiation.

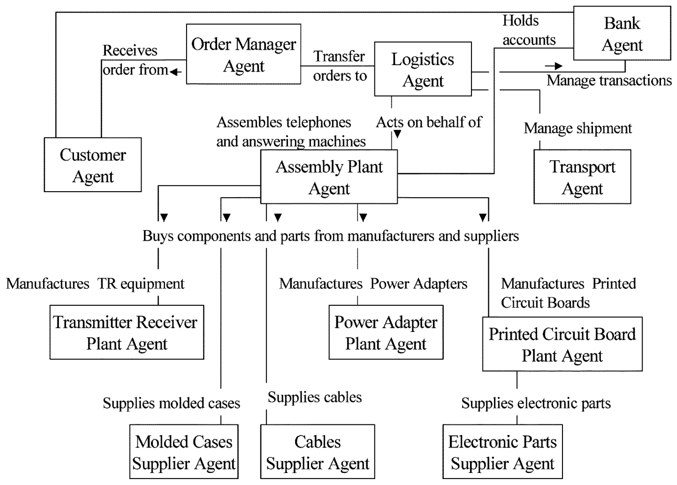

Holds accounts
Bank Agent
Receives order from
Order Manager Agent
Transfer orders to
Logistics Agent
Manage transactions
Assembles telephones and answering machines
Acts on behalf of
Manage shipment
Customer Agent
Assembly Plant Agent
Transport Agent
Buys components and parts from manufacturers and suppliers
Manufactures TR equipment
Manufactures Power Adapters
Manufactures Printed Circuit Boards
Transmitter Receiver Plant Agent
Power Adapter Plant Agent
Printed Circuit Board Plant Agent
Supplies molded cases
Supplies cables
Supplies electronic parts
Molded Cases Supplier Agent
Cables Supplier Agent
Electronic Parts Supplier Agent

Fig. 7. Conceptual model of supply chain agents.

- *The Transport Agent* is responsible for the allocation and scheduling of transportation resources required by the Logistics Agents.
- *The Customer Agent* is enabled with an interface, acts on behalf of the customer, and is the one that demands a product, thus starting the production and transfer of resources.

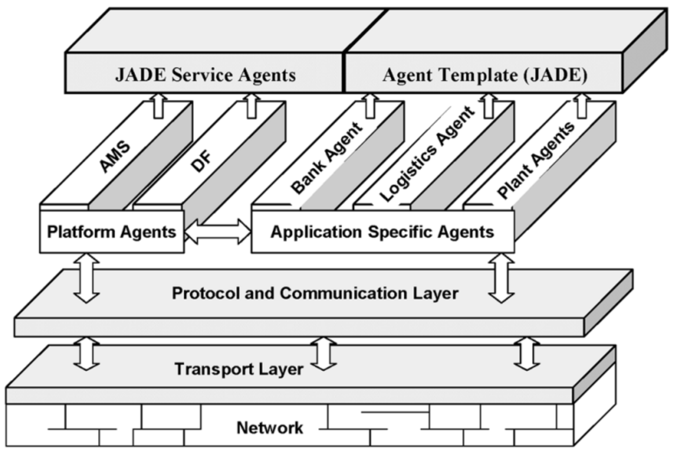


Fig. 8. Functional perspective of system architecture.

- *The Order Manager Agent* is responsible for handling requests from customers, approving orders, and getting information regarding the orders. Then, the information captured would be transferred to the Logistics Agents, which are responsible for coordinating plants and suppliers and negotiating the production and delivery of resources across the supply chain network. This agent is also responsible for locating the service providers by looking up the directory facilitator for services that match a request.
- *The Bank Agent* is responsible for transferring money between accounts when a transaction is completed.

C. Agency Design

The agency design process is divided into five steps: 1) presentation of the system software architecture; 2) agent definitions and responsibilities; 3) knowledge modeling; 4) the structure of an agent class, and 5) sequence class diagram.

1) System Software Architecture: To build our SCM system, we will extend the generic FIPA-compliant agent framework Java Agent Development Environment (JADE) [7] by implementing application specific (functional) agents, which are specialized to perform the activities related to the supply chain.

Fig. 8 illustrates the layers of interaction between agents. The top layer provides assistance for a certain subset of design issues, for example, handling communication, registration, deregistration, search, etc., and consists of generic agents from which functional agents can be derived. This high-level layer provides tools for an abstracted view of the multiagent system, allowing us to concentrate on the behavior level. The next layer is made of two sets of agents: the platform agents responsible for platform registration, directory service registration, search, etc., and the application agents, which is derived from the core agent of JADE. The third layer ensures that the communications (interactions) between agents follow the specified protocols and are handled in conformity with FIPA standards [6]. The bottom layer enables the interagent and agent to environment communications by the means of a network.

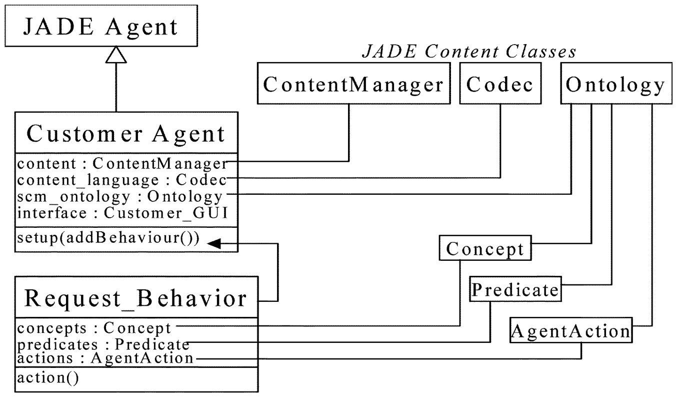


Fig. 9. Class structure for customer agent.

