Incorporating Conversation Managers into Multi-agent Systems

Fuhua Lin
Institute for Information Technology
National Research Council of Canada
M-50, Montreal Rd., Ottawa
Canada K1A 0R6
1-613-9932535
fuhua.lin@nrc.ca

Douglas H. Norrie
Department of Mechanical & Manufacturing Engineering
University of Calgary
2500 University Dr., Calgary, AB
Canada T2N 1N4
1-403-2205787
norrie@enme.ucalgary.ca

R.A. Flores, R. Kremer
Dept. of Computer Science
University of Calgary
2500 University Dr., Calgary, AB
Canada T2N 1N4
1-403-2205112
robertof@cpsc.ucalgary.ca
kremer@cpsc.ucalgary.ca

ABSTRACT

This paper presents ongoing research into enhancing high-level communication capability in multi-agent systems through modeling agent conversations and incorporating distributed conversation managers. First, conversations are formulated as task-oriented descriptive units, called conversation schemata, by capturing interaction patterns, task constraints, and task-related information of agents. After verification, conversation schemata are translated into production rule sets and implemented as Java threads. Second, coordination of conversations is enabled by incorporating a set of distributed agents—conversation managers. A 'proof-of-concept' implementation shows the potential of this approach.

KEYWORDS

Software agent, agent communication, agent conversation, schemata

1. INTRODUCTION

Two recent developments in agent technologies—both theoretical and practical—have made it possible to enhance high-level communication capability by incorporating conversation management and concurrency coordination mechanisms into multi-agents systems. First, ACLs (such as KQML [15], FIPA ACL [7]) have provided a tool and framework to tackle the interoperability problems of inter-agent communication [2]. Second, many studies on conversation policies (CPs) have been conducted [10] based on the notion that agent communication can be better modeled and more easily implemented when conversations (rather than isolated messages) are taken as the primary unit of analysis.

Various constraints need to be considered when developing application-level communication modules. Such constrains include task, execution, and semantics constraints. Therefore, it is desirable to build up a union between high-level, task-directed interactions and communicative acts-based implementation models. Due to the concurrent nature of agent conversations, large-scale multi-agent systems need to address issues derived from their concurrency, such as starvation, livelock, and deadlock of agent conversations. It is therefore necessary to incorporate a conversation management mechanism and infrastructure in which security, concurrency, and scalability are taken into account to prevent systems from these problems.

This paper presents ongoing research towards enhancing agent high-level communication capabilities by modeling agent conversations, and incorporating conversation managers (CMs). First, interaction patterns, task constraints, and task-related information are formulated as conversation schemata. After their verification, conversation schemata are translated into production rule sets. We have implemented these rule sets as Java [12] threads within specialized agents called conversation managers. Conversation manager agents use these rules to mediate and coordinate concurrent conversations among agents. A 'proof-of-concept' example shows the effectiveness of this approach.

The remainder of this paper is organized as follows. Section 2 gives a brief introduction to the background on conversation modeling. Section 3 presents an approach to conversation modeling by describing how to formulate, verify, and implement conversation schemata. Section 4 discusses the structure and functionality of conversation managers; Section 5 describes a 'proof-of-concept' example; and finally, Section 6 concludes with a brief summary of this paper.

2. RESEARCH BACKGROUND

2.1 Point-to-point vs. mediated interaction

Traditionally, agent systems have used Agent Name Servers (ANS) to enable interaction between agents. In the case of Internet-based systems, agents use ANS simply to look up the IP address of another agent and then to use that address to make a socket connection directly to that agent for the purpose of exchanging messages. The problem with this approach is that if the IP address of the other agent ever changes, the first agent will only find out after the next attempt to send a message fails. Likewise, if an agent ever 'crashes' in any way, it is the responsibility of every other agent with whom it was communicating to properly save any failed messages and re-send them later. An improved approach is that of the JATLite Agent Message Router (AMR) [12]. The AMR is a specialized application that receives messages from registered agents, and queues these messages to the file system before routing them to the correct receivers.

2.2 Conversation rules

Agent conversation protocols are often designed through state transition diagrams (STD) called conversation rules [1], which help to abstract the details of the particular agent communication language (ACL) and agent implementation used. These rules

specify how an agent in a given state receives a message of a specified type, does local actions, sends out messages, and switches to one of the possible next states.

2.3 Representing conversations

For design representation of conversations, state transition diagrams (STD), finite-state machines (FSM), and Petri nets [5] have been often used. The choice of Colored Petri Nets (CPN) [14] was guided by its natural support for concurrency, synchronization, and its use of 'colored' tokens carrying information represented by data structures [3]. CPNs arc expressions—symbolizing preconditions and post-conditions—can be used to represent agent interaction in a modular way [4].

3. CONVERSATION SCHEMATA

Unlike conversation rules, conversation schemata are proposed and used to describe task-oriented group conversational behaviors, conversation knowledge and related information (such as synchronization rules, meta-rules, and facts).

