Formal Model and Analysis of Multi-Agent Framework for Supply Chain Management

A.M. Rahman & A.O. Enikuomehin
Department of Computer Science
Lagos State University
Ojoo, Lagos, Nigeria
toyin@lasunigeria.org, ramalade@yahoo.com

ABSTRACT
Fulfilling customers’ requirements instantaneously is one of the major challenges faced by global competitive business processes in dynamic environment. Based on this, many enterprises have explored and adopted the Supply Chain Management (SCM) as an avenue to better compete and collaborate in creating customer-centered products. Multi-agent system (MAS) is very suitable to support this SCM in order to bring about prompt and concurrent production system considering the adaptive, flexible, mobile, and collaborative nature of agents. In this paper, we present an agent-based SCM framework in a virtual enterprise (VE) and formalize the model using Pi-calculus to enhance its reliability. Pi-calculus provides techniques and tools to verify and validate formal abstractions. We analyze and validate the model in order to ensure the consistency of system’s specification as part of its non-functional requirements.

Keywords: multi-agent system, supply chain management, formal methods, virtual enterprise, decentralized systems

1. INTRODUCTION

In meeting and even exceeding these customers’ demands, an organization often needs to cooperate and collaborate with its counterparts. Apart from time constraints in fulfilling these requests from clients, availability of both human and material resources of individual enterprise sometimes makes this cooperation very inevitable. These enterprises may virtually come together to better respond and take advantage of any business opportunity. This cooperation varies along product/service development phase, in marketing and sales activities and sharing of their business processes, resources, core competencies, skills and know-how.

The coming together of some business organizations to collaboratively respond and take advantages of business opportunities through sharing of skills and resources in a computer network forms a VE. Usually, VE formation is expected to reduce costs and time to market as well as increase in flexibility. Another important feature of VE is limitation in lifeline – they are meant to collaborate for certain duration in achieving an objective. Unlike the virtual organization (VO) that is not profit oriented, VE are meant to generate revenues for the participating enterprises. We propose in this paper a MAS framework of SCM in a virtual enterprise involved in product manufacturing.
Furthermore, we model the SCM framework by using the pi-calculus notation in order to assure unambiguous specification and improve the reliability of the system at its developmental stage. This contribution builds on [5], where we presented agent-based SCM in analogous to the digital ecosystem. We also illustrate how the formal model can be verified using existing tools. Our choice of this notation and tool was aimed at reducing redundant information by explicitly highlighting the information flow as well as processes coordination for reliability and lower chances of error. In this paper, we do not discuss specific MAS architecture or implementation. Previous work of our MAS platform called SAGE is discussed in [6]. Since our focus is on modeling and analysing the MAS using pi-calculus, the Belief, Desire and Intention (BDI) in agenthood as well as its formal model is outside the scope of this work. The remainder of the paper is structured as follows. In section 2, we relate the agent-based ADS with the SCM. Related work is discussed in section 3. The system’s architecture and the SCM stages involved are discussed in section 4. Section 5 presents our Pi-Calculus notation of the framework and followed by its analysis and verification. Conclusions and brief outlook on our future work is highlighted in section 6.

2. SCM AND AGENT-BASED ADS

The SCM involves the management of the series of activities ranging from raw materials acquisition, to manufacturing, distribution, transportation, warehousing, and product sales aimed at satisfying customers’ demands. An Internet-enabled supply chain makes it possible for online and real-time material acquisitions and payments as well as other stages of the SCM. This makes the SCM an excellent model of a decentralized system. However, major challenges of this system are coordination of rapid production planning and real-time response to changes in meeting customer’s requirements. Thus, there is need for such system to be autonomous and fault-tolerant in meeting these challenges. Systems with such features are referred to as autonomous decentralized systems (ADS).