2) Agent Definitions and Responsibilities: Based on their role models, agents can be defined and modeled. The agents' descriptions, protocols, and responsibilities can be summarized as illustrated for the logistics agent [5]:

Role Schema: (Logistics Agent)

Description: Acts on behalf of the Assembly Plant, manages shipping of products, and takes care of financial transactions.

Protocols: FIPA REQUEST protocol, FIPA CONTRACT NET protocol, and nested FIPA ITERATED CONTRACT NET protocol.

Responsibilities: To register with the directory facilitator as a provider of services (products), to maintain information about products, to be aware of the Assembly Plant capabilities, to initiate negotiation on the supply of a resource, to negotiate the terms of supplying a specified resource, to ask about transportation facilities, to request shipping of products, and to request financial transactions.

3) Knowledge Modeling: To model the knowledge that will be used by the agent roles results in the identification of the following elements inherent to the application (see Fig. 9):

Concepts: Expressions that indicate entities that "exist" in the world and those which agents talk and reason about. They are referenced inside predicates or agent actions.

Predicates: Expressions that state the ability of agents in the form of a Boolean variable.

Agent Actions: Expressions that indicate actions that can be performed by some agents.

To indicate the type of variables inside each concept, predicate, or agent action, low-level expressions named primitives (strings and integers) are used.

4) Agent Class Structure: Fig. 9 shows the minimum structure of the Customer agent class that extends the core agent of JADE, thus inheriting all the functionalities that it needs to setup, register, shut down, communicate, and so on. The attributes are instances of JADE content classes, which help manipulate the content of a message composed in a content language with reference to the specific ontology. The setup() method from the bottom field retrieves the content language and adds a previously created behavior to the agent.

5) Message Sequence Diagram: Fig. 10 presents a sequence diagram where agents communicate through interaction protocols. The FIPA Iterated Contract Net protocol [6] is used to handle the negotiations between the Assembly Plant and other