Conversation knowledge can be represented by predicate logic, attribute lists, semantic networks, and 'if-then' production rules. One example can be that of the following rules: 'agent A1 has a type TA1'; 'agent A2 has a type TA2'; 'agent A3 has a type TA3'; 'TA1 provides service SER1 and SER2'; 'Agent A1 is in the state #1 right now'; IF 'agent A1 receive a message M1 with type MT1' AND 'agent A2 is ready' AND 'agent A3 is ready', THEN 'Execute conversation schema (TA1, TA2, TA3, Schem#1)'. As this example shows, conversation policies can be organized as well-formed, independent knowledge pieces that can be represented in an appropriate form. The detailed setup and modeling procedures using CPN are described in the next section.

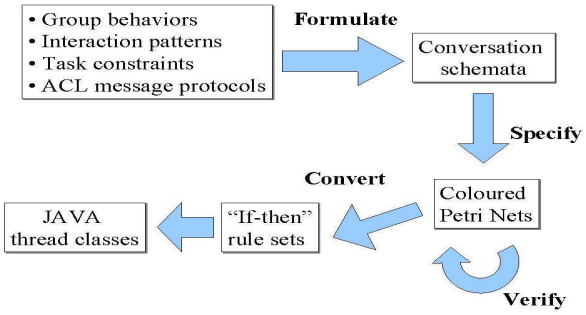


Figure 1. Overview of schema construction approach.

The process of modeling conversations begins with analyses of agent class identification, inter-agent interaction, and conversation topic. The next step is to acquire and construct agent group interaction patterns and task constraints as conversation schemata (schemata, for short). These can be detailed by communicative acts. Schemata are represented using CPN for checking possible inconsistencies, livelocks, and deadlocks. For achieving a desired behavior, schemata are then converted into production rule sets and implemented as executable software components, such as Java threads. Figure 1 gives an overview of the schema construction approach.

3.1 Agent class identification

The first step in conversation modeling is identifying agent classes. We believe that entities in an application context will be identified as agents depending on the granularity of the model concerned. For instance, in advanced manufacturing systems, at the supply chain level, one can organize the supply chain as a network of cooperating agents, each performing one or more supply chain functions, and each coordinating their actions with other agents. These systems commonly include agent types for Logistics, Procurement, Customer service, Financing, Marketing, and Distribution. At the enterprise level, advanced manufacturing systems can be decomposed into business and management, research and development, design, manufacturing, and marketing. At the workshop level, examples of agent types are AGVs, machines, tools, and cell/line controllers.

3.2 Inter-agent interaction analysis

A real-world application often deals with various activities. These activities are conducted through interactions in different ways. For instance, mobile agents have the ability to move by themselves through a network. Agents, however, need either a partial model or partial knowledge of both the task and the environment to know where they should go and what they should do. Inter-agent interaction analysis explores the essence of the relationships within the agent society and with its environment.

3.3 Conversation topic modeling

A conversation typically focuses on one or more 'topics' each associated with task-related information. A topic can be described by a set of variables having values to be agreed upon by the agents involved, as well as constraints to be satisfied by other agents or users.

3.4 Constraint analysis

There are four main types of constraints that should be considered during conversation modeling: security constraints (obtaining information from a system with access control), execution constraints (load balancing), result constraints (determining when the task should end), and task constraints (relations among the several sub-tasks).

3.5 Conversation schemata

We define a conversation schema as a conversational pattern consisting of a set of conversation policies for information exchange among a group of agents centering on a specific topic to accomplish a collective task. Conversation schemata have the following characteristics. A conversation schema consists of a set of sub-schemata. Communicative acts, such as 'inform', 'reply', 'acknowledge', and the like, are the simplest type of schemata and hence we term these atomic schemata.

3.6 CPN representation of conversation schemata

In our approach, the sub-schemata of a schema are represented as transitions of CPNs. Their states are described as places with tokens holding structured messages. Relation flows are represented as pre-conditions and post-conditions in the form of arc expressions.

3.7 Composition of CPN-represented conversation schemata

Compared to other design representations, such as FSM and STD, it is easier to express compositional relationships of components with CPN, since in a CPN a place or a transition of a CPN can represent every input or output of a component. There are three main compositional relations in CPN: sequential, parallel, and mutual exclusion. The composition of two or more schemata at the specification stage corresponds to the merging of two or more corresponding rule sets at the 'if-then' rules construction stage.

3.8 Schemata CPN verification

The derived CPN representation of schemata allows verification for logical consistency, completeness, and absence of deadlock and livelock. There are three methods for verification. The first is based on stepwise refinement and design of well-formed building blocks. The second is through simulation by selecting the area of interest. After specifying an initial marking, which will put one token in each source, the CPN can be simulated for refining the logical consistency of requirements and for assessment of the behaviour of the schemata. The last method is by hierarchically computing a reachability tree [14].

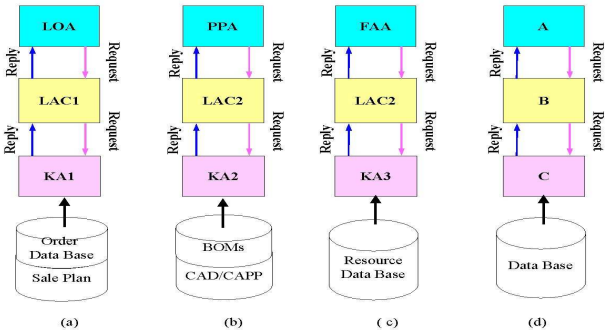


Figure 2. Instances and abstract schema for Information-obtaining.

