ELSEVIER

Computers in Industry 46 (2001) 259–274

COMPUTERS IN INDUSTRY

www.elsevier.com/locate/compind

Schema-based conversation modeling for agent-oriented manufacturing systems

Fuhua Lin[a,*], Douglas H. Norrie[b,1]

[a]*Center for Computing and Information Systems, Athabasca University, 1 University Drive, Athabasca, Alta., Canada T9S 3A3*
[b]*Department of Mechanical and Manufacturing Engineering, The University of Calgary, 2500 University Dr. NW, Calgary, Alta., Canada T2N 1N4*

Abstract

In agent-oriented manufacturing systems, an efficient and flexible computational conversation model can enable more effective and robust communication, cooperation, and negotiation among agents. This paper therefore describes a schema-based agent conversation model for intelligent agent-oriented manufacturing systems. Conversation schemata are constructed by (1) identifying agent types and their interactions; (2) capturing and formalizing interaction constraints; (3) verifying schemata using colored petri net (CPN) (to prevent deadlock and livelock of conversations); (4) translating the schemata into production rule sets and Java thread classes; (5) identifying inter-schema relationships and building class hierarchies of schemata; (6) constructing conversation managers (CMs) to control and coordinate the schema thread-based dynamic execution. An application example in distributed production planning, implemented in the Java environment, shows the effectiveness of the method. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Intelligent manufacturing system; Intelligent agent; Schema; Colored petri nets

1. Introduction

Two architectures, the holonic manufacturing system (HMS) [1–3] and the agent-oriented manufacturing system [4,5] have been presented in recent years as the basis for an intelligent manufacturing system (IMS). A HMS can be considered to be a particular type of agent society specialized for manufacturing, i.e. having constraints for structure and behavior special to manufacturing [6]. One of the common characteristics of both approaches is that the behavior of the entire manufacturing system is collaboratively determined by many distributed interacting sub-systems that may have their own independent interests, values, and modes of operation.

To facilitate multi-agent collaboration, it is vital that agents effectively exchange task-related information via communication. As agents are usually heterogeneous and running in an open and distributed environment, the main requirement is to ensure reliability and flexibility so that

- there are no possible inconsistencies, deadlocks, and livelocks;[2]
- conversations end with the expected acts and the expected beliefs in the memory of each agent;

* Corresponding author. Tel.: +1-780-675-6175; fax: +1-780-675-6186.
E-mail addresses: oscarl@athabascau.ca (F. Lin), norrie@enme.ucalgary.ca (D.H. Norrie).
[1] Tel.: +1-403-220-5787; fax: +1-403-282-8406.

[2] Here a livelock refers to an infinite execution that produces no useful results.

0166-3615/01/$ – see front matter © 2001 Elsevier Science B.V. All rights reserved.
PII: S0166-3615(01)00133-6

- the inference burden for selection of communicative acts is reduced [7].

A conversation is a sequence of messages involving two or more agents, intended to achieve a particular purpose. Research experience has shown that agent communication can be better modeled and more easily implemented when a conversation rather than an isolated message is taken as the primary unit of analysis [8]. Several workshops on agent conversations are being held as parts of prestigious international conferences. This demonstrates the growing interest in this topic.

This paper proposes a schema[3]-based conversation modeling approach for developing intelligent agent-oriented manufacturing systems. In this approach, interaction patterns and task constraints in manufacturing systems are constructed within class hierarchies of conversation schemata. These patterns and interactions are extracted by identifying agent types and investigating interactions among agents at the different levels of a manufacturing system, as well as by considering the relevant information to be exchanged. Colored petri net (CPN) [9] are then used to verify their correctness and completeness. Next, the schemata are detailed in terms of communicative acts [10–12] and translated into production rule sets and Java thread classes. The dynamic selection, instantiation, and execution processes of conversations are then enabled by incorporating conversation managers (CMs).

The remainder of this paper is organized as follows: Section 2 reviews related work in conversations; Section 3 details the process conversation modeling and introduces the concept of schema; Section 4 describes the construction and verification of conversation schemata; Section 5 discusses the structure of CMs; Section 6 describes an application example; and finally, Section 7 completes the paper with discussion and conclusions.

2. Related work

Numerous agent-oriented approaches to manufacturing enterprise modeling have been proposed within the last decade. Some representative examples are the data flow-based approach [13], the workflow-based methodology [14], the MetaMorph architecture [7], the CIMOSA architecture-based system [15], and the contract net-based approach [16]. Various agent-oriented approaches have been applied to many areas of manufacturing including dynamic scheduling [5,17], enterprise integration [18], concurrent engineering [19], motion planning [20], and production sequencing [21].

One of the most important characteristics of agents is their capability for communication. Based on *speech-acts* [22], some agent communication language (ACL) approaches [23,24] have hoped to emulate unstructured human dialogue. The decision about when to use a protocol, what information to transmit, how to describe the structure of conversation and exchange, and so on, however, has been left to the system designers. An alternative is the plan-based or protocol-based approach [24,25] for agent conversation scenario modeling. These allow agents to have predictable interactions and are effective when used for capturing "goal tree"-based procedure patterns in applications. They lack, however, the flexibility needed to function robustly in an unpredictable environment.

A conversation policy (CP) is defined as a set of specifications for information exchange among agents that facilitates a specified change in the agents' states [26]. CPs are considerably less complicated to implement than the unrestricted agent dialogue model. The CP-based approach is essentially trying to find a sweet spot somewhere between the extremes mentioned above. How to specify, verify, and implement CPs for effective and robust implementations are research questions that still remain open.

In design representations of conversations, state transition diagrams and finite-state machines have been used for specifying conversation protocols due to their clarity [8,27], yet they have limited ability to express the varied interaction-intensive constraints involving multiple entities in conversations. Dooley Graphs have also been used in design representations as an intuitive way to represent sequential relationships of messages during a conversation [28].

Our choice of CPN for representation was guided by its natural support for concurrency, synchronization, and its use of "colored" tokens for carrying

[3] The terminology of conversation schema, schema, and schema-based conversation model will be used synonymously.

information. CPN arc expressions representing preconditions and postconditions can be used to represent agent interaction constraints [29]. Also, object-oriented high-level petri nets can significantly reduce the complexity of model nets and support modular and incremental design [30–32]. Moreover, PN-based tools [33] and compositional modeling techniques [34,35] are available for modeling and simulating agent behaviors.

3. Conversation modeling

In our approach, modeling of conversations begins with agent class identification, inter-agent interaction analysis, and task-related information analysis. The next task is acquiring and grouping interaction patterns with sub-task constraints and then constructing task-oriented conversation schemata. After that, the schemata are represented by CPN for checking correctness and completeness. When they have the desired behavior, they are converted into production rule sets and implemented as software components, such as, Java threads. Through inheritance and aggregation, class hierarchies of schemata can be derived. Fig. 1 gives an overview of this approach.

