A Multi-agent Based Information Infrastructure for Manufacturing

Manas R. Patra, Richard Moore

United Nations University, International Institute for Software Technology, Macau, China
mrp@iist.unu.edu, rm@iist.unu.edu

Keywords Multi-agent, Information Infrastructure, Formal Model

Abstract In the present competitive market environment manufacturing enterprises cannot function as monolithic entities. Rather they have to collaborate with other enterprises both for raw materials as well as for component fabrication. In this paper we propose a generic information infrastructure for manufacturing enterprises. The rationale behind building such an infrastructure is to facilitate information access which can help manufacturing enterprises of all sizes to collaborate. The model presented here is based on a multi-agent paradigm.

1 INTRODUCTION

Today's manufacturing companies are faced with global competition and fast changing customer requirements. In order to cope with this challenge they introduce variations in their products and at times completely new products. But this involves large investments both for building the necessary infrastructure and for other operational costs. Thus, manufacturing companies tend to collaborate with other enterprises both for raw materials and for component fabrication. This helps them concentrate more on innovative product design and quality considerations. But, with the mushrooming of small and medium-size enterprises it is difficult to gather information about the ones which can provide the required services reliably and at a competitive cost. In addition, even enterprises which build components rather than finished products need to search for bigger enterprises which require their components. This observation has motivated us to propose an information infrastructure to support manufacturing enterprises in their search for information about enterprises for possible collaboration.

Currently, a number of research projects are underway to develop infrastructures for supporting different functionalities of manufacturing enterprises [1,6,4]. Most of them emphasize the automation of internal processes, the management of internal activities, workflows, supply chain management, etc. and to our knowledge little attention has been paid to developing an information infrastructure that can help manufacturing enterprises collaborate in an effective way. We believe manufacturing enterprises of all sizes can greatly benefit from this kind of infrastructure.

Our model is based on a *Multi-agent paradigm* which is appropriate for inherently distributed application domains. We formally specify the requirements of such an infrastructure using the RAISE (Rigorous Approach to Industrial Software Engineering) Specification Language, RSL [5].

2 A MULTI-AGENT MODEL

In recent years, *Agent technology* has been propounded as a promising approach to handle applications that are information-rich and involve repetitive information processing. The *Multi-agent computing paradigm* makes use of this technology for modelling application domains which are inherently distributed. The basic agent architecture for the proposed model is depicted in Figure 1.

Figure 1 Agent Architecture

The agent software is conceptually organised in seven major components: *communication module*, *event management module*, *control module*, *interaction management module*, *agent knowledge base*, a library of *action primitives*, and a repository of *interaction templates*. The communication module, which is augmented with a *message-in-queue* and a *message-out-queue*, manages all incoming and outgoing messages. The behavior of an agent is modeled in an *event-driven* manner where events are triggered by

the arrival of messages. The event management module identifies events, sequences them (based on priority) before passing them to the control module, and signals errors in the case of unknown events. At the core of an agent software is the control module which is in fact an inference engine responsible for the control of the software. When an event is signaled the control module triggers the appropriate action. The actions are mostly predefined at the time of agent design and are maintained as a library of action primitives. The interaction management module monitors all ongoing interactions at an agent. The agent knowledge base is the information repository for an agent which encompasses knowledge of its own capabilities and of other agents with which it interacts.

2.1 The Agents

The information infrastructure that we model is depicted as a multi-agent environment. The environment is populated by enterprises of varying sizes involved in some form of manufacturing activity, namely fabricating components or making finished goods. The enterprises interact with each other through their personalized software agents.

We identify three different roles of agents: *buyers*, which represent enterprises intending to purchase components or products; *sellers*, which represent enterprises which are offering to supply components or products; and *brokers*, which serve as intermediaries between buyers and sellers to facilitate the buying and selling process. Any enterprise can of course adopt one or more of these different roles at any given time.

Both buyers and sellers can register with brokers, who maintain information about them and the products they are interested in buying or selling. Brokers may also pass requests for products they do not deal in themselves on to other brokers. Both buyers and sellers thus effectively have access to all the enterprises which are registered with the same brokers as themselves. This eases the search for information about smaller enterprises from whom a required component can be obtained.

Brokers can be useful to buyers for both *product brokering*, which addresses the problem of *what to buy*, and *merchant brokering*, which helps to determine from *whom to buy*. When sellers advertise their products to brokers, this information can be passed on to prospective buyers, and when buyers contact a broker regarding products or components they are interested in buying the broker can pass this information on to prospective sellers. Brokers may additionally act as mediators for any purchase deal that they arrange in this way. We include all these aspects in our model.

3 THE FORMAL MODEL

We now give an overview of our formal specification of the proposed information infrastructure using the RAISE specification language. Full details, which are omitted here for brevity, can be found in [3].