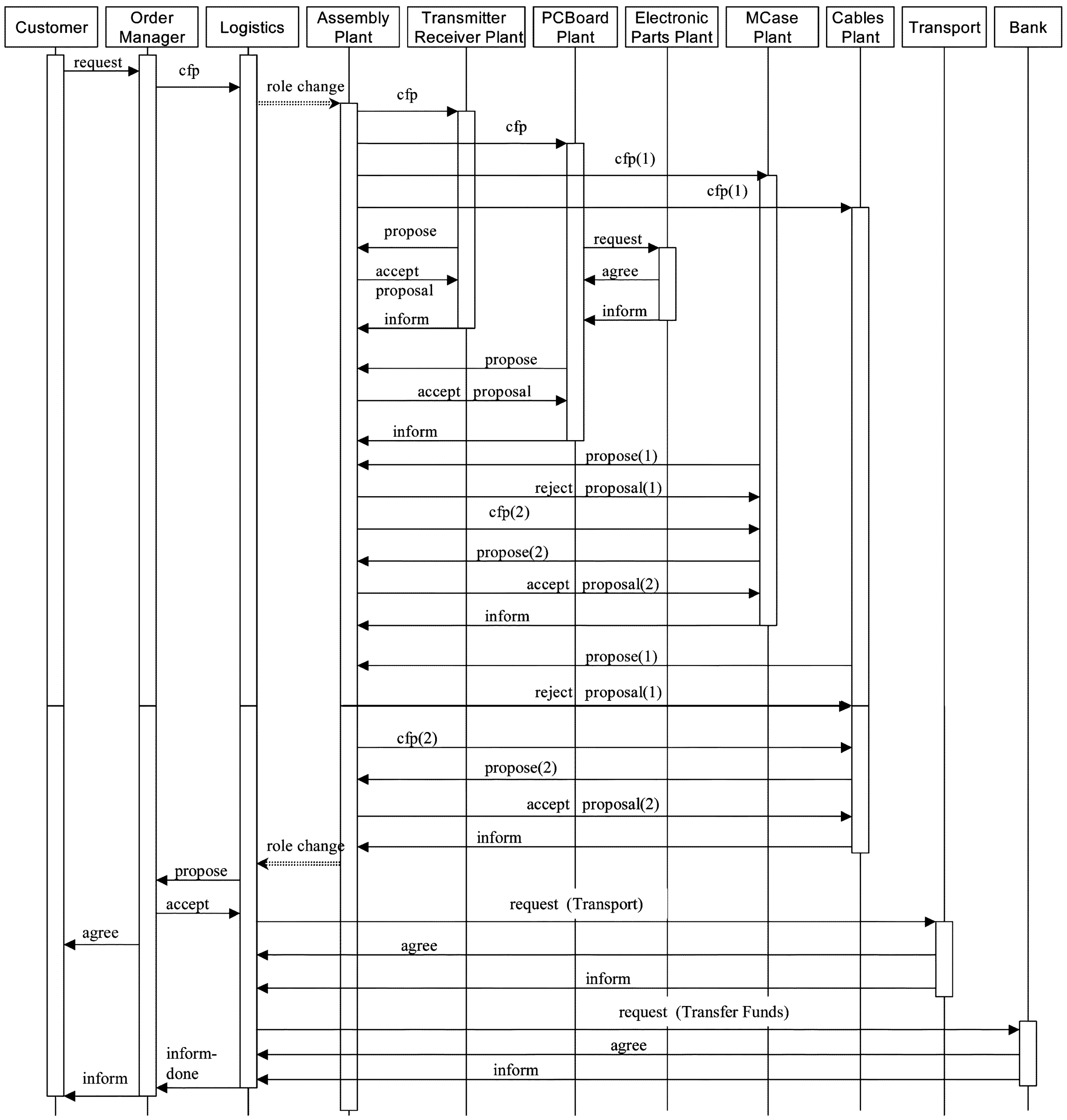


Fig. 10. Sequence diagram depicting interactions among SCM agents.

plants for acquiring the needed resources. (Cfp stands for Call for Proposals).

D. Application Realization

The third stage of the agent development methodology shows how the Supply Chain application is actually implemented and consists of the following activities:

— creation of the application ontology;
— building the inference engine;
— creation of agents.

1) Ontology Creation: Our scenario considers the manufacturing and delivery of products made of components and parts. An ontology for SCM is a set of schemas defining the structure of the predicates, agent actions, and concepts (their names and slots), all related to the process of ordering, building a telephone or an answering machine, and transportation. The entities (concepts, agent actions, and predicates) and their associated attributes, which are an important part of the conversations between agents, are presented in Tables I–III. The physical instances (see Table I) will be inherited from the concept (see Table II). The capabilities of a plant will be inherited from predicates, and the acts of selling or delivering a product will be inherited from agent actions (see Table III). The supply chain ontology is defined in a way that will ensure that the interaction model between agents points to the same concepts or actions in a given context.

Proper Java classes for all types of predicates, agent actions, and concepts in the ontology need to be developed. At this point in coding the ontology, it has to be mentioned that a class structure and relations among classes in an ontology are different

TABLE I
INSTANCES OF PRODUCTS

Telephones					
Slots (Attributes)	Description	Type	Presence	Cardinality	Restricted Values
tel_type	The telephone type (cordless, desk)	String	Mandatory	single	
tel_brand_name	The telephone brand (Sony, Bell)	String	Optional	single	
callerID	Is there a callerID incorporated?	Boolean	Optional		true, false
Answering_Machines					
Slots (Attributes)	Description	Type	Presence	Cardinality	Restricted Values
am_type	The answering machine type (tape, digital)	String	Mandatory	single	
am_brand_name	The answering machine brand (Bell, GE)	String	Optional	single	

TABLE II
"ORDER" CONCEPT

Order					
Slots (Attributes)	Description	Type	Presence	Cardinality	Restricted Values
priority	Priority of the order	String	Optional	single	"HIGH" "LOW"
ordered_products	A list of ordered products	List of Products	Mandatory	multiple	
destination	The destination of the order	String	Mandatory	single	
issueDate	The date when the order was placed	Date	Mandatory	single	

TABLE III
"DELIVER" ACTION

Deliver					
Slots (Attributes)	Description	Type	Presence	Cardinality	Restricted Values
order	The order to be delivered	Order	Mandatory	single	
customer	Who places the order	AgentID	Mandatory	single	

from the structure of a similar domain in an object-oriented program because decisions are made based on the structural properties of a class rather than its operational properties. In practical terms, this involves the following steps.

- Define classes in the ontology, and arrange them in a hierarchy.
- Define slots, and describe their allowed values.
- Define individual instances of these classes.

Fig. 11 presents the relationships between classes and instances for our supply chain example.

In Fig. 11, classes are represented as white boxes and instances as gray boxes. Solid arrows represent internal links such as instance of class or implementation of interface, and dotted arrows pointing to the "Supplies" class represent slots. The access methods are set() and get() functions for the defined variables.

2) Rule-Based Engine: The rule-based engine presented in Fig. 12 is written in Java and consists of several classes as follows:

The RuleBase class stores the list of rules and the instance of a single goal. The Rule class contains arrays of Conclusions and Conditions as well as methods for converting these to strings. The Condition and Conclusion classes have as variables the names and values of the corresponding elements in the Negotiation.xml file (the CONDITION and CONCLUSION tags), plus methods for converting these elements to strings.

In Fig. 12, the Goal class has a method for converting the goal based on the variable (price) corresponding to the one found in the Conclusion ("lower price"). The PriceN class in this implementation is constructed with the fixed variable "price," which corresponds to the "ChildText" of the element ⟨VARIABLE⟩ of CONDITION. The other variable found in this class is the choice corresponding to the ⟨VALUE⟩ in CONDITION, which is a variable that will be the actual input into the engine. A data buffer is used to write this value from which the engine can read it and then process it by calling the resolveGoal() method. If we choose to have the variable "price" of class PriceN as input as well, all we have to do is modify the xml file to accommodate this change. This way, we will have different conditions depending on this variables, and the rules will look like this: IF $(\text{var} = \text{molded\_case\_price})$ AND $(\text{price} = 3.0)$ THEN $(\text{value} = 2.5)$,

The Negotiation.xml file contains the if–then rules that are written in XML, which is ideally suited for sharing them across applications. Fig. 12 illustrates the association between the elements in the file and the classes that were briefly presented above. The elements and attributes present in the Negotiation.xml file are retrieved using JDOM, which is an application programming interface (API) for manipulating XML documents.