3.1. Agent class identification

Which entities in an application will be identified as agents depends on the granularity of the model concerned. In an advanced manufacturing system, at the supply chain level, following are usually present: logistics agent (LOA), procurement agent, customer service agent, financing agent, marketing agent, and distribution agent. Each carries out one or more different supply chain tasks. At the enterprise level, there could be design agents, process planning agents, and manufacturing (factory) agents. At the workshop level, there could be transport agents (e.g. AGVs), machine agents, tool agents, and cell/line controllers.

Agents can be classified, according to the services they provide, into the following five categories. (1) Interface agents: agents which are responsible for interacting with physical visual interfaces; (2) collaboration agents: agents involved in activities related to inter-human or inter-agent or human–agent collaboration; (3) knowledge management agent: agents which are responsible for knowledge management; (4) domain agents: agents participating in processes involved in the application (e.g. MRPII, CAD, job scheduling); (5) resource agents (RAs): RAs are related to the management (control) of physical devices and systems associated with applications.

3.2. Inter-agent interaction analysis

A manufacturing-process control and management system often deals with varied activities at different levels of hierarchy. Inter-agent interaction analysis explores the essence of the relationships within the agent society and with its environment. Table 1 shows some examples of inter-agent interactions in an agent-oriented manufacturing enterprise.

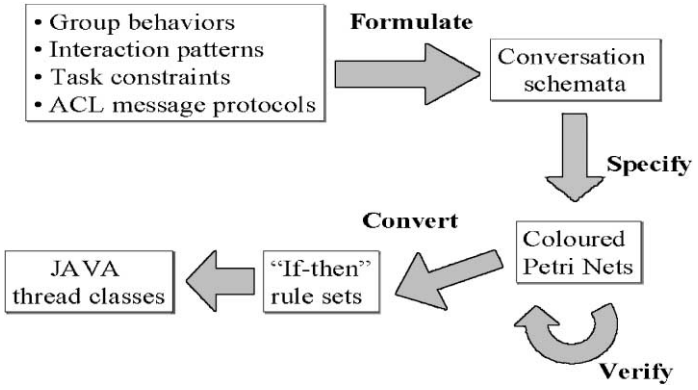


Fig. 1. Steps to construct conversation schemata.

Table 1
Some examples of interactions in an agent-oriented manufacturing system

Interaction	Level of abstraction	Related agent type
Cust_order_agreement	Supply chain	Logistics agent, production planning agent
Product_Process modification	Enterprise	Production planning agent, product design agent
Item-exchange	Cell/line	Cell agent#1, cell agent#2, part agents, transport agents
Part-control-exchange	Workstation	Workstation agent#1, workstation agent#2, tool agents

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 and having constraints to be satisfied by other agents or users.

A conversation topic, denoted by *TP*, can be described by $TP = (TP\_ID; ARGUMENTS)$. Here, *TP_ID* is the identity of a conversation topic and *ARGUMENTS* lists all arguments of the topic. Table 2 shows some examples of conversation topics. For example, the conversation topic customer order *Cust_order* relates to the interaction *Cust_order_agreement* in Table 1. It means 'ship products *Product_ID* to customer *Cust* at location *Loc*, with due-date *DD* and price *Price*'.

3.4. 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.

1. A conversation schema is usually triggered by an individual agent who initializes the conversation to fulfill some specific *task*.
2. A conversation schema is always involved in a series of services provided by a set of agent types $Agent\ Types = \{Agt_1, Agt_2, \ldots, Agt_m\}$ $(m > 0)$. Their instances will participate in the conversation.
3. A conversation schema consists of a set of sub-schemata: $Subs = \{Sub\_Schema_1, Sub\_Schema_2, \ldots, Sub\_Schema_s\}$ $(s \geq 0)$. Communicative acts, such as ''inform'', ''reply'', ''acknowledge'', and the like, are the simplest type of schemata and hence we term these *atomic schemata*.
4. Atomic schemata and sub-schemata change the states of the agents participating in a conversation. These states are denoted as $States = \{Sta_1, Sta_2, \ldots, Sta_l\}$ $(l > 0)$.
5. Precedence relations of the states and sub-schemata can be represented as $Flows = Subs \times States \cup States \times Subs$.

Example 1. In an agent-oriented manufacturing system, a 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 would request a local area coordinator (LAC) near the area where the LOA works to obtain information about ''who to collaborate with''. Then the LAC would request some knowledge agent (KA) to obtain updated and accurate information. Reply actions will occur in reverse order. Typically a PPA, on receiving a customer order from a LOA, needs to search for

Table 2
Some examples of conversational topics in an agent-oriented manufacturing system

Topic	Description (arguments)	Related interactions
Cust_order	*Product_ID*, *Cust*, *Loc*, *DD*, *Price*	*Cust_order_agreement*
Factory_Order	*Product_ID*, supplier, *Loc*, *DD*, manufacturing cost	*Factory_Order_agreement*
Product_Process	Product specification, manufacturing constraints, material cost	*Product_Process* exchange
Item	Items, location, time-stamp	Item-exchange

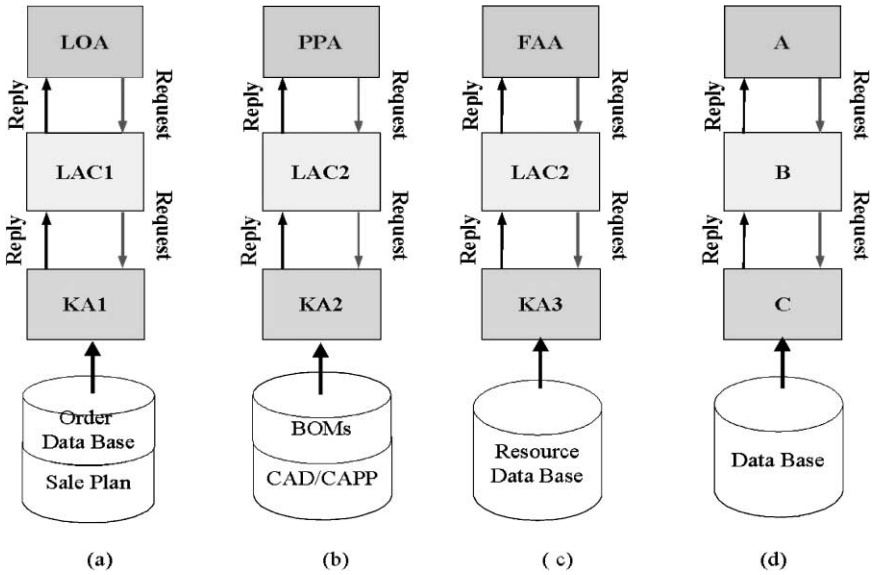


Fig. 2. Collaboration diagram of instances and abstract schema for 'information-obtaining'.

product information from its LAC and then from further KA. The procedure for *information-obtaining* for a factory agent (FAA) is similar. Because it is quite widely used, it is useful to have an abstract conversation schema for *information-obtaining*. Instances of this are shown in Fig. 2(a)–(c) and the abstract schema is shown in Fig. 2(d) for abstract agent types A, B, and C.