Agents are most suitable to solve these problems. Agent-based approaches to business modeling take advantage of the nature of the loosely coupled entities called agents for better operations. These characteristics include flexibility, adaptive, communicative and intelligence ability. A community of agents coming together to solve a problem which is too complex to be tackled by a single agent in a system is referred to as a Multi-Agent System (MAS). MAS can support distributed collaborative problem solving by agent collections that dynamically organize themselves [7].

MAS, which seems suitable in developing an automated manufacturing process, has been used successfully in distributed systems in the Internet such as air traffic control, business process management, electronic commerce and information gathering [7].

3. RELATED WORKS

MAS has enjoined wide acceptance in developing practical applications both for commercial and industrial uses [8]. Agent-based approach was used by [9] in simulating real-world business problems. In [10], methodology for integration of the project-driven production planning based on agent-based engineering with the existing enterprise resource planning system of ExplanTech was introduced. This was aimed at facilitating optimization of resource utilization and supplier chain while meeting customer demands. Similar efforts are contained in [11, 12, 13, 14].

The intelligent feature being associated with agents makes it imperative for such agent-based systems to be formally specified in guaranteeing their reliability as described in [15, 16, 17]. However, these formal specifications failed to consider the adaptive and dynamic nature of agents which often resulted into parallel communication as against only sequential. In [18], Z notation was used for modelling MAS but could not cater for their interactions of parallel communication. Other work that had successfully used pi-calculus notation in formalizing agent systems includes [19, 20]. Still, these notations were not analyzed and validated to illustrate their correctness.

4. SYSTEM ARCHITECTURE

Four stages of the SCM as modelled in this work are controlled by agents. These stages are ordering, planning, execution and delivery. Orders are received from customers and plans are generated based on availability of raw materials, storage capacity and deadline specified by the client. A hybrid of build-to-order and make-to-stock product requests is assumed where customers can either specify certain kind of products to be branded by enterprise or get from readily forecasted and stocked ones based or prevailing circumstances. Once product order is raised, the logistic agent determines the capability of performing the operation based on the conversation with other agents in the system. After checking for feasibility internally, collaboration may be extended to other units in the VE if it has no capability of meeting clients’ demands based on some constraints like unavailability of raw materials and meeting a delivery date. After due considerations, this order may be accepted or rejected accordingly.

4.1 The Agents Roles

The Six types of agents are involved in the system. These are Order Agent (OA), Logistic Agent (LA), Production Agent (PA), Resource Agent (RA), Delivery Agent (DA) and Client Agent (CA). The agents’ roles are described extensively in [5]. These are briefly highlighted in section 5.1.
5. FORMAL MODEL USING Π-CALCULUS

The pi-calculus is a mathematical notation that can be used to describe processes’ interactions not only sequentially, but also for parallel communication [21]. Its syntax consists of a set of prefixes and process expressions which can be used to represent communication between processes through channels. While an independent thread of control is called a process, a channel is an abstraction of the communication link between two processes [22]. These interactions occur by sending and receiving of messages over channels. The communication channels among the agents in the system are shown in Fig. 1.

The channels of communication between the logistic and production agents is Manufact( ). For instance, the LA sends a message reqst to the PA and the reply from the PA, through the same channel Manufact ( ), is bounded to the variable msg. The pi-calculus syntax is defined in table 1 where P and Q are any two processes.

The definition of abstract service and service is given through the formal description of the syntax and semantic structure of channel. The relationship between channels is defined upon several factors including those used in constrained optimization problems for transportation or communication channel, which are usually defined by a Pi-calculus, and the validity of them is explained through an examples and notations.
Table 1. Syntax of Calculus Notation