EqualsPredicate and OfferCommand classes are used to set the command ("Offer" in the xml file) and the predicate ("Equals" in the xml file) when the engine is instantiated. When it comes to embedding the inference engine into the agent, we input our choice (value), set the command and the predicate, read the xml file, and call the resolveGoal() method that will give as the output based on the rules found.

Due to its order of magnitude, we dedicate a separate section to the Agent Creation in the sequel.

IV. AGENT CREATION

In this section, we present the most important aspects of creating an agent based on the behavioral model of JADE and the use of the interaction protocol templates provided as a package

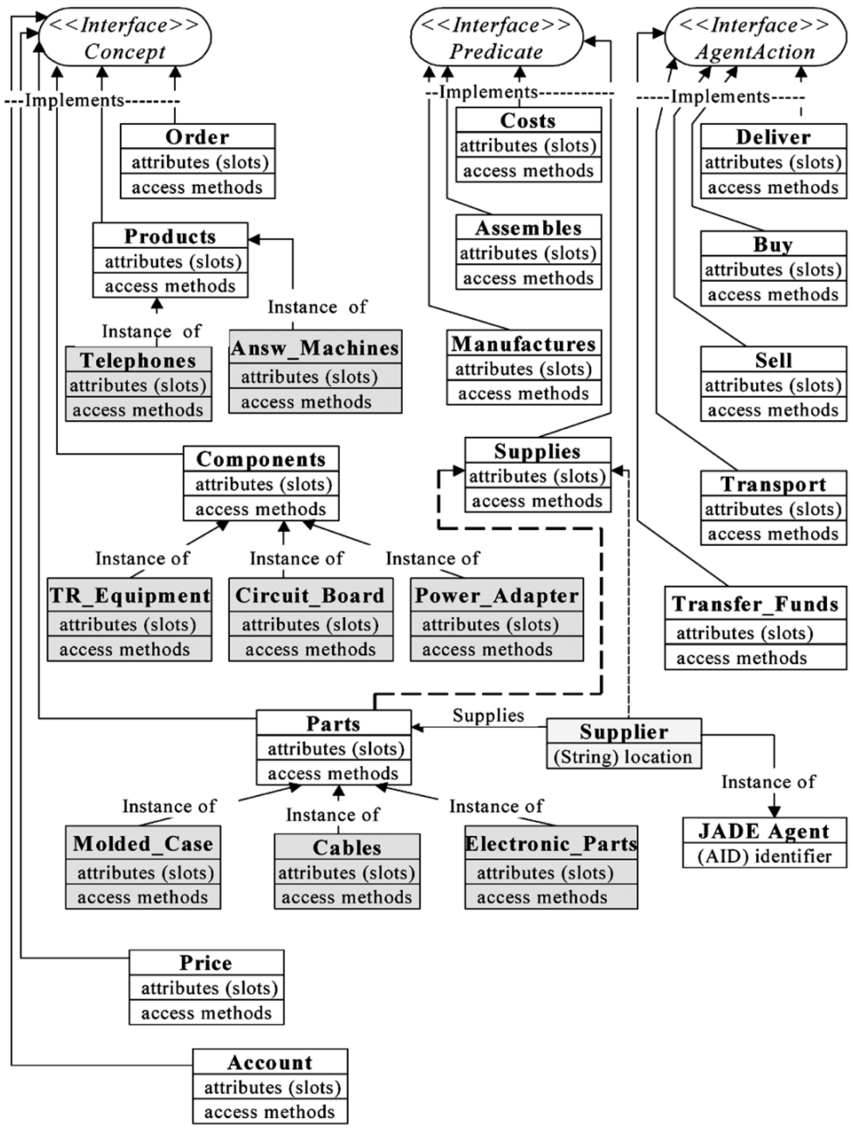


Fig. 11. Dependency relationships in the supply chain domain ontology.

to help manage the conversation states among agents in an efficient manner.