The 'information-obtaining' schema can now be described in a textual form as shown in Fig. 3.

Schema

Name: Information-obtaining

Task: looking for a specific information

Topic: information

Agent_types: A, B, C

Acts: Request by A, Request by B, Reply by B, Reply by C,

States: A ready, B ready, C ready, requested, replied,

Timeout for A, Timeout for B

Flow: (to be described in Figure 4)

Fig. 3. Textual description of 'information-obtaining' schema.

4. Representation of conversation schemata

4.1. CPN representation of conversation schemata

In our approach, the *sub-schemata* of a schema are represented as *transitions* of CPN. Its *states* are described by *places* with *tokens* holding *structured messages*. Relation *Flows* are represented as preconditions and postconditions in the forms of *arc expressions*. The following shows the construction process of schemata.

Step 1: Identifying agent types, attributes and state variable of the schema according to the topics.

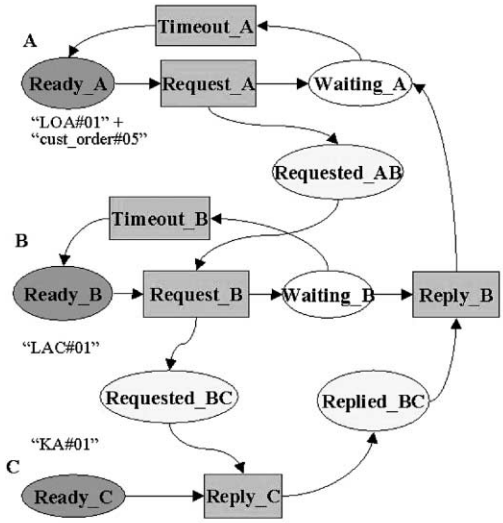
Step 2: For every agent type, adding the transitions for communicative acts or sub-schemata, representing the actions performed by the same agent, which are aligned horizontally.

Step 3: Adding the places and flow expressions between the transitions and connected to them.

Step 4: Adding the information exchanges represented by collective state places that occur among the agents for the topics.

Step 5: Establishment of external interfaces.

Example 2. The CPN representation of the schema for a simplified schema class "Info-obtaining" for




```
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.
```

Fig. 4. CPN-represented schema class ''Info-obtaining'' (simplified).

Example 1. Fig. 4 shows the CPN and colors and variables used in the ''Info-obtaining'' schema.

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.

The variable declarations, specifying the variables *agt*, *ord*, *tim*, and *info*, and their types *agent*, *order*, *time*, *info*, are also shown in Fig. 4. As an example of transition firing, the occurrence of the transition *Request_A* from the place *Ready_A* removes an *agent* token with the value ''LOA#01'' and an *order* token with the value ''Cust_order#05''. This action adds token *time* to the output place *requested* between A and B with a value $tim = 0$ and token *order* to the outplace place *Waiting_A* with a value ''Cust_order#05''.

4.2. Composition of conversation schemata CPNs

Compared to other design representations, such as the finite-state machine and the state transition

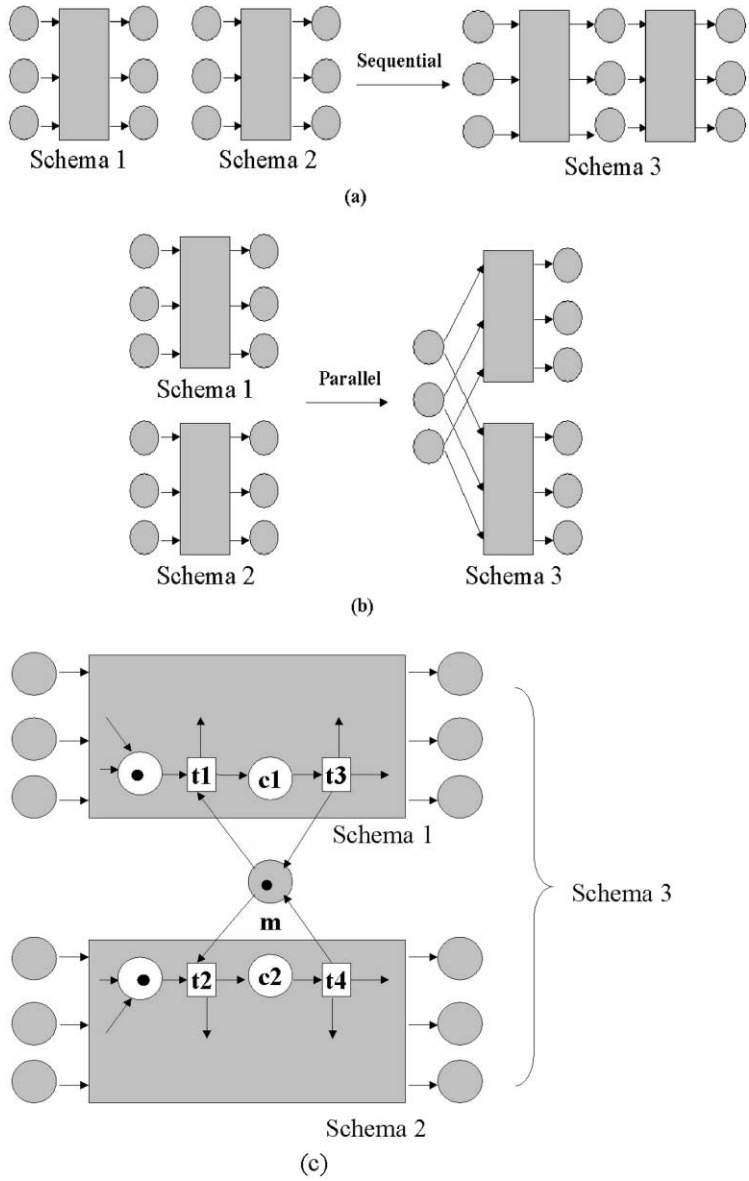


Fig. 5. Composition of conversation schemata.

diagram, it is easier to express compositional relationships of components with the CPN, since every input or output of a component can be represented by a place, or a transition of a CPN. There are three main compositional relations: sequential, parallel, mutual exclusion. The first two are straightforward.