3.9 Rule-based schema classes

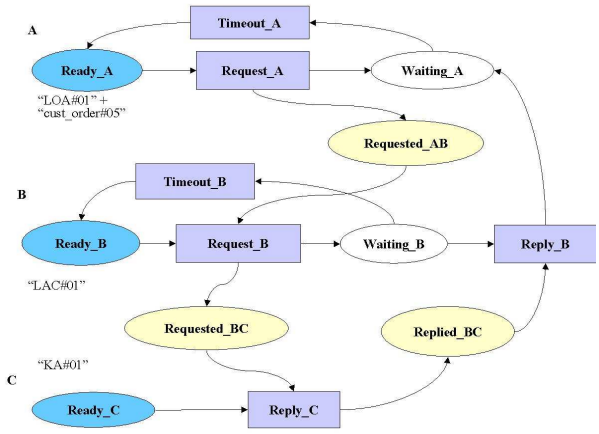
CPN-represented schemata contain three types of knowledge: (1) Descriptive and factual knowledge (tasks, topics, states) represented by colour sets in places; (2) Actions represented by transitions; and, (3) Control knowledge represented by arc functions and flow-controls. If all conditions for a transition are satisfied, the transition can fire, i.e., the conversation can be conducted. In essence, a conversation schema specifies a set of conversation rules, their control mechanism and the local knowledge base maintaining the state of the conversation.

After verification, and for implementation purposes, we can convert each conversation schema to a set of production rules. Each 'place' in a CPN corresponds to the 'condition' part of a rule; every 'transition' in a CPN corresponds to the 'action' part of a rule.

The following shows a simple example of schema.

In an agent-oriented manufacturing system, a Logistics Agent (LOA), after receiving a customer order, needs to find some capable and available Production Planning Agent (PPA) who can commit to this order. The LOA agent can request to a Local Area Coordinator (LAC)—a collaboration agent located near the area where the LOA works—to obtain information about 'who to collaborate with'. The LAC can request to a Knowledge Management Agent (KA) to obtain updated and accurate information about who can collaborate. Reply actions will occur in reverse order. Typically, a PPA receiving a customer order from a logistics agent needs to search for product information from its LAC and then from further KAs. As shown in Figure 2 (a, b, and c), the procedures of information-obtaining performed by a LOA, a PPA, and a factory agent (FAA) are similar. Since the process of information-obtaining is quite widely used, it would be useful to have an abstract conversation schema for that purpose. Figure 2d illustrates an instance of such an abstract schema for abstract agent types A, B, and C.

Figures 3 and 4 show the CPN-represented schemata and derived production rules, respectively, for the schema 'Info-obtaining '.




```
type (color set) definition for "Info-obtaining" schema
      (* order*)
color item = String;
color cust_order = List item;
color mfg_order = List item;
color factory_order = List item;
color order = with cust_order | mfg_order | factory_order
      (* agent *)
color agent_ID = String;
      (* info *)
color product_model = List item;
color feature_model = List item;
color info = with agent_ID | product_model | feature_model
      (* time *)
color time = int;
-----------------------------------------------------
variable declarations for the "Info-obtaining" schema
var agt: agent; var ord: order; var inf: info; var tim: time.
```

Figure 3: CPN representation of schema "Info-obtaining".

The place *Ready_A* has the color *order* and *agent*. The color *order* is defined as a list of items representing the possible contents of the order received. The color *agent* is defined as a string denoting the name of an agent, such as 'LOA#01', 'PPA#05', 'FAA#02'. The colors of the place *Waiting_A* and *Waiting_B*, have the type *order* and *status*. The place *Ready_B*

has the type *agent* and *order*. The place *Ready_C* has the type *agent*. The place *Ready_A* has an initial marking consisting of an agent 'LOA' and an order 'cust_order': 'LOA#1'+ 'cust_order#05'. The place *Ready_B* has an initial marking 'LAC#01'. The place *ready* in C has an initial marking 'KA#01'.

The schema 'info-obtaining' consists of six transitions: *Request_A*, *Request_B*, *Reply_C*, *Reply_B*, *Timeout_A*, and *Timeout_B*. The transition *Request_A* models the replying action taken by the agent A after it receives an order from user/agent. The *Request_B* models the action taken by the agent B after it receives a request from the agent A. The transition *Reply_B* and *Reply_C* are the actions by the agent C and the agent B respectively. *Timeout_A* (*Timeout_B*) is used to model the occurrence of a timeout so that the requests from Agent *A* (Agent *B*) can be sent again. Figure 3 also shows the variable declarations for variables *agt*, *ord*, *tim*, and *info*, and their types *agent*, *order*, *time*, *info*.

```
Rule#001( for Request in A, changing the state of
the schema):
     If (Color(ready_A, agt.) = LOA)
     and (Color(ready_A, ord) = cust_order)
     Then (set_Value(waiting, time) = 0)
     and (set_Value(requested_AB, ord) =
     get_Value(ready_A, ord))
     and (set_Value (requested_AB, agt) =
     get_Value (ready_A, agt))
Rule #002 (for Request in B, sending a message to
LAC)
     If (Color (requested_AB, agt) = LOA)
     and (Color(requested_AB, ord) = cust_order)
     and (Color(ready_B, agt) = LAC)
     Then send_message("request", get_Value
     (requested_AB, agt),
     get_Value(ready_B, agt), cust_order, info)
```

Figure 4. Examples of production rules for the schema "Info_obtaining".