A. Behaviors in JADE

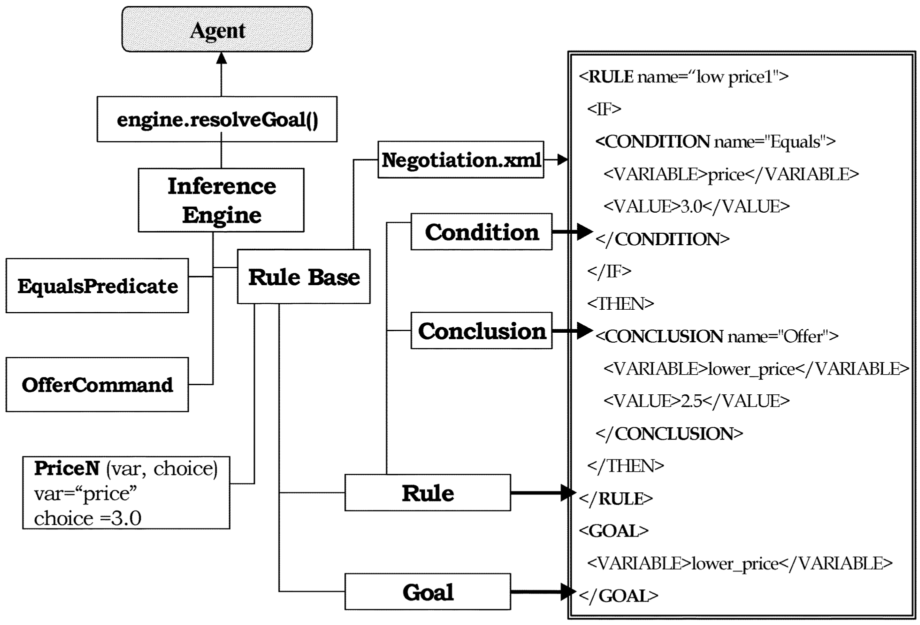
Agent implementation in JADE is based on the behavior model. An agent in JADE is composed of a single execution thread and its tasks are implemented as "Behavior" objects. Therefore, to implement an agent specific task, we have to define one or more Behavior subclasses, instantiate them, and add the behavior objects to the agent task list. Fig. 13 presents the class hierarchy for JADE behaviors.

The behavior classes that inherit from the generic Behavior class are named after the kinds of tasks that they model. For example, the CompositeBehavior class models a complex task, the SimpleBehavior class models a simple task that is not composed of subtasks, the FSMBehavior class models tasks that are composed of subtasks performed in the states of a Finite State Machine, and so on.

B. Interaction Protocols in JADE

To construct the agents' conversations, we can use a set of standard templates for the FIPA interaction protocols. For each conversation between two agents, we have two corresponding roles: that of the Initiator, which is the agent that starts the conversation, and that of the Responder, which is the agent that responds to a message as part of a conversation that will take place. As an example, the ContractNetBehavior class implements FIPA contract net interaction protocol as a template for an agent responder to a call for proposals (cfp) message.

A couple of other templates (AchieveREInitiator/Responder) make it easy to implement inside the agents' behavior simple protocols such as FIPA-Request, FIPA-Query, FIPA—Propose, etc. The instance of AchieveREInitiator is constructed by passing as argument of its constructor the message used to initiate the protocol (e.g., request), and the instance of AchieveREResponder is constructed by passing as argument of its constructor a message template.



```
<RULE name="low price1">
 <IF>
  <CONDITION name="Equals">
   <VARIABLE>price</VARIABLE>
   <VALUE>3.0</VALUE>
  </CONDITION>
 </IF>
 <THEN>
  <CONCLUSION name="Offer">
   <VARIABLE>lower_price</VARIABLE>
   <VALUE>2.5</VALUE>
  </CONCLUSION>
 </THEN>
</RULE>
<GOAL>
 <VARIABLE>lower_price</VARIABLE>
</GOAL>
```

Fig. 12. Rule-based engine (XML).

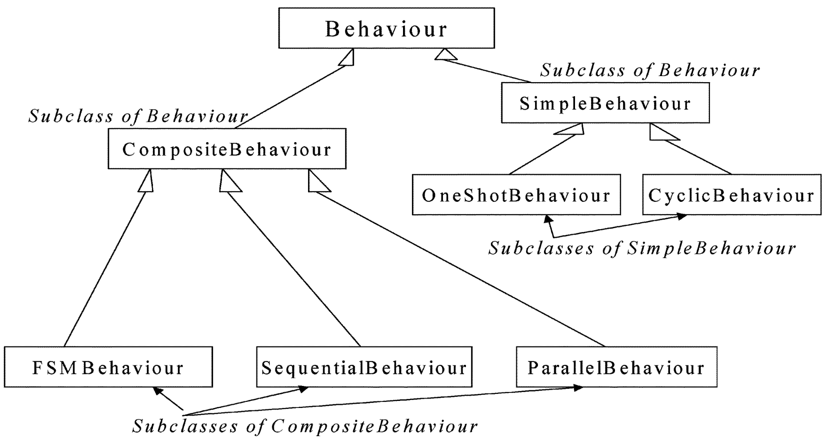


Fig. 13. Class hierarchy for JADE behaviors.