For the mutual exclusion relation, shown in Fig. 5(c), transition t1 and transition t2 are in schema 1 and 2, respectively. Only one of them can be fired at a time as they use a common resource that is denoted as the place c1 in schema 1 and the place c2 in schema 2. To merge these two schemata, we can introduce a shared place *m* and link them as depicted in Fig. 5(c). There is always one and only one token in the place *m*. As schema 1 (or schema 2) intends to enter place c1 (or c2), it must wait until *m* holds a token. Once

schema 1 (or schema 2) fires t1 (or t2), it ''locks'' place p1 (or p2). When the transition t3 (or t4) is finished, it ''unlocks'' the place.

The derived CPN representation of schemata allows verification for logical consistency, completeness, and absence of deadlock and livelock. There are three ways for verification. The first is based on stepwise refinement and designing well-formed building blocks [30]. 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 to assess the behavior of the schemata. The last one is hierarchically computing reachability tree [9].

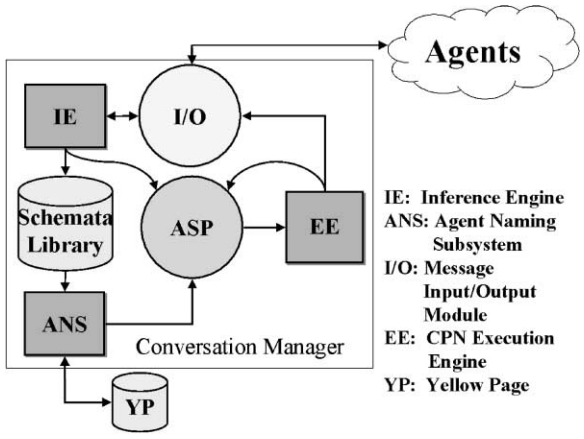


Fig. 6. Structure of conversation manager.

5. Conversation managers

We developed an agent system architecture in which high-level communication is supported by specialized agents called CMs. In this architecture, agents do not communicate with one another in traditional 'point-to-point', multi-cast' or 'broadcast' manner. Instead, a group of agents that are working together form a cooperation area. Each agent in a cooperation area routes all its outgoing messages through a local CM, which can direct it to a specific agent (imitating 'point-to-point' communication), to several agents (imitating 'multi-cast' communication), or to all agents in the cooperation area (imitating 'broadcast' communication). All incoming messages are received from the CM as well.

5.1. Structure of conversation manager

A CM has the following main components: a schema class library, an agent naming sub-system (ANS), an active schema pool (ASP), an inference engine (IE), a schema execution engine, and a communication module (see Fig. 6).

The schema class library *schema_class_lib* of a CM consists of a set of schema thread classes, which comprise the template to construct schema instances. The ANS is used to realize the process from agent class to agent instance for the registered agents stored in a yellow page (YP) of a coordination area. The ASP of a CM stores all acting schemata. A particular size of an ASP, called the threshold, is set for performance based on certain criteria. The IE of a CM utilizes a load balancing mechanism to allow a message to be forwarded to start a new conversation. A schema-matching process is triggered whenever a CM receives a message from an agent who wants to initiate a conversation. If the IE of the CM detects that the size of the ASP has not reached its threshold, the CM selects an appropriate schema class from the *schema_class_lib*; creates an instance of it and adds it to the ASP. The schema remains in the ASP until reaching an ending state, then it can be removed from the ASP. If the IE of the CM senses that 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 of the ASP.

The schema is executed by the *schema execution engine* which simulates CPN execution through a ''recognize-match-act'' process of rule variable value changing and rule utilization (see Fig. 7). Specifically, the following functionality is thus provided for

- keeping track of the current conversation by memorizing previous messages and historical information accumulated during the conversation;
- analyzing messages, and recognizing the relevant situations;sending an instruction to appropriate agents about topics depending on the current state.

The communication module of a CM includes an instance of class *in_router* to handle incoming messages, an object of the class *out_router* to handle

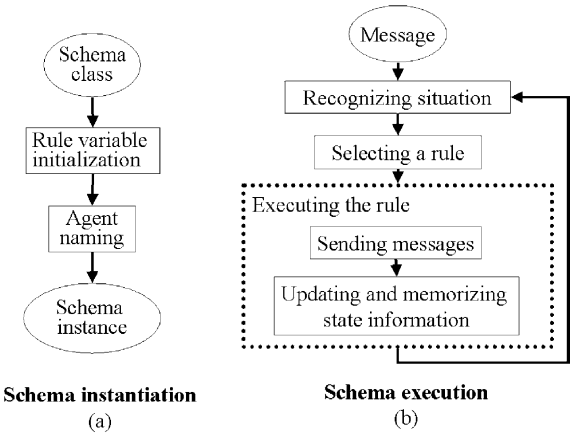


Fig. 7. Activity diagrams of schema instantiation and execution.

outgoing messages, and an object of class *input_buffer* containing received messages.

5.2. Rule-based schema class

CPN-represented schemata contain three types of knowledge: (1) descriptive and factual knowledge (tasks, topics, states) represented by *color sets* in *places*; (2) actions represented by *transitions*; (3) control knowledge represented by *arc functions* and *flow-controls*. If all firing conditions of a transition are satisfied, the transition can fire. This implies that the conversation can be conducted. Therefore, 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, for implementation, we can convert each conversation schema to a set of production rules. Each '$place(s) \rightarrow transition$' of individual agents participating in the conversation in a CPN corresponds to the "condition" part of a rule; every '$transition \rightarrow place(s)$' in a CPN corresponds to the "action" part of a rule.

Example 3. Two examples of production rule representation of the schema "Info-obtaining" is shown in Fig. 8.

A CM is responsible for managing a conversation to ensure its *local* consistency and coherency. If a task cannot be completed by the agents in the local area, the CM will send a message to a higher-level CM that will create a new local CM. *Global* consistency and coherency are the task of a higher-level CM that manages communication among agents related to different CMs (see Fig. 9). Under the control and coordination of the CMs in the Collaborative Agent System Architecture, agents participating in a conversation only communicate directly with their local CMs. Nonetheless, they can still cooperate, negotiate, and interact with agents in remote CMs, since the higher levels of communication and coordination are handled by the higher-level CMs.

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*)

Fig. 8. Examples of production rules for the schema "Info-obtaining".