4. CONVERSATION MANAGERS

4.1 Design principles

The current rapid growth in the flexibility, diversity, and complexity of multiagent systems requires a richer conversation management mechanism and communication infrastructure. Within the conversation management paradigm, there is a spectrum of possible designs. At one end is the centralized conversation management. A conversation manager (CM) can tell each agent exactly what to do. In this case the agents participating in the conversation have no role in managing the conversation, they just do what they are told. At the other end of the spectrum, we could have no CM outside the agents, but a kind of (distributed) local one inside each agent. These internal "CMs" would manage the conversation by their interactions. This spectrum thus defines just how much of the conversation management is external to the agents and how much is internal. There are various points along the spectrum (such as a hierarchical approach or a hybrid approach) which indicate the degrees of conversation controllability. Each approach has advantages and disadvantages. Often, one architecture has more applicability in a particular instance.

Similar to agent control, centralized conversation management tends to produce more globally optimal solutions since all agents' behaviours are carefully orchestrated, but may take more time for communication. Distributed management tends to produce solutions faster, and are more locally optimized since agents can act independently without waiting for commands, but it tends to produce less globally optimal solutions since individual agents may not have a good view of the overall situation. Some of the advantages of a centralized conversation manager are:

- The centralised conversation manager approach provides an effective mechanism for the instance where a small redundant number of entities require control activation, because it provides global group state information, therefore reducing the probability that inconsistency or conflicts or deadlocks will arise. It also enhances fault-tolerance, exchangeability, and the possibility of concurrent execution.
- It is important to get each agent to do the most fruitful thing for the overall activity and keep the agents from getting bogged down in communication. Without a centralized view, agents may send redundant or even erroneous messages. Therefore, the time spent in inter-agent reasoning can be saved and this will compensate for the loss of time in message passing.
- Very often, we need a central knowledge base and a central management unit when we divide a whole complex task into several smaller and simpler sub-tasks that are carried out by several agents in parallel and controlled by one agent. This combination of the central knowledge base and management unit is called a Coordinator, or a Mediator [10], or a Supervisor [8]. Under this circumstance, ensuring that all the information is processed in the right way and actions are done at the right time is particularly necessary.

The advantage of centralized approach is the motivation of incorporating conversation managers into multiagent systems. However, the benefits of centralized approach must be weighed against the need to provide scalability. Another obvious disadvantage of the (completely) centralized approach is that it does not allow control data to be processed where and when it is appropriate to do so. In this research, to optimize the performance of CMs in scalability, efficiency, and effectiveness, we propose and are developing a hybrid approach in which a hierarchy of multi-threaded centralized conversation managers is deployed. Each thread of execution is based on a task-oriented well-formulated conversation schema.

4.2 Structure of conversation managers

We developed a 'proof-of-concept' agent system architecture called the Collaborative Agent System Architecture (CASA) [17][9], proposed by the authors, in which the high-level conversational communication is supported by conversation managers.

Agents in the architecture do not communicate with one another in traditional 'point-to-point', 'multicast' or 'broadcast' manner. Instead, agents working together form cooperation areas to ensure system scalability. Each agent in a cooperation area routes all its outgoing messages through a nearest (local) CM, which can direct it to a specific agent (imitating point-to-point communication), to several agents (imitating multicast communication), or to all agents in the cooperation area

(imitating broadcast communication). All incoming messages are received from the CM as well.

Whenever it receives a message from the conversation initiator, the CM agent will create a suitable schema by instantiating a schema class. By executing this schema, the CM keeps track of conversation states, forwards a sender's request to the receiver, and thereafter, relays the receiver's reply to the original sender.

CMs are responsible for managing conversations to ensure their local consistency and coherency. If agents in the local area cannot complete the task, the CM will send a message to a higher CM. The task of a higher-level CM is to assure global consistency and coherency by managing the communications among agents related to different CMs. This creates a dynamic hierarchical stucture of CMs. Under the control and coordination of CMs, agents participating in a conversation in CASA only communicate through their CMs. They cooperate, negotiate, and interact, but they are discharged from resolving communication problems.

As illustrated in Figure 5, a CM has the following main components: a schema class library, an agent naming subsystem (ANS), an active schema pool (ASP), an inference engine (IE), a schema execution engine, and a communications module.

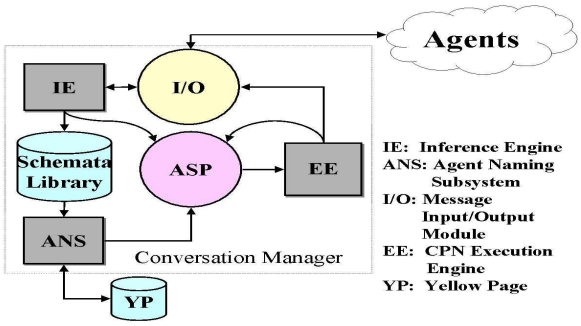


Figure 5. Structure of conversation manager.