The following examples show how FIPA-Request Initiator and Responder Behaviors are added to a couple of interacting agents:

```
The Initiator—The Customer Agent
public class CustomerAgent extends Agent {
private boolean finished = false;
protected void setup() {
   try {
      // create the agent description of itself
      DFAgentDescription dfd = new DFAgentDescription();
      dfd.setName(getAID());
      DFService.register(this, dfd);
      }
   catch (Exception e) {e.printStackTrace();}
                     }
public void sendMessage(){
System.out.println("Sending request . . .");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
request.setProtocol(FIPAProtocolNames.FIPA_REQUEST);
request.addReceiver(new AID("OrderManager", AID.ISLOCALNAME));
finished = true;
      Behavior order_request = new MyRequest(this, request);
      addBehavior(order_request);}
   class MyRequest extends AchieveREInitiator
         {
         public MyRequest (Agent a, ACLMessage request) {
           super(a, request);}
         protected void handleAgree(ACLMessage agree) {
                 System.out.println("Agree received from OrderManager");
         }
protected void handleInform(ACLMessage inform) {
System.out.println("Received the inform message:" +inform);
                 }
         }// end of inner class MyRequest
}
```

```
The Responder—The OrderManager Agent
public class OrderManager extends Agent {
```

```
public void setup(){Behavior requestB = new MyRequestResponder(this,
AchieveREResponder.createMessageTemplate(FIPAProtocolNames.FIPA_REQUEST));
addBehavior(requestB);}
------------------------------------------------------------------------
class MyRequestResponder extends AchieveREResponder {
public MyRequestResponder(Agent a, MessageTemplate mt){super(a, mt);}
protected ACLMessage prepareResponse(ACLMessage request) throws RefuseException,
NotUnderstoodException {
ACLMessage response = request.createReply();
response.setPerformative(ACLMessage.AGREE);
System.out.println("Manager has sent the agree message:" +response);
return response;}
protected ACLMessage prepareResultNotification(ACLMessage request, ACLMessage re-
sponse) throws FailureException {
ACLMessage informDone = request.createReply();
informDone.setPerformative(ACLMessage.INFORM);
System.out.println(getLocalName() + "has sent the message:" +inform);
return informDone;}}
```

C. Nesting Protocols

Interaction protocols can be nested by using handlers of the states of the protocols that will register other protocols or application-specific behaviors. Interaction protocol classes provide methods to handle different states of the protocol. As an example, the method registerPrepareResponse() in FIPA ContractNetResponder protocol allows registration, in the state of sending back a proposal, of another Contract Net Protocol with the role of initiator. When messages from this second protocol have been received, the agent can return and send back the proposal to the first agent that initiated the conversation.

D. Content Passing and Information Sharing

Once we have the ontology classes in place, the content manager, and the content language registered inside the agent, we can fill out the content of a message and send it.

The following example shows the steps taken to send a message with a meaningful content inside. First, we set up the slots for the needed ontology class, then we add an action to the list of content elements (in the case of AgentAction), and finally, we fill the content of the message, which is then sent to the proper agent.

```
protected ACLMessage prepareResponse(ACLMessage request) throws RefuseException,
NotUnderstoodException {
ACLMessage response = request.createReply();
response.setPerformative(ACLMessage.AGREE);
try {
        Deliver deliv = new Deliver();
  // set up the slots for Deliver AgentAction
        deliv.setCustomer(request.getAID())
        deliv.setOrder(theOrder);
        Action actionD = new Action();
        actionD.setActor(request.getAID());
        actionD.setAction(deliv);
        ContentElementList contEl = new ContentElementList();
        contEl.add(actionD);
        response.setLanguage(codec.getName());
        response.setOntology(ontologyscm.getName());
        contentmanager.fillContent(response, contEl);
        }catch(Exception e) {e.printStackTrace();}
return response;}
```

Information can be shared among agents by

A) passing the desired information as a parameter onto the constructor of a registered behavior;

```
Order = new Order();
0rder.setDestination("City");
order.addOrdered_products(products);
---------------------------------------------------
Behavior requestOrder = new MyInform(this, request, order);
addBehavior(requestOrder);
```

B) using a set of available variables (_KEY) that provide the keys to retrieve the information from the DataStore of the behaviors. The DataStore of a behavior is a HashMap of keys with which messages can be paired in order to retrieve and send the information almost anywhere in the system. This facilitates the distribution of information and its penetration deeply into the system where and when it is needed.

```
protected Vector prepareCfps(ACLMessage cfp) {
Vector cfpsVector = new Vector();
DataStore ds = (DataStore)getDataStore();
setDataStore(ds);
ACLMessage info = (ACLMessage) ds.get(requestB.REQUEST_KEY);
System.out.println("We retrieved the message:" +info);
try {
ContentElement ce = manager.extractContent(info);
-----------------------------------------------
} catch(Exception e) {e.printStackTrace();}
v.addElement(cfp);
return cfpsVector;}
```

V. Illustration on a Simple Scenario

We start from the following assumptions.

— The negotiation process is driven by the manufacturers' need for a specific resource.
— When a resource has been produced or received, it is made available to the next entity in the supply chain that needs it.
— Agents know about each other's capabilities through the Directory Facilitator with which they are registered at start up, which makes it easier to add new agents
— To initiate and engage in a transaction dialog, agents will be equipped with appropriate protocols and abilities that influence their dealings with others.

We will illustrate how the system works on a simple scenario (see Figs. 7 and 10).