The abstract types *BuyerId*, *SellerId* and *BrokerId* are used to uniquely identify the buyer, seller, and broker agents respectively. Similarly *ProductId* uniquely identifies products, the details of each product (its category, brand, features, etc.) being specified by the record type *ProdDesc* with which the *ProductId* is associated in the map type *Products*.

```

BuyerId, SellerId, BrokerId, ProductId,
ProdDesc ::
  prodName : ProdName
  prodCategory : ProdCat
  brand : Brand
  feature : Feature  $\leftrightarrow$  chg_feature,
Products = ProductId  $\xrightarrow{m}$  ProdDesc

```

The information related to a seller includes the brokers with which it is registered together with details about the products it can supply, including their basic cost, discount rates, and the quantity of stock available. Since a seller can deal with many products we use the map type *MyOffer* to record this information for all the seller's products. The map type *Sellers* records information about all sellers in the market. Buyers are specified similarly, though we omit the details here for reasons of space.

```

SellerAddr, SellerBankDetail,
SellerInfo ::
  sbrokers : BrokerId-set  $\leftrightarrow$  chg_sbk
  sAddr : SellerAddr  $\leftrightarrow$  chg_sAddr
  sBank : SellerBankDetail  $\leftrightarrow$  chg_sBank
  canSell : MyOffer  $\leftrightarrow$  chg_canSell,
ProdOffer ::
  basicCost : Nat  $\leftrightarrow$  chg_cost
  discount : Nat  $\leftrightarrow$  chg_disc
  stockQty : Nat  $\leftrightarrow$  chg_qty,
MyOffer = ProductId  $\xrightarrow{m}$  ProdOffer,
Sellers = SellerId  $\xrightarrow{m}$  SellerInfo

```

In order to implement the required functionality of a broker, the broker must maintain information pertaining to its registered buyers and sellers as well as other brokers which it deals with. These form three fields of the record type *BrokerDB0* defined below. For the remaining two fields, *notavailProd* records information about products about which buyers have enquired but which the broker cannot currently supply - this information is

retained while the broker attempts to find a broker or a new seller who can deal with the request. Finally, the field *trans* records information about transactions which the broker is mediating. Each transaction is allocated a unique *TransId* and the details of the transaction - the buyer, seller and product involved together with the current status of the transaction - are recorded in the associated *Transac*. Various consistency or well-formedness conditions on brokers, for example that transactions can only involve registered buyers and sellers, are embodied in the function *is_wf_broker*, details of which are omitted here for brevity.

```

TransId,
Transac ::
  prod : ProductId
  buyer : BuyerId
  seller : SellerId
  status : TransState,
TransState == orderRecvd | shipmentConfid,
BrokerDB0 ::
  buyerList : BuyerId-set ↔ chg_buyerList
  sellerList : SellerId-set ↔ chg_sellerList
  notavailProd : ProductId → BuyerId-set ↔ chg_notavail
  brokerList : BrokerId-set ↔ chg_brokerList
  trans : TransId → Transac ↔ chg_trans
  no_of_quer : Nat ↔ chg_quer
  no_of_purs : Nat ↔ chg_purs,
BrokerDB = {l bdb : BrokerDB0 • is_wf_broker(bdb) l},
Brokers = BrokerId → BrokerDB

```

The information infrastructure as a whole is viewed as a market place populated by buyers, sellers and brokers dealing in particular products. We specify this using the record type *Market0*, which is composed of each of these four components. Again, consistency conditions, for example that the buyers and sellers registered with a broker must belong to the market, are captured by the well-formedness function *is_wf_market*.

```

Market0 ::
  buyers : Buyers ↔ chg_buyers
  sellers : Sellers ↔ chg_sellers
  brokers : Brokers ↔ chg_brokers
  products : Products ↔ chg_products,
Market = {l mk : Market0 • is_wf_market(mk) l}

```

3.1 A Dynamic View of the Model

Agents join and leave the market, new products are introduced, existing products become unavailable, and so on. The possibility of such changes is

incorporated into the model by defining functions which act on the data types defined above. Each function corresponds to one of the possible actions that can be performed in the market and has an appropriate precondition to ensure that the consistency conditions are preserved when it is executed.

We have defined a set of basic functions for updating information available in the electronic market. These include functions for registering a buyer or a seller with a broker; adding a new product to or removing an existing product from a seller's product range or a buyer's interests; recording a request from a buyer to a broker for a product which the broker cannot supply directly (by storing the request in the broker's *notavailProd* field; changing the details (address, bank details, etc. relating to buyers and sellers; and changing a seller's offer price, discount, etc. for a product. We explain one of these functions here. Other functions can be found in [3].