<table>
<thead>
<tr>
<th>Process Representation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty $\emptyset$</td>
<td>The process where no action takes place.</td>
</tr>
<tr>
<td>Parallel $P</td>
<td>Q$</td>
</tr>
<tr>
<td>Output $\pi(x) \cdot P$</td>
<td>The process that sends message $x$ over a channel $a$ and behaves as process $P$ afterward.</td>
</tr>
<tr>
<td>Input $a(x) \cdot P$</td>
<td>The process that waits on channel $a$ to receive a value bound to variable $x$ and behaves as process $P$ afterward.</td>
</tr>
<tr>
<td>Non-deterministic choice $P + Q$</td>
<td>The process where either process $P$ or $Q$ runs.</td>
</tr>
<tr>
<td>Repetition $!P$</td>
<td>Infinite number of process $P$ running in parallel.</td>
</tr>
<tr>
<td>Match $[x = y]P$</td>
<td>The process that behaves as $P$ provided $x$ and $y$ are the same. Otherwise nothing happens.</td>
</tr>
<tr>
<td>Restriction $(vz)P$</td>
<td>The process that behaves as $P$. However, $z$ is a local channel and can only be used for communication only between processes within the scope of $P$.</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Nothing observable happens i.e. action without interaction with environment.</td>
</tr>
</tbody>
</table>

The symbol $\text{def}$ is to introduce definition of processes.

5.1 The Players

The agents in the system are formally defined using Pi-calculus notations as follows:

1. **The OA.** This agent accepts request from customer through the CA, sends it to the LA and terminates. After some internal system interaction i.e. planning activities (illustrated by the $\tau \rightarrow$ symbol), it gets feedback and communicates it to the CA.

$$OA \overset{\text{def}}{=} prorder(a, b, c) \cdot \overline{\text{order}(x, y, z) \cdot 0 \overset{\tau}{\rightarrow} \text{order}(msg) \cdot prorder(msg) \cdot OA'}$$

2. **The CA.** The CA sends product order to the enterprise, which comprises the product type, the quantity and the due date through the prorder channel. Feedback is also received from the enterprise.

$$CA \overset{\text{def}}{=} prorder(p, q, d) \overset{\tau}{\rightarrow} prorder(msg) \cdot CA'$$
4. **The RA.** It takes resource request from the LA and sends update information on resource availability and acquisition, then terminates; or contacts the other units in the virtual enterprise as may be necessary.

\[
RA \overset{\text{def}}{=} \text{resource}(\text{reqst}) \cdot \text{resource}(\text{msg}) \cdot 0 + \text{resource}(\text{reqst}) \cdot \text{virtual}(\text{reqst}) \cdot RA'
\]

5. **The DA.** This agent takes delivery request, sends necessary update and terminates.

\[
DA \overset{\text{def}}{=} \text{delivery}(\text{reqst}) \cdot \text{delivery}(\text{msg}) \cdot 0
\]

6. **The PA.** The PA takes manufacturing request and sends necessary information to the LA and awaits next instruction. The information include production feasibility and storage capability.

\[
PA \overset{\text{def}}{=} \text{delivery}(\text{reqst}) \cdot \text{manufac}(\text{msg}) \cdot PA'
\]

7. **The LA.** This agent, which is at the centre of the system, accepts processing order through the OA and simultaneously contacts the DA, the PA and the RA for production and delivery planning.

\[
LA \overset{\text{def}}{=} \text{order}(a, b, c) \cdot LA' \rightarrow \text{(delivery}(\text{reqst}) \not\mid \text{manufac}(\text{reqst}) \not\mid \text{resource}(\text{reqst})) \cdot LA'
\]

5.2 SCM Stages

1. **Ordering of product.** The CA sends product order to the OA which comprises the product type, require quantity and the expected date of delivery. This request is sent to the LA for necessary planning and feedbacks.

\[
\text{Ordering} \overset{\text{def}}{=} \text{prorder}(p, q, d) \cdot CA' \not\mid \text{prorder}(a, b, c) \\
\not\mid \text{order}(a, b, c) \cdot OA' \rightarrow \text{(order}(\text{msg}) \cdot LA' \\
\not\mid \text{order}(\text{msg}) \cdot \text{prorder}(\text{msg}) \cdot OA'
\]

2. **Delivery.** The LA sends delivery information to the DA which accepts and responds with the delivery update.