The customer inputs the order through the order taking form via The Customer Agent's graphical user interface (see Fig. 14). This form captures the demand, which will initiate the supply

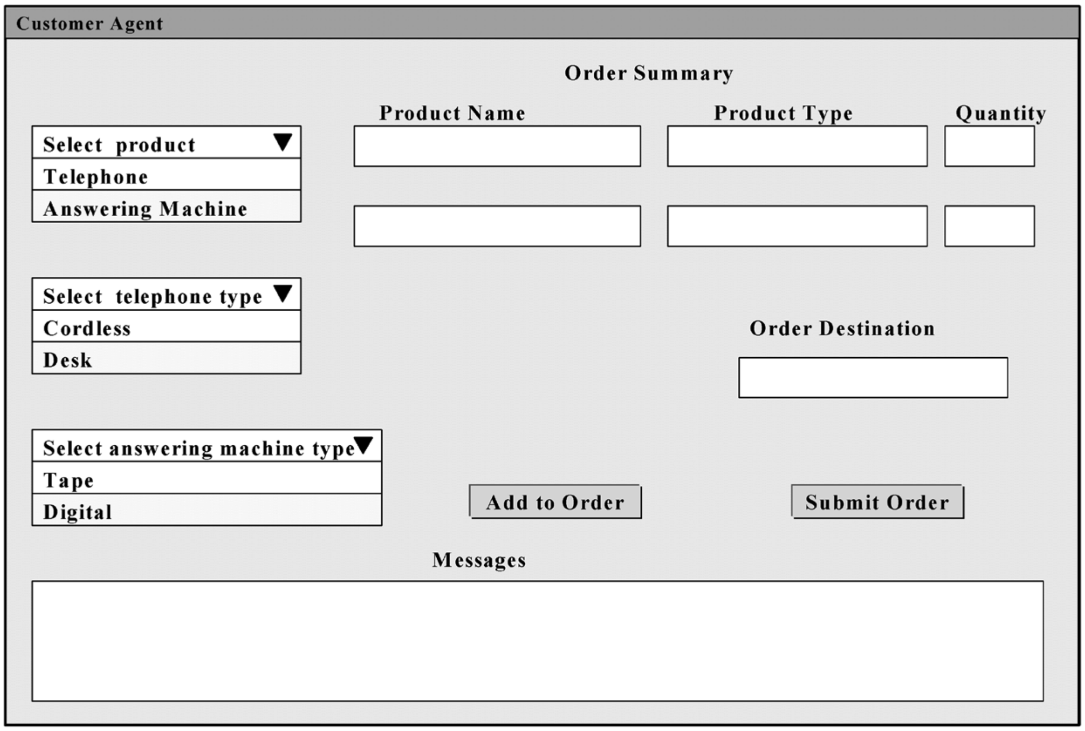


Fig. 14. Customer graphical user interface (GUI).

chain. The message exchange is depicted in Fig. 10. The request (part of an Initiator Fipa-Request-Protocol) will be sent to the Order Manager Agent.

The Order Manager Agent is equipped with a Responder Fipa-Request-Protocol to answer the request and will redirect the order upon arrival to Logistics Agents with a cfp for the ordered products.

The Logistics Agent acts on behalf of the Assembly Plant and is embedded with a Responder Fipa-Contract-Protocol. Upon receiving the cfp from the Order Manager Agent for a certain order, it will start contacting manufacturers and suppliers for components and parts (materials). A negotiation function is used in conjunction with this protocol. The agent that buys components to assemble them later into the final product will have an implementation of FIPA Iterated Contract Net with the role of Initiator and each of the participants (agents) in the negotiation will have their own protocol of FIPA Iterated Contract Net with the corresponding roles of participants or responders. The negotiation takes place in the form of bids and deals with reaching a common price for a certain component. If the negotiation results in a price acceptable to both parties, the rule-based engine of each agent (see Fig. 12) switches to a settlement stage and the required component (resource) will be transferred to the buyer.

A Rule-based engine with a set of specific rules is integrated into the responders (Plant Agents) to determine what actions should be taken at a certain point in the negotiation process. Once a Plant Agent has agreed to supply a particular resource at a certain price, it fulfils its commitment by running the task that produces it.

A similar engine is used by the Initiator of the FIPA Iterated Contract Net (the Logistics Agent), which starts to call for proposals from manufacturers and suppliers and based on its goals, it iterates these proposals until an acceptable offer is made or a certain period of time elapses.

If all resources are sufficient and all negotiations end well, the Logistics Agent will contact the Transport Agent for available routes and prices in order to transport the final product to the Distribution Center and the Bank Agent to request the needed transaction. The Logistics Agent finishes its job by sending its proposal back to the Order Manager Agent, which will decide based on the final price and delivery time whether the order complies with the customer requirements and, if it does, will send an “agree” message to the Customer Agent (as in Fig. 10).

On the other side, if the negotiation at the enterprise level between the Logistics Agent and the manufacturers does not reach a common ground, a “refuse” message will be sent back to the Order Manager Agent, which in turn will inform the Customer Agent of the failure to satisfy its requirements.

VI. Conclusion

As the effectiveness of centralized command and control in SCM starts to be questioned, there is a critical need to organize supply chain systems in a decentralized and outsourced manner. Agent-based models can easily be distributed across a network due to their modular nature. Therefore, the distribution of decision-making and execution capabilities to achieve system decentralization is possible through models of operation with communication among them. In principle, each entity in a supply chain can maintain an agent-based model of its operations and make only a part of these operations (those that are needed) available to its partners. In other words, entities in the supply chain operate subject to their local objectives, but they can incorporate constraints and data derived form the operations of their supply chain partners.