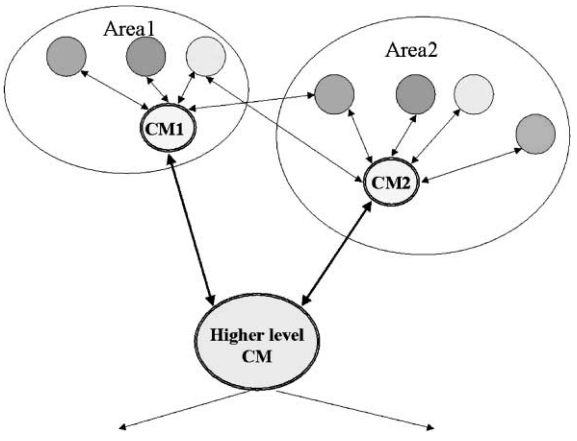


Fig. 9. Conversation managers in different coordination areas.

6. An example

6.1. Agent types

In this case study, the conceptual model of a decentralized manufacturing enterprise has the following seven types of agent: LOA, PPA, FAA, RAs, LAC, KA, and CM.

When an LOA takes an order *Cust_order* from a customer for a product *Product_ID* with delivery due-date *DD*, in an amount *M*, at a cost of *Cust_cost*, the LOA will first look for a PPA from among the registered PPAs that is the most capable to fulfill the order. Then, the LOA will initiate a conversation with the PPA about the relevant production order *Prod_Order*.

The PPA will attempt to realize a certain service level while keeping inventory costs low. The goal of the PPA will be to balance production according to the capacities of the factories and to take into account the overall manufacturing strategy. The PPA will analyze the order using BOM data and CAD/CAPP product specifications and generate a manufacturing order in terms of product features. It will then search for one or more FAA that are able to perform that action, by using a protocol based on the CNP [34]. The FAAs receive the *Manu_Order*, then decompose the *Manu_Order* into manufacturing features (with corresponding machining operations). The FAAs query appropriate RAs about machining operations, plan the award of machining operation tasks to suitable RAs, calculate the corresponding cost of manufacture *manu_cost* and send back the relevant information to the PPA.

If there is a ''conflict'' between available capacity and required capacity, managerial action such as overtime authorization, adding shifts, increasing/decreasing the number of machine tools in use, or subcontracting may be desirable. The necessary information exchanges and actions are performed in a collaborative manner, via conversations between system agents and human agents. Capacity requirements can be changed by alternate routing, make or buy decisions, sub-contracting, raw material changes, inventory changes, and changing customer promised due-dates.

6.2. Due-date negotiation

How to negotiate a mutually acceptable due-date between a manufacturer and its customers is an important issue in make-to-order manufacturing systems [36] because sales are normally based on both the cost and delivery date. The basic process of due-date bargaining is described in the following. Based on the customer-desired due-date, the LOA will first search for a PPA in a suitable production center or organization. Next, the LOA will negotiate with the PPA about the due-date. The PPA will calculate an optimal due-date that minimizes the total weighted earliness and tardiness penalties of all orders, subject to resource constraints. To obtain the resource constraints, it will request relevant information from FAA. Usually, the optimal due-date is not equal to the desired due-date. If it is earlier, then there is no problem, but if later then further negotiation is required. Some customers may insist on their desired due-dates and begin bargaining with the manufacturer. The company's principle may be to let the customer prespecify a desired due-date, and commit to that even though it requires increasing capacity (e.g. by additional shifts) but require the customer to pay a higher price (penalty) for the resulting extras costs. Note the LOA may work on multiple customer orders at the same time, while being in the same or different states. The overall goal of the LOA is 'to maximize the amount of orders from customers, to minimize costs, and to satisfy due-dates for customer orders'. To achieve this goal, the agent makes a plan with three collaboration tasks: *Task1*: receive orders from customers; *Task2*: find most

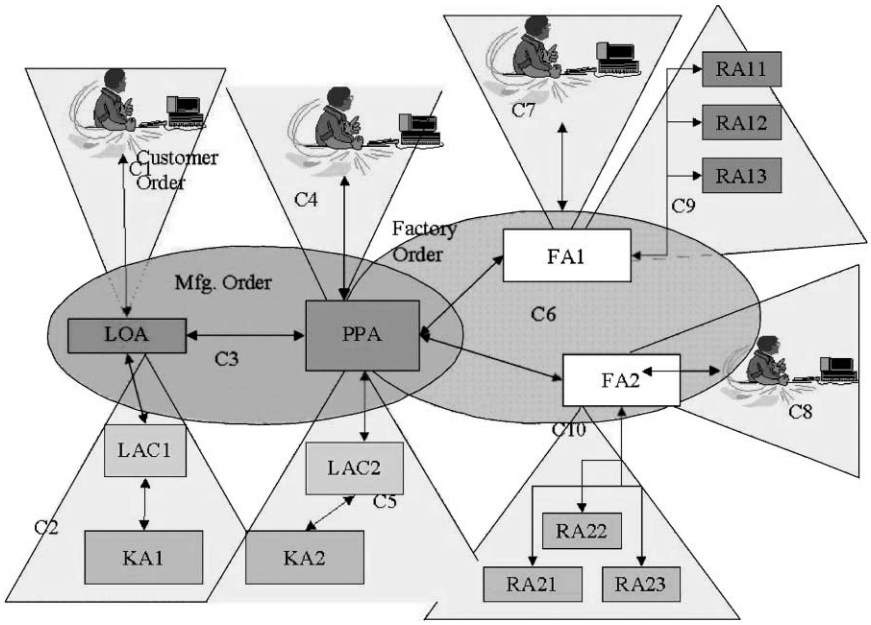


Fig. 10. Conversations in 'due-date negotiation'.

suitable collaborators for production; *Task3*: discuss with the collaborators about due-date and cost.

For *Task1*, the LOA adopts *Sub-goal1* to direct the conversation. Upon receiving the order, it would adopt *Sub-goal2* for *Task2* and *Task3*. Moreover, for each collaboration task, there is a relevant topic. For example, for *Task1*, the topic is *Topic1*: customer order, *Product_ID*, *Cust_ID*, location, *Due_Date*, *Price*; for *Task2*, the topic is *Topic2*: *Collaborator_ID*; for *Task3*, the topic is *Topic3*: Production planning request, *Part_IDs*, *Due_Date*, *Prices*.

Now let us look at how conversations merge in this process of the due-date negotiation. The first conversation *Conv1* will be initiated by the customer through an interface agent, to place an order to a LOA. This conversation can be carried out with the *order_taking* schema, during which the LOA is accomplishing its *Task1*. The LOA then initiates *Conv*2 = 'collaborator-finding' by using the schema *PPA_obtaining* with a LAC and a KA and with the focus being *Topic*, to accomplish *Task2*. When suitable PPAs have been found, the LOA will commence *Task3* by starting a new conversation *Conv3* (with each) using the schema 'due-date negotiation' with the PPA with respect to *Topic3*. Fig. 10 depicts such a scenario of 'due-date negotiation'.