4.2.1 Schema class library

The schema class library consists of a set of schema thread classes, which comprise templates to construct schema instances.

4.2.2 Agent naming subsystem

The agent naming subsystem (ANS) is used to realize the process from agent class to agent instance for the agents belonging to a coordination area.

4.2.3 Active schema pool

The active schema pool (ASP) stores all acting schemata in a CM. A particular size of an active schema pool, called the threshold, is set for performance based on certain criteria.

4.2.4 Inference engine

The inference engine (IE) utilizes a load balancing mechanism to allow a message to be forwarded to start a new conversation. The schema-matching process is triggered whenever a CM receives a message from an agent that wants to initiate a conversation. If the IE detects that the size of the ASP has not reached its threshold, the CM selects an appropriate schema class from the schema class library, creates an instance of it and adds it to the ASP. The schema remains in the ASP until reaching an ending state. At this point it can be removed. If the IE senses it has reached its threshold, it balances the CM load over another available CM in which the size of its ASP is less than the threshold set for the ASP. Figure 6 shows the dynamics of schema instantiation and execution.

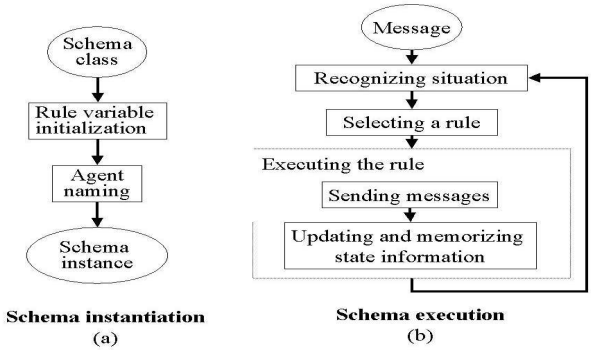


Figure 6. Activity diagram of a conversation schema instantiation and execution.

4.2.5 Schema execution engine

The CPN-based schema execution engine (EE) executes the schema through a 'recognize-match-act' process that involves the following: (1) Keeping track of the current conversation by memorizing the previous messages and historical information accumulated during conversation execution; (2) Analyzing messages, and recognizing the situations, and (3) Sending messages to selected agents about topics related to the current state.

4.2.6 I/O mechanism

The input/output mechanism of a CM includes an object of the class 'in_router' to handle incoming messages, an object of the class 'out_router' to handle outgoing messages, and an object of the class 'input_buffer' containing received messages.

5. A 'PROOF-OF-CONCEPT' IMPLEMENTATION

The conceptual model of a decentralized manufacturing enterprise in this case study consists of the following collaborative agents: Headquarters with a Logistics Agent (LOA), a list of registered Production Planning Agents (PPAs), Factory Agents (FAAs), and Resources Agents (RAs) (for machines and tools). Figures 7 and 8 illustrate two conversation scenarios (agents in one or two areas, respectively, without migration) in which this case study can take place.

Through an Interface Agent (IA), the LOA takes an order 'Cust_Order' from a customer requesting a product: P, with a delivery due date DD, for an amount M as the cost to the customer (cust_cost). The LOA first looks for a suitable PPA from the registered PPAs capable of fulfilling the order. After selecting one PPA, the LOA initiates a conversation with this agent about a production order 'Prod_Order'.

The goal of the PPA will be to balance production according to the capacity of the system and with respect to the manufacturing plan. The required knowledge includes how to react to stochastic events such as demand and supply, as well as how to plan the use of production capacity.

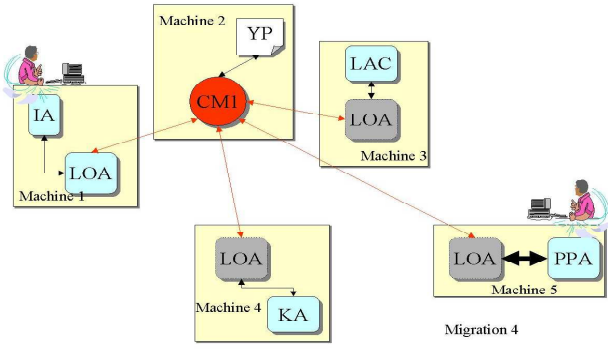


Figure 7. Conversation scenario in one area.

The PPA will first analyze the order using bill of materials (BOM) data, and product design and process design (CAD/CAPP) information to generate a manufacturing order in terms of product features. Based on this information, the PPA finds one or more factory agents (FAAs) able to perform that action by using a protocol similar to the contract-net protocol (CNP) [18]. The FAAs receive the manufacturing order 'Manu_Order', which they decompose into manufacturing features (with corresponding machining operations). Lastly, FAAs make requests to resource agents (RAs) for performing the obtained machining operations, award machining operation tasks to suitable RAs, calculate the corresponding manufacturing cost 'manu_cost', and then send related information back to the PPA.