The function *add_Seller* is used to register a new seller with a broker. The seller is added to the broker's *sellerList* and the broker is added to the seller's *sbrokers*. The precondition ensures that both the broker and the seller belong to the market and that the seller is not already registered with the broker.

```

add_Seller : BrokerId × SellerId × Market  $\rightarrow$  Market
add_Seller(kid,sid,mk)  $\equiv$ 
  let nbinf =
    chg_sellerList(sellerList((brokers(mk))(kid))  $\cup$  {sid},(brokers(mk))(kid)),
    nkinf = chg_sbk(sbrokers((sellers(mk))(sid))  $\cup$  {kid},(sellers(mk))(sid)),
    cs = chg_sellers(sellers(mk)  $\uparrow$  [sid  $\rightarrow$  nkinf], mk)
  in
    chg_brokers(brokers(mk)  $\uparrow$  [kid  $\rightarrow$  nbinf], cs)
  end
pre is_valid_broker(kid,mk)  $\wedge$  sid  $\in$  dom sellers(mk)  $\wedge$ 
    sid  $\notin$  sellerList((brokers(mk))(kid))  $\wedge$  kid  $\notin$  sbrokers((sellers(mk))(sid))

```

3.2 Seller Selection

In a similar way, we have specified interactions between buyers, brokers and sellers, beginning with the request from a buyer to a broker regarding availability of a product through to the brokering of the sale. When a buyer contacts a broker for information about sellers of a product, it may wish to know all the sellers who are registered with the broker with a view to select a particular seller from this list itself, or it might provide the broker with some criteria for selecting an appropriate seller and ask the broker to select a seller based on these criteria. The appropriateness of a selection would depend on how closely it satisfied the criteria. In the following specification the variant type *SelMode* captures these two selection options. The record type

SellerReport gives the format of the response to the buyer's query.

```
SelMode == giveAll(ProductId) | preferential(ProductId, Criteria),
SellerReport ::
  prodId : ProductId
  sellerId : SellerId
  prodinfo : ProdOffer,
  Reply == replyFormat(SellerReport-set) | no_seller_found
```

The function *prodSellers* finds all the sellers of a specified product that are registered with a particular broker, while the function *applyCriteria* takes the buyer's specified criteria as an additional input parameter and selects only those sellers that satisfy the criteria.

```
prodSellers : BrokerDB × ProductId × Market → SellerId-set
prodSellers(db,pid,mk) ≡
  {sid | sid: SellerId • sid ∈ dom sellers(mk) ∧ sid isin sellerList(db) ∧
    pid ∈ dom canSell((sellers(mk))(sid))},
applyCriteria : Criteria × SellerId-set × ProductId → SellerId-set,
```

The function *select_seller* uses these functions to determine the appropriate set of sellers which match the request from a buyer.

```
select_seller : SelMode × BrokerDB × Market → AgentId-set
select_seller(sm, db, mk) ≡
  case sm of
    giveAll(pid) → prodSellers(db,pid,mk),
    preferential(pid, cr) → let f = prodSellers(db,pid,mk)
      in applyCriteria(cr, f, pid)
  end
end
```

4 AGENT COMMUNICATION

In order for the information infrastructure to be effective, agents must communicate with each other in an efficient way. We enforce structured communication among the agents by defining a set of communication primitives. These primitives are similar to the ones used in the KQML [2]. Some of the primitives that we define to facilitate agent communication include: *subscribe* and *unsubscribe* for agents to subscribe to or unsubscribe from services offered by a broker; *advertise* for agents to advertise their capabilities, e.g. the products they deal with; *enquire* for agents to submit a query about a particular product; *reply* to respond to queries; *request* for agents to request a particular service; and *commit* for agents to commit to provide a requested service. The interaction of the agents is governed by a

set of protocols such that the agents adhere to a well-defined pattern of communication. By well-defined we mean that agents know *what to communicate*, *when to communicate*}, and *with whom to communicate*. We do not address the issue of *how to communicate* because this as a lower-level network communication problem which involves message communication through Internet or Intranets whereas our emphasis is on the high-level communication which can be built on top of these lower-level details.

5 CONCLUSION

In this paper we have proposed a multi-agent based information infrastructure which can be used for the benefit of the manufacturing enterprises who would like to collaborate with each other. The formal approach adopted in specifying different aspects of the infrastructure ensures conceptual clarity in the design level and avoids inconsistencies during implementation.

REFERENCES

- [1] Bussmann S. Daimler AG. (1998). An Agent-Oriented Architecture for Holonic Manufacturing Control, Ist Open workshop IMS Europe, Lausanne, Switzerland
- [2] Finin T. et al. (1992). Specification of the KQML Agent Communication language, The DARPA Knowledge Sharing Initiative, External Interfaces Working Group
- [3] Patra M.R., Moore R. (2000). A Formal Model of an Agent-mediated Electronic Market, UNU/IIST Tech. Report No. 211, Macau, August 2000
- [4] Shen W., Xue D., Norrie D.H. (1997). An Agent-based Manufacturing Enterprise Infrastructure for Distributed Integrated Intelligent Manufacturing Systems, In Proc. of the 3rd Int. conf. on the practical application of Intelligent Agents and Multi-agents, London, UK, March 23-25, 533-548
- [5] The RAISE Language Group (1992). {The RAISE Specification Language}, Prentice Hall International (UK) Ltd.
- [6] Vision of the National Industrial Information Infrastructure Protocols (NIIP), <http://www.niip.org/vision.html>