6.3. Schema classes

We first formalized the most common conversation schemata for several key agents and tested them through a design/CPN simulation [9]. We also succeeded in extending and modifying the original schema to eliminate the pitfalls of asynchronous communication, so the resulting schema is 'sound' in the sense that all those possible inconsistencies, deadlocks, and live-locks have been excluded. These models were then translated into "If–then" rule sets and implemented as Java threads. Some of them are abstract classes, which

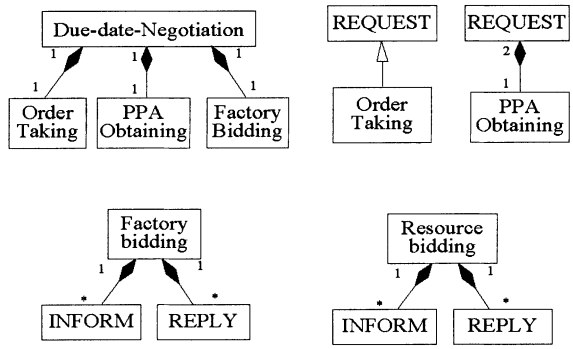


Fig. 11. The class diagram for the example.

can be sub-classed to create new specific application domains. Fig. 11 shows the class diagram for the conversation schemata used in this example. Fig. 12 depicts the rule utilization relations among the socket-based schema class *INF_OBT_SHEMA*, agent class LAC, and agent class LOA (Fig. 13).

6.4. Empirical results and discussion

We tested the model in 'due-date negotiation', with virtual clustering at both factory level and machine level and with 'Capacity negotiation' processes using bargaining procedure and algorithms proposed in [37],

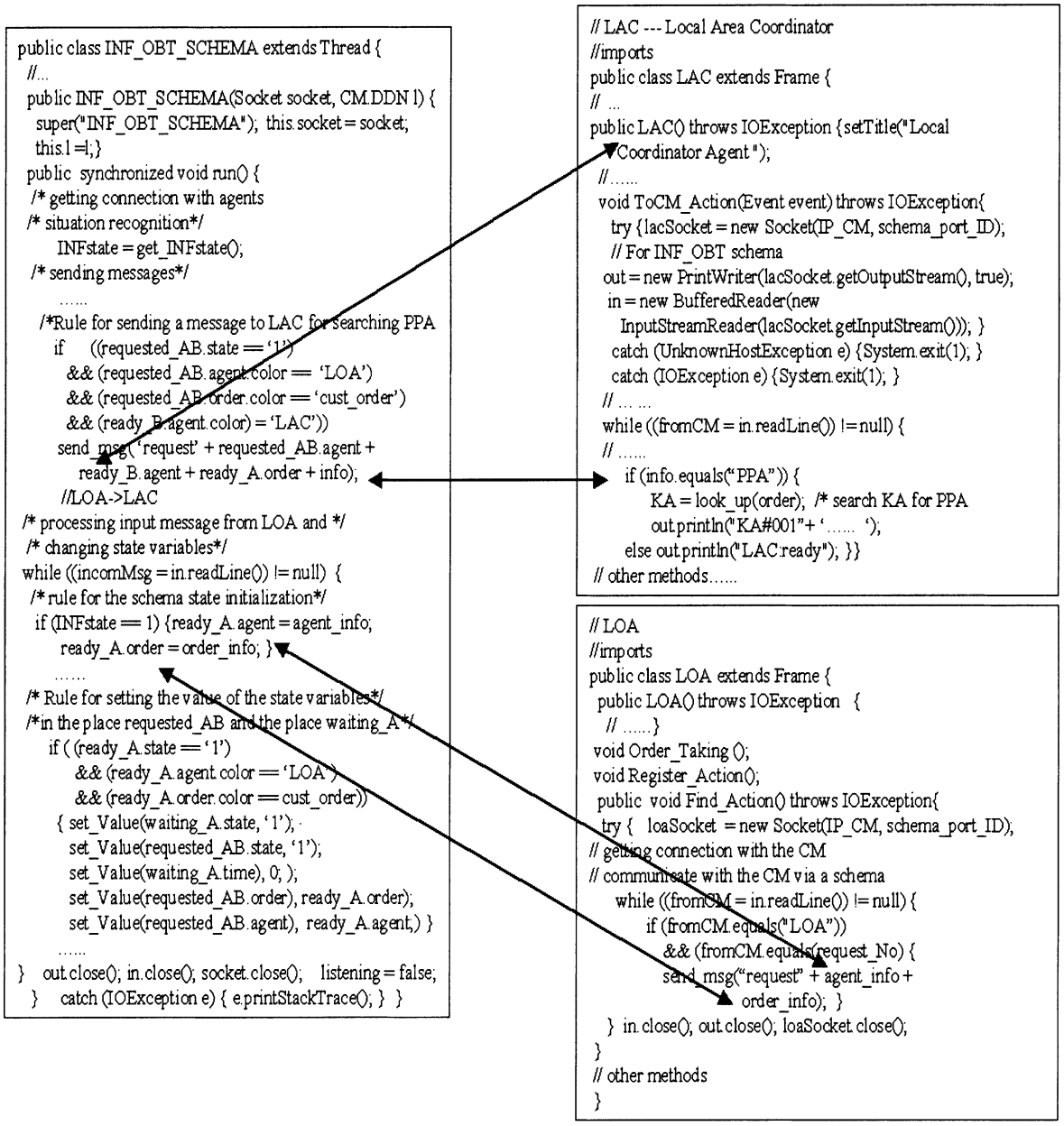


Fig. 12. Rules in Java codes for a schema class and links to two conversing agents.