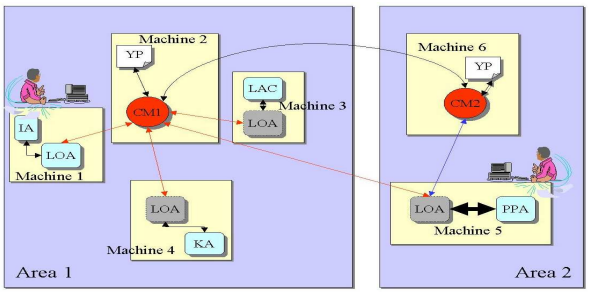


Figure 8. Conversation scenario in two areas.

If there was a 'conflict' between available capacity and required capacity, some managerial actions (e.g., overtime authorizations, adding shifts, increasing/decreasing the number of machine tools or their times in use) can be conducted. These actions are performed in a collaborative manner--via conversations—in order to exchange information. Capacity requirements can be changed by alternate routing, make or buy decisions, subcontracting, raw material changes, inventory changes, and changing due dates arranged with customers. If the available factories are not able to produce the requested parts before the due date DD, the LOA will be responsible for negotiating a new due date with customers, or maybe subcontracting some sub-tasks to other factories outside of the manufacturing enterprise. In the case of this study, we formalized the most common conversation schemata for several key agents and tested them through a Design/CPN simulation [4]. These models were translated into

```
public class INF_OBT_SCHEMA extends Thread {
 //...
 public INF_OBT_SCHEMA(Socket socket, CM.DDN l) {
  super("INF_OBT_SCHEMA"); this.socket = socket;
  this.l =l;}
 public synchronized void run() {
  /* getting connection with agents
 /* situation recognition*/
      INFstate = get_INFstate();
 /* sending messages*/
     ……
   /*Rule for sending a message to LAC for searching PPA
      if     ((requested_AB.state == '1')
        && (requested_AB.agent.color == 'LOA')
        && (requested_AB.order.color == 'cust_order')
        && (ready_B.agent.color) = 'LAC'))
      send_msg('request' + requested_AB.agent +
          ready_B.agent + ready_A.order + info);
       //LOA->LAC
/* processing input message from LOA and */
 /* changing state variables*/
while ((incomMsg = in.readLine()) != null)  {
 /* rule for the schema state initialization*/
  if (INFstate == 1) {ready_A.agent = agent_info;
      ready_A.order = order_info; }
     ……
 /* Rule for setting the value of the state variables*/
 /*in the place requested_AB and the place waiting_A*/
    if ( (ready_A.state == '1')
         && (ready_A.agent.color == 'LOA')
         && (ready_A.order.color == cust_order))
      { set_Value(waiting_A.state, '1');
        set_Value(requested_AB.state, '1');
        set_Value(waiting_A.time), 0; );
        set_Value(requested_AB.order), ready_A.order);
        set_Value(requested_AB.agent), ready_A.agent,) }
       ……
}   out.close(); in.close(); socket.close();   listening = false;
  }    catch (IOException e) { e.printStackTrace(); } }
```

```
// LAC --- Local Area Coordinator
//imports
public class LAC extends Frame {
// ...
public LAC() throws IOException {setTitle("Local
        Coordinator Agent ");
 //.....
 void ToCM_Action(Event event) throws IOException{
   try {lacSocket = new Socket(IP_CM, schema_port_ID);
    // For INF_OBT schema
  out = new PrintWriter(lacSocket.getOutputStream(), true);
  in = new BufferedReader(new
     InputStreamReader(lacSocket.getInputStream())); }
   catch (UnknownHostException e) {System.exit(1); }
   catch (IOException e) {System.exit(1); }
 // ... ...
 while ((fromCM = in.readLine()) !=null) {
 // ……
     if (info.equals("PPA")) {
          KA = look_up(order);  /* search KA for PPA
          out.println("KA#001"+ '……  ');
      else out.println("LAC ready"); }}
// other methods……
```

```
// LOA
//imports
public class LOA extends Frame {
 public LOA() throws IOException   {
  // ……}
 void Order_Taking ();
 void Register_Action();
 public  void Find_Action() throws IOException{
  try {   loaSocket  = new Socket(IP_CM, schema_port_ID);
// getting connection with the CM
// communicate with the CM via a schema
     while ((fromCM = in.readLine()) != null) {
          if (fromCM.equals("LOA"))
              && (fromCM.equals(request_No) {
              send_msg("request" + agent_info +
                        order_info);  }
   } in.close(); out.close(); loaSocket.close();
 }
 // other methods
 }
```

'if-then' rule sets and implemented as Java threads.

Figure 9 shows the class definitions for the information-obtaining conversation schemata used as an example, as well as the rule utilization relations among the socket-based schema class INF_OBT_SHEMA, agent class LAC, and agent class LOA. Figure 10 shows the screen dumps of the due-date negotiation implementation. This example consists of the following seven types of independently running agents: LOA, LAC, KA, PPA, FAA, RA, and CM. For these agents, only the implementation of the CM is presented in this paper.

5.1.1 Conversation manager implementation

The Schema class defines a set of rule variables and rules, along with high-level methods for forward chaining them. Each schema has a name, a list containing the entire rule variable referenced by rules, and a list containing all of the rules. This class contains a method for forward chaining as well as other data structures that are used by the inference algorithms. There is also a Rule class for defining a single rule. This class contains methods that support the conversation management process.