\[
\text{Delivery} \overset{\text{def}}{=} \text{delivery}(\text{reqst}) \cdot LA' \not\mid \text{delivery}(x) \cdot \text{delivery}(\text{update}) \cdot DA'
\]
4. Production planning. The LA sends information simultaneously to the RA and the PA. If the PA satisfies request, product is manufactured. Otherwise, the RA is expected to contact the units in the VE as may be necessary.

```
Planning def
(resource(x))·resource(msg)·RA′
\[\langle\text{manufact}\langle\text{reqst}\rangle\rangle\]
\[\text{LA} \cdot \text{manufact}(y) \cdot \text{manufact}(msg)\]
\[\cdot \text{PA} + \langle\text{resource}(z) \cdot \text{virtual}(\text{reqst}) \cdot \text{virtual}(msg)\]
\[\cdot \text{resource(update)} \cdot \text{RA}′\]
```

5. Planning execution. Here, the LA informs the PA of production plans based on the availability of materials. Also, the LA can ask the RA to contact the other units in the VE for makeup request and feedback is sought from corresponding unit(s) in the VE.

```
Execution def
\[\langle\text{manufact}\langle\text{reqst}\rangle\rangle\cdot \text{LA} ′ \cdot \text{manufact}(y) \cdot \text{PA} \rightarrow\]
\[\text{manufact更新} \cdot \text{0} + \langle\text{resource}(\text{reqst})\rangle \cdot \text{LA} \cdot \text{resource}(x)\]
\[\cdot \text{virtual}(\text{reqst}) \cdot \text{virtual}(msg) \cdot \text{resource}(msg) \cdot \text{RA}\]
```

5.3 Model Analysis and Verification

We have analyzed and verified the behaviour properties of above π-calculus notations using Another Bisimilarity Checker (ABC) tool [23], which checks for open equivalence between terms of the π-calculus. An open bisimulation refers to a family of binary relations between processes in the π-calculus that is indexed by a set of distinctions. A distinction $D$ is an irreflexive and symmetric binary relation between names and represents all the disequalities that should hold [23]. It can also be used to "simulate" free constant names. Two agents are said to be similar if they are related by some bisimulation; this means they can indefinitely mimic the transition of each other.

This tool can also be used in tracing the information flow as well as processes coordination for reliability and discovery of deadlocks. We check for strong bisimilarity between OA and RA by using the command eq after defining the two agents and the result shows that the agents are designed not to perform the same action and the derivatives do not lie in the same direction (see appendix I). We also discovered same characteristics between any pair of agents in our framework. More importantly, we traced the flow of information in the architecture for verifying the possibility of any deadlock in the system. The command step is used to view and simulate the commitments of the agents. The result shows there is free flow of information without any chance of redundancy and deadlock (see appendix II).

This verification and analysis of the notations ensures correct specifications of the model before implementation in guaranteeing the reliability of the system.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a conceptual framework for agent-based supply chain management and the architecture was formalized using π-calculus which supports parallel communications of dynamic systems such as MAS. The agents involved in the system support and participate actively in taking orders from clients, make feasible planning of production, execute the planned processes and likewise coordinate the prompt delivery of finished products. This formal model was analysed and validated using ABC tool which checked for bisimilarity and deadlock in the architecture.

Our contributions are two folds. Firstly, a MAS framework for the SC enhances its capability considering the flexible and autonomous nature of agents as well as their communicative property. Secondly, the π-calculus formalism specifies the system’s architecture and is useful in guaranteeing its reliability and effectiveness. Unlike other formal notations, pi-calculus effectively caters for not only the static state of our MAS system at certain instances, but adequately considers its dynamic nature as well. Thus, this error free model serves as a prelude to a reliable system to be built from the specification. Our next effort is to develop a prototype application on a scalable MAS platform such as SAGE [6].
REFERENCES