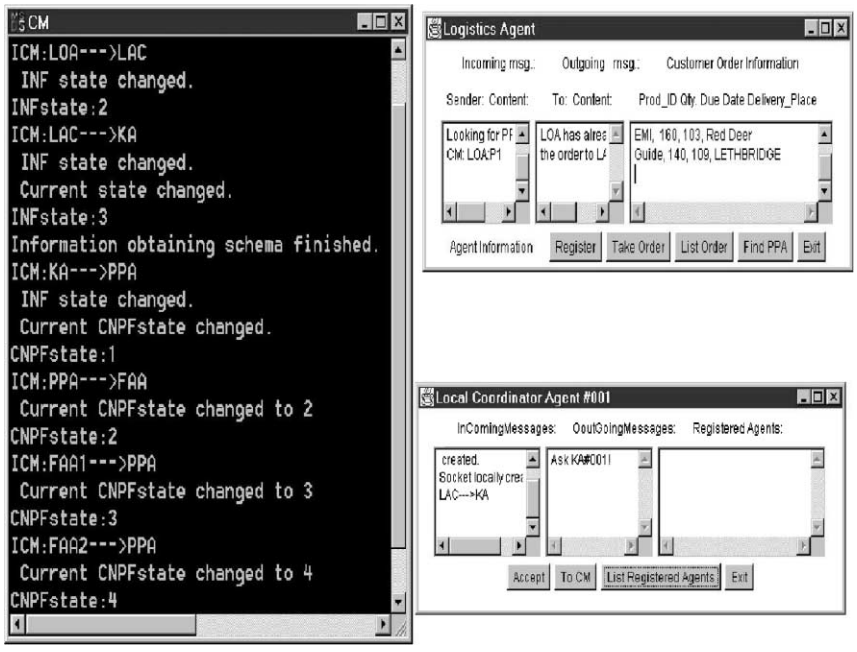


Fig. 13. Screen dumps of the due-date negotiation implementation.

within a Java-based programming environment. Using Java 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 Java socket 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 were able to take advantage of CPN and RBS for specification and implementation, as described. 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 CMs because agent states, as a kind of information resource, have to be changed in a procedural manner (state sequences) and yet CPs (declarative logic) have also to be used. Combining the two approaches is very efficient.

For this case study, we designed a series of experiments involving N agents, to test scalability and emergent behavior. These agents had to carry out randomly generated customer orders. A description of the products' main features is provided in Tables 3 and 4.

Factories A and B have production machines and tools as shown in Tables 5 and 6.

Each experiment was repeated 20 times separately with '*schema-based*' conversations and with traditional '*broadcast-based*' conversation strategies.

Table 3
Description of products' main features

Product name	Primary geometric features				
	Hole	Counter bore	Thread	Pocket	Slot
Bearing cover	5	1	0	1	1
EMI	8	0	8	3	0
Guide	4	0	4	0	1

Table 4
Production orders and results of due-date negotiation

Product name	Quantity	Customer due-date	Initial due-date	After negotiation
Bearing cover	120	100	100	100
EMI	160	103	138	118
Guide	140	109	112	109

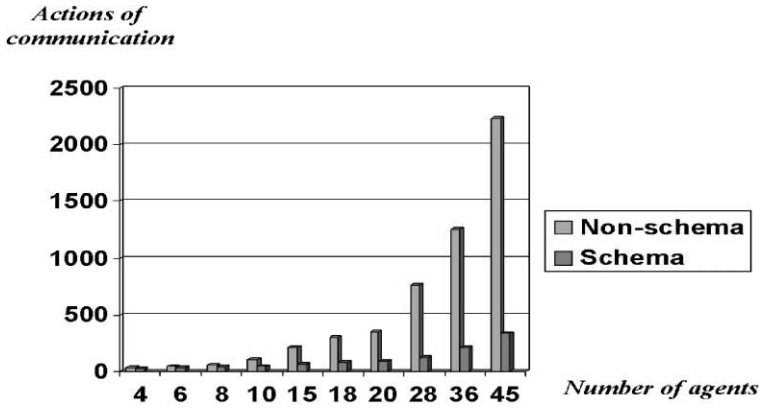


Fig. 14. Performance comparison between CM-based and non-CM-based broadcast approach.

Table 5
Production machines of factories A and B

Factory	Machine name	Machine type	Family
A	VMC1	Mill	Primary
A	VMC2	Mill	Primary
A	VMC3	Mill	Primary
A	Igrind1	Internal grinder	Secondary
A	Igrind2	Internal grinder	Secondary
B	VMCA	Mill	Primary
B	VMCA	Mill	Primary
B	VMCA	Mill	Primary

Table 6
Production tools of factories A and B

Tools in factory A	End-mill, drill, tap, DoveTailMill, ream, counter bore, BallEndMll, wheel
Tools in factory B	End-mill, drill, ream, tap, DoveTailMill, counter bore, BallEndMll

Fig. 14 shows the performance for the two strategies in terms of the average number of communicative actions needed per customer order, when there are two LACs and two CMs. Other more detailed experimental results will be reported in a separate paper.

7. Conclusion

We have proposed a new approach to modeling conversations, for agent communication and cooperation. The concept of conversation schema *links* tasks and task-related information (topics) with low-level communicative acts. This approach has the following distinctive advantages. First, it ensures consistency and effectiveness of agent conversation by considering sub-task constraints. Second, it lowers communication overheads by incorporating CMs that can quickly determine for the participating agents what to do without resorting to lengthy reasoning by them. Third, it reduces complexity of implementation by constructing CMs that separate the description of common agents' functionality from that of communication and synchronization, to ensure local and global coherency. Fourth, it enhances the reusability of software components. Domain-independent and domain-specific conversation knowledge are organized and formulated into hierarchies of conversation schema classes using object-oriented methodologies. Finally, flexibility is realized by incorporating decision-making units and control mechanisms via arc expressions and tokens of nets. The initial markings of CPN representing a schema are used to represent the kinds of situations the agent will likely encounter. Therefore, they provide a possibility for incorporating *learning* mechanisms into conversation schemata so that CMs are capable of learning emergent relationships from past agent interactions, for near optimization and rapid response during task–schema reasoning. Such learning mechanisms are included in proposed future research. Also as a 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 in Web-based manufacturing applications.

References

[1] H.V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, P. Peeters, Reference architecture for holonic manufacturing systems: PROSA, Computers in Industry 37 (1998) 255–274.

[2] P. Valckenaers, H.V. Brussel, F. Bonneville, L. Bongaerts, J. Wyns, IMS Test Case 5: Holonic Manufacturing Systems, IMS'94, Vienna, Austria, 1994, pp. 19–24.

[3] J.H. Christensen, Holonic manufacturing systems: initial architecture and standards directions, in: Proceedings of the 1st European Conference on HMS, 1994.

[4] A. Kwok, D.H. Norrie, Intelligent agent systems for manufacturing applications, Journal of Intelligent Manufacturing 4 (1993) 285–293.

[5] K. Kouiss, H. Pierreval, N. Mebarki, Towards the use of a multi-agent approach to the dynamic scheduling of Flexible Manufacturing Systems, in: Proceedings of the International Conference on Industrial Engineering and Production Management, 1995, pp. 118–26.

[6] L. Rannanjärvi, T. Heikkilä, Software development for holonic manufacturing systems, Computers in Industry 37 (1998) 233–253.

[7] F. Maturana, D.H. Norrie, Multi-agent mediator architecture for distributed manufacturing, Journal of Intelligent Manufacturing 7 (1996) 257–270.