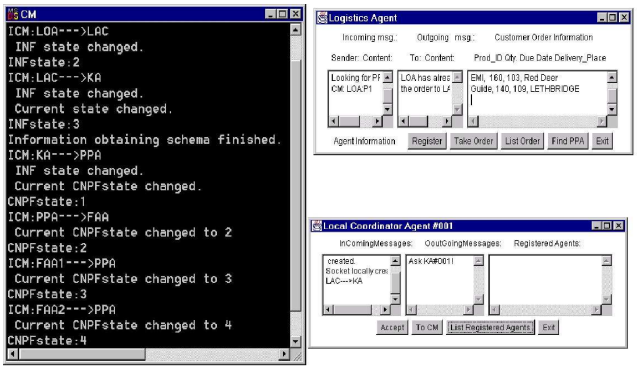


Figure 10: Screen dumps of the due-date negotiation implementation

The current implementation of CMs is based on the TCP/IP protocol, which facilitates the interconnection among remote agents via message communication. Each CM has the ability to create sockets as needed. The messages are wrapped in KQML-like streams. Although socket-based communication allows a great degree of system portability, and communication of dissimilar systems, still each agent needs to implement the same type of messaging language and ontology.

How to negotiate a mutually acceptable due date for a manufacturer and its customers is an important issue in make-to-order manufacturing systems, because sales are normally based on both the cost of the order and its delivery date. The due date bargaining procedure is the following. Based on a customer due date, the LOA looks for an available PPA from a suitable production organization. Next, the LOA negotiates with the PPA about the due date. The PPA calculates the optimal desired due date that minimizes the total weighted penalties for timeliness of all orders, subject to resource constraints. To get the resource constraints, it will request relevant information from a FAA. Usually, the customer due date is not equal to the desired due date. Some customers may insist on their due dates and begin bargaining with the manufacturer. The company's principle may be to let the customer pre-specify a desired due date, but the customer has to pay a tardiness penalty for the resulting displacement of other customers. Note that the LOA can handle multiple customer orders simultaneously, while being in the same or in different internal states.

Now let us examine how conversations merge in this process of due date negotiation. The first conversation is initiated by the customer through an interface agent, which results in an order request being sent to a LOA. This conversation can be carried out with a simple 'order_request' schema. Then, the LOA initiates a 'collaborator-finding' conversation by using the abstract schema 'info_obtaining', which is instantiated with a LAC, a KA, and a topic Topic2. When a suitable PPA is found, the LOA will adopt the same goal as Task2, and then will complete a task, Task3, by starting a new conversation with the PPA using the schema 'Due-date-negotiation' and a topic Topic3.

The contract net protocol can also be constructed as a useful conversation schema, and can be used as the basis for initiating negotiation among agents at the factory and machine level. As an interoperability control protocol, the CNP facilitates the initial bidding process. After initial bidding is established, CMs can use selective broadcasting to instantiate detailed negotiation and solution search actions. Distributed CMs operating at the intra-agent communication level create prioritized tables for agent planning purposes. Final plans are built by aggregating distributed partial plans created at different coordination cluster levels. The CNP is finally reused to inform the sender of the initial request about the final bids in order to award the execution of plans to the best candidate agent. Termination protocols are applied from the bottom up at different negotiation levels. Agents in coordination clusters first decide if they can participate in the negotiation process before committing to the negotiation iterations. There is no fixed number of iterations controlling permanence of individual agents in the clusters.

5.2 'Proof of concept' experimental results

We tested the model in 'Due-date negotiation', with virtual clustering at both factory and machine level, and with 'Capacity negotiation' processes using bargaining procedure and algorithms proposed in [19], within a Java-based programming environment. Using multiple threads with synchronization and scheduling as well as timeouts enables conversations to be efficient and robust. The CMs are implemented as socket-based multi-threaded servers responding to several—perhaps many—concurrent conversing agents. Because networks are not perfectly reliable, data exchanges sometimes fail. The timeout facility helps CMs cope with such real-world problems. Scalability is obtained through loosely coupled interconnection of schemata and partition of coordination areas. We take advantage of CPN and rule-based systems for specification and implementation. The CPN approach can be viewed as being between a pure declarative approach (logic and rules) and a pure procedural one (sequences of events). This is a considerable advantage for conversation managers because agent states, as a kind of information resource, have to be changed in a procedural manner (state sequences) and yet CPs (declarative logic) can also be used. Combining the two approaches proved very efficient.

For this case study, we designed a series of experiments involving *N* agents, to test scalability and emergent behaviour. These agents had to carry out randomly generated customer orders. Each experiment was repeated 20 times separately with 'schema-based' conversations and with traditional 'broadcast-based' conversation strategies which is a situation of naïve communication and each agent broadcasts to every other agent in the system. Figure 11 shows the performance for the two strategies in terms of the average number of communicative actions, called communication transaction, needed per customer order. This number does not include the sub-level partitions (datagrams or packets) that the communication medium needs when transmitting the message streams. In this case, there were two LACs and two CMs. Other more detailed experimental results will be reported in a future paper.