Having selected and employed a framework that provides portability in distributed environments and interoperability enables us to pass data between applications and share functionalities. The ontology structure of the JADE framework is, in our opinion, one of the best designed to address the issues of accessing and sharing information pertinent to a specific application. The combination of behavior features and the ontology structure makes a powerful means of penetrating information deeply into the system. Thus, in our case, information about an order, a product, or a customer can be accessed even at the far end of the supply chain if needed (e.g., a supplier of materials for a supplier that provides parts for a manufacturer of components, which in turn provides these to an assembly plant, can obtain information about an order as long as the order has entered the system at some point). Making the most of this feature can push a customer order deeply into the supply chain, thus providing a customer-centric approach to SCM.

This work can be further improved by

— extension of the ontology to include most of the terms related to supply chain management, thus creating a basis for building product specific ontologies;

— integration of plant scheduling and control. Enabled by the interconnection of the enterprise and intraenterprise levels, it lays the ground for building and implementing a virtual enterprise in which the organization and automation of the manufacturing processes is incorporated within wide supply chain networks and their related management systems.

References

[1] Apprise Company [Online]. Available: http://www.apprise.com/apprisenet.asp

[2] M. Barbuceanu. An architecture for agents with obligations. [Online]. Available: http://www.eil.utoronto.ca/

[3] R. D. Blackwell and K. Blackwell, "The century of the consumer: Converting supply chains into demand chains," *Supply Chain Manage. Rev.*, Fall 1999.

[4] J. H. Christensen, "Holonic manufacturing systems: Initial architecture and standards directions," in *Proc. First Eur. Conf. Holonic Manufacturing Syst.*, 1994.

[5] M. Cobzaru, "Agent-Based Supply Chain Management System," M.Sc. thesis, Elect. Eng. Dept., Univ. Calgary, Calgary, AB, Canada, 2003.

[6] Foundation for Intelligent Physical Agents—FIPA Specification Repository (2002). [Online]. Available: http://www.fipa.org/repository/index.html

[7] Java Agent Development Environment [Online]. Available: http://sharon.cselt.it/projects/jade/

[8] J. D. Edwards Co. [Online]. Available: http://www.jdedwards.com/

[9] E. A. Kendall, "Agent roles and role models: New abstractions for multiagent system analysis and design," in *Proc. Int. Workshop Intelligent Agents Inf. Process Management*, Sep. 1998.

[10] M. Knapik and J. Johnson, *Developing Intelligent Systems for Distributed Systems: Exploring Architecture, Technologies, and Applications*. New York: McGraw-Hill, 1998.

[11] Manugistics' SCM [Online]. Available: http://www.manugistics.com/solutions/scm.asp

[12] P. J. Metz, "Demystifying supply chain management," *Supply Chain Management Review*, Winter 1998.

[13] Oracle [Online]. Available: http://www.oracle.com/applications/B2B/

[14] PeopleSoft [Online]. Available: http://www.peoplesoft.com/corp/en/products/line/scm/

[15] SAP Co. [Online]. Available: http://www.sap.com/

[16] J. N. Sheth, R. S. Sisodia, and A. Sharma, "The antecedents and consequences of customer-centric marketing," *J. Acad. Market Sci.*, 2000.

[17] M. Ulieru, R. Brennan, and S. Walker, "The holonic enterprise—A model for internet-enabled global supply chain and workflow management," *Int. J. Integr. Manuf. Syst.*, vol. 13, no. 8, 2002.

Mihaela Ulieru (M'95–SM'02) received the M.Sc. degree in electrical engineering and computer science from the Electrical Engineering and Computer Science Department, Politehnica University, Bucharest, Romania, in 1985 and the Dr.-Ing. degree in electrical engineering and computer science from the Control Engineering and Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, in 1995.

She was a Postdoctoral Fellow in intelligent distributed systems with the Intelligent Robotics and Manufacturing Systems Laboratory, School of Engineering Science, Simon Fraser University, Vancouver, BC, Canada, from 1996 to 1998. Her research targets the design and implementation of adaptive information infrastructures fueling tomorrow's e-Society.

Dr. Ulieru chairs several International R&D initiatives and is on the IEEE AdCom board of the Industrial Electronics Society, in charge of the emerging area of Industrial Informatics. In 2001, she founded the Canadian Global Agents Integration Network (GAIN), which joined the research efforts of 19 Universities and Research Institutes across Canada, working together to develop intelligent web services for collaborative virtual organizations. Her extensive work with the industry led to her significant contributions to the emerging area of Industrial Informatics, which earned her the Chairmanship of the First IEEE International Conference on Industrial Informatics and the prestigious Canada Research Chair Award from the National Science and Engineering Research Council of Canada.

Mircea Cobzaru received the B.Sc. degree in mechanical engineering from the Technical University of Iassy, Iassy, Romania in 1989 with a thesis on "Automated orbital machine for manufacturing valves" and the M.Sc. degree in electrical and computer engineering from the University of Calgary, Calgary, AB, Canada, in 2003, with a thesis on "Agent-based supply chain management systems."

Since 2004, he has been a Technical Support Engineer with Verity, Inc. Calgary, in charge of the Verity search and indexing enterprise applications. From 1997 to 2000, he was Production Manager at Magna Shutters, Calgary, where he coordinated activities of manufacturing, production control, and material planning and provided technical assistance regarding manufacturing and installation. His current research interests are in the areas of software simulation and modeling, intelligent systems, agent-oriented software, robotics, multidisciplinary interests (supply chain, manufacturing, information technology), search engines, information retrieval, document indexing, and classification.