APPENDICES

Appendix I:
Bisimilarity check between order agent and resource agent

#usr/local/abc/abc
Welcome to Another Bisimulation Checker
abc > agent OA(prorder,order) =
    prorder.'order.t.order.'prorder.OA
Agent OA is defined.
abc > agent RA(resource,virtual) = resource.'resource.0 +
    resource.'virtual.RA
Agent RA is defined.
abc > eq OA RA
The two agents are not strongly related (2).
Do you want to see some traces (yes/no) ? yes
traces of OA RA
- %0-> -%0-> 'prorder.t.#1 'prorder.RA
- 'prorder-> -'prorder-> t.#1 (x0,x1)
  (x0.x1.RA + x0.0)#1 ::= prorder.%0.OA

Appendix II:
Verification of deadlock in the system
#usr/local/abc/abc
Welcome to Another Bisimulation Checker
abc > agent RA(resource,virtual) = resource.0 +
    resource.'virtual.RA
Agent RA is defined.
abc > agent PA(manufac) = manufac.'manufac.PA
Agent PA is defined.
abc > agent LA(order,delivery,manufac,resource) =
    order.t.'delivery|'manufac|'resource
Agent LA is defined.
abc > agent Execution(manufac,resource,virtual) =
    ('manufac.LA|manufac.t.'manufac.PA) +
    ('resource.LA|resource.'virtual.virtual.'resource.RA)
Agent Execution is defined.
abc > step Execution
1: { } => Execution --%0-> (%0.LA | t.%0.PA)
2: { } => Execution --resource-> (%virtual.#2 | 'resource.LA)
3: { } => Execution --%0->
    (x0,x1,x2,x3)(x2.0 | x3.0 | %0.t.%0.PA | x0.t.'x1.0)
4: { } => Execution --resource->
    (x0,x1,x2,x3)(x2.0 | x3.0 | resource.#5 | x0.t.'x1.0)
#1 ::= resource.RA
#2 ::= virtual.#1
#3 ::= resource.RA
#4 ::= virtual.#3
#5 ::= virtual.#4
Please choose a commitment (between 1 and 2) or 0 to exit:
1
1: { } => (%0.LA | t.%0.PA) --t-> (%0.PA | %0.LA)
2: { } => (%0.LA | t.%0.PA) --%0->
    (x0,x1,x2,x3)(x2.0 | x3.0 | x0.t.'x1.0 | t.%0.PA)

Please choose a commitment (between 1 and 2) or 0 to exit:
2
1: { } => (%0.LA | resource.#6) --resource->
    (resource.PA | %0.LA)
2: { } => (%0.LA | resource.#6) --%0->
    (x0,x1,x2,x3)(x2.0 | x3.0 | resource.#7 | x0.t.'x1.0)
#6 ::= resource.PA
#7 ::= resource.PA
Please choose a commitment (between 1 and 2) or 0 to exit:
0
abc >

Authors’ Bio

Rahman M. A. had his BSc, MSc. & PhD. in Computer Sciences from the University of Lagos, Akoka, Lagos. He also obtained his PGD (Education) from the Lagos State University (LASU), Ojo, Lagos. Presently, he is the acting HOD, Computer Science Department, LASU, Ojo. His research interests include Formal Methods in Software Engineering, Modeling & Simulation, Computational Methods, Cloud Computing, Information Systems, Medical Image Analysis and Computer Science Education. He is member of professional organizations including, IEEE, CPN, NCS, TRCN,STAN and NATT.

Enikuomehin, A. Oluwatoyin is a Lecturer at the Department of Computer Science, Lagos State University and the Head, Application Development and Data Services Unit of the University’s External System. His current research is concerned at developing applications for modelling emotions in information retrieval, search space reduction softwares, biologically inspired search engines and methods in software engineering. He is a member of several reputable associations like the British Computer Society, America Association of Artificial Intelligence, Nigeria Computer Professional Association, and Nigeria Association of Mathematical Physicist amongst others. Enikuomehin obtained his PhD in Computer Science from the University of Ilorin, Ilorin.