[8] J.M. Bradshaw, S. Dutfield, P. Benoit, J.D. Woolley, KAoS: toward an industrial-strength open agent architecture, in: J.M. Bradshaw (Ed.), Software Agents, AAAI/MIT Press, Cambridge, MA, 1997, pp. 375–418.

[9] K. Jensen, Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 1, Springer, New York, 1992.

[10] Y. Labrou, T. Finin, A Semantics Approach for KQML: A General Purpose Communication Language for Software Agents, Paper Presented at CIKM'94, 1994.

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

[12] D.R. Traum, J.F. Allen, Discourse obligations in dialogue processing, in: Proceedings of the 32th Annual Meeting of the ACL, 1994, pp. 1–8.

[13] M.M. Huntbach, N.R. Jennings, G.A. Ringwood, How agents do it in stream logic programming, in: Proceedings of the 1st International Conference on Multi-Agent Systems, 1995, pp. 177–184.

[14] E.A. Kendall, M.T. Malkoun, C.H. Jiang, The Layered Agent Patterns, The Pattern Languages of Programs (PloP'96), 1996.

[15] R.J. Rabelo, L.M. Camarinha-Matos, From a generic architecture to a particular dynamic scheduling system: the HOLOS methodologies, in: Proceedings of the IEEE/ECLA/IFIP BASYS'95 International Conference on Architectures & Design Methods for Balanced Automatic Systems, Victoria, Brazil, 24–26 July 1995.

[16] J.Y.C. Pan, J.M. Tenenbaum, An intelligent agent framework for enterprise integration, IEEE Transactions on Systems, Man and Cybernetics 21 (6) (1991) 1391–1407.

[17] M.R. Cutkosky, R.S. Engelmore, R.E. Kikes, M.R. Genesereth, T.R. Gruber, W.S. Mark, J.M. Tenenbaum, J.C. Weber, PACT: an experiment in integrating concurrent engineering systems, Computer 26 (1) (1993) 28–37.

[18] H.V.D. Parunak, Manufacturing experience with the contract net, in: M. Huhns (Ed.), DAI, Pitman, London, 1987, pp. 285–310.

[19] L. Overgaard, H.G. Petersen, J.W. Perram, Reactive motion planning: a multi-agent approach, Applied AI 10 (1) (1996) 35–52.

[20] K.T. Chung, C.H. Wu, Dynamical Scheduling with Intelligent Agent, Metra Application Note 105, Metra, Palo Alto, CA, 1997.

[21] M. Wooldridge, S. Bussmann, M. Klosterberg, Production sequencing as negotiation, in: Proceedings of the PAAM'96, 1996, pp. 709–726.

[22] Y. Shoham, Agent-oriented programming, Journal of Artificial Intelligence 60 (1) (1993) 51–94.

[23] M.R. Genesereth, S.P. Ketchpel, Software agents, Communication of the ACM 37 (7) (1994) 49–53.

[24] M. Barbuceanu, M.S. Fox, The Specification of COOL: A Language for Representation Cooperation Knowledge in MAS, EIL, University of Toronto, Internal Report, 1996.

[25] A. Haddadi, Communication and Cooperation in Agent Systems: A Pragmatic Theory, Lecture Notes on Artificial Intelligence 1056, Springer, Berlin, 1995.

[26] F. Lin, D.H. Norrie, W. Shen, R. Kremer, Schema-Based Approach to Specifying Conversation Policies, Issues on Agent Communications, Lecture Notes on Artificial Intelligence, Vol. 1916, Springer, Berlin, 2000.

[27] F. von Martial, Coordinating Plans of Autonomous Agents, LNAI 610, Springer, Berlin, 1992.

[28] H.V.D. Parunak, Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis, in: Proceedings of the 1996 ICMAS, 1996.

[29] T. Suzuki, S. Shatz, T. Murata, A protocol modeling and verification approach based on a specification language and petri nets, IEEE Transactions on Software Engineering 16 (5) (1990) 523–536.

[30] R. David, H. Alla, Petri Nets and Grafcet: Tools for Modeling Discrete Events Systems, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[31] A. Mehra, V. Chiodini, An Integrated Development Environment for Distributed Multi-Agent Applications, http://www.gensym.com/successstories/adepaper.htm.

[32] J. Ferber, Les système multi-agents, InterEditions, 1996.

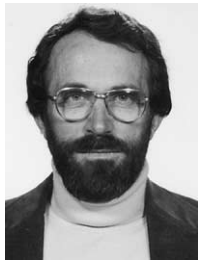
[33] M.-J. Yoo, W. Merlat, J.-P. Briot, Modeling and Validation of Mobile Agents on the Web, http://www.isima.fr/scs/wbms/d9/yoomj.html.

[34] R.G. Smith, The contract net protocol: high-level communication and control in a distributed problem solver, IEEE Transactions on Computers C-29 (12) (1980) 1104–1113.

[35] C. Sibertin-Blanc, R. Bastide, Object-oriented structuration for high-level petri nets, in: Proceedings of the 11th Conference on Application and Theory of Petri Nets, 1990.

[36] T.C.E. Cheng, M.C. Gupta, Survey of scheduling research involving due-date determination decision, Eur. J. Oper. Res. 38 (1989) 156–166.
[37] D. Wang, S.-C. Fang, T.J. Hodgson, A fuzzy due-date bargainer for the make-to-order manufacturing systems, IEEE Transactions on SMC, Part C 28 (3) (1998) 492–497.

Dr. Fuhua Lin is Associate Professor of Center for Computing and Information Systems at Athabasca University in Canada. Before joining Athabasca University, he was working at Intelligent System Group (ISG) at University of Calgary as a postdoctoral research fellow and the National Research Council of Canada as an Assistant Research Officer. He obtained PhD from Hong Kong University of Science and Technology (HKUST) in 1998. He was involved in many research projects on the development of intelligent systems. He is author or co-author of more than 20 publications. He is a member of the IEEE Computer Society and the ACM. His current research interests are related to the design and implementation of Distributed Intelligent Systems for e-Manufacturing and e-Learning, and the Agent-based Software Engineering.



Douglas H. Norrie currently holds the Nortel Chair in Intelligent Manufacturing at The University of Calgary, Alberta, Canada. Formerly, he was Head of the Department of Mechanical Engineering at The University of Calgary, and more recently Head of the Division of Manufacturing Engineering. He is also Adjunct Professor of Computer Science at the same institution. He has been a member of two major International Research Consortia in Intelligent Manufacturing Systems, being a founder member in the Holonic Manufacturing Systems and the Gnosis Knowledge Systematization consortia. He is author or co-author of numerous publications, including seven books. His research interests are in Intelligent Systems and, in particular, in Multi-Agent Systems.