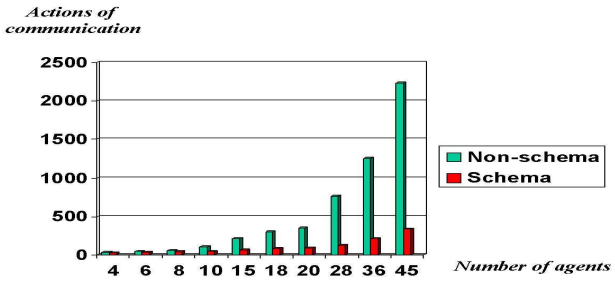


Figure 11: Performance comparison between CM-based and non CM-based broadcast approach.

6. CONCLUSION

Centralized conversation management is necessary for coordinating concurrent conversations and monitoring conversation and ensuring security. This may result in bottlenecks. We have proposed an approach to enhance high-level agent communication capability through modeling conversations based on a component-based construction approach and by deploying conversation managers. Conversation managers provide inter-area coordination. Multi-threaded dynamic conversation schemata governed by conversation managers are used to coordinate agents at inter- and intra-communication levels. to optimize the performance of CMs in scalability, efficiency, and effectiveness, we propose and are developing a hybrid approach in which a hierarchy of multi-threaded centralized conversation managers are deployed. Each thread of execution is based on a task-oriented well-formulated conversation schema.

Initial experimental results show that this goal can be achieved by setting the appropriate parameters (such as the maximum number of threads in a CM, the number of CMs in a cooperation area), and by considering the task complexity, environment, number of shared or competing goals, communication bandwidth, communication reliability, intelligence of individual agents, etc.

Learning mechanisms for conversation managers to further improve conversation behaviours is the subject for future research. Another future research topic is building security and trust mechanisms into schema-based conversation modeling, to realize the level of secure agent communication that is particularly important for electronic commerce applications.

7. ACKNOWLEDGEMENTS

We thank Dr. Mihaela Ulieru, Dr. Deyi Xue, Dr. Weiming Shen, Dr. Xiaokun Zhang and other members in the Intelligent Systems Group (ISG) at the University of Calgary for their assistance. Dr. Fuhua Lin was a researcher with the ISG group on this conversations project.

8. REFERENCES

[1] Barbuceanu, M. & M. S. Fox (1996). The Specification of COOL: A Language for Representation Cooperation Knowledge in MAS. EIL, University of Toronto, Internal report.

[2] Bradshaw, J. M., S. Dutfield, P. Benoit, & J. D. Woolley (1997). KAoS: Toward an industrial-strength open agent architecture. In J.M. Bradshaw (Ed.) Software Agents. AAAI/MIT Press, pp.375-418.

[3] Cost, R.S., Y. Chen, T. Finin, Y. Labrou, Y. Peng, Modelling agent conversation with coloured Petri nets, Issues on agent communication, LNAI, Springer-verlag, 2000 (in press)

[4] Design/CPN: http://www.daimi.au.dk/designCPN/

[5] Ferber, J., Les Système Multi-Agents, InterEditions, 1996

[6] Finin, T, Y. Labrou, and Y. Peng, Mobile agents can benefit from standards efforts on interagent communication. IEEE Communication Magazine, v36 n8, July 1998, pp.50-56

[7] FIPA ACL: http://www.fipa.org/spec/fipa97.html

[8] Fischmeister, S., and W. Lugmayr. The Supervisor-Worker Pattern. Pattern Languages of Programs (PLoP'99), Allerton House, IL, August 15-18, 1999.

[9] Flores, R.A. and Wijngaards, N.J.E. (1999) Primitive Interaction Protocols for Agents in a Dynamic Environment. Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW '99), B.R. Gaines, R.C. Kremer and M. Musen (Eds.), Volume 1.

[10] Gaines, B.R., D.H. Norrie and A.Z. Lapsley (1995). *Mediator: an Intelligent Information System Supporting the Virtual Manufacturing Enterprise*. In Proc. of 1995 IEEE International Conference on Systems, Man and Cybernetics, New York, IEEE, pp. 964-969.

[11] Greaves, M. and F. Dignum (2000), Issues on Agent Communication, LNAI, , Springer -Verlag (in Press)

[12] JATLite Introductory FAQ, http://java.stanford.edu/ProcessLink/papers/JATL.html

[13] JAVA: http://www.javasoft.com/

[14] Jensen, K. (1992). Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 1, Springer-Verlag, New York.

[15] KQML: http://www.cs.umbc.edu/kqml/

[16] Lin, F., Norrie, D.H., Shen, W., and Kremer R., "Schema-based Approach to Specifying Conversation Policies" Issues on Agent Communication, edited by Frank Dignum and Mark Greaves, LNAI, 2000 (in press), Springer-Verlag.

[17] Shen, W., Ulieru, M., Norrie, D.H. and Kremer, R. (1999). "Implementing the Internet Enabled Supply Chain through a Collaborative Agent System", Agents'99 Workshop on Agent Based Decision-Support for Managing the Internet-Enabled Supply-Chain, Seattle, WA.

[18] Smith, R. G. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, IEEE Trans. on Computers, Vol. C-29, No. 12, pp.1104-1113.

[19] Wang D., S-C Fang & T. J. Hodgson, 1998, A fuzzy due-date bargainer for the make-to-order manufacturing systems2, IEEE Trans. on SMC-part C, Vol.28, No.3, pp.492-497