

# Business Process Model Based Multi-agent System Development

Li Guo, Dave Roberston, Yun-Heh Chen-Burger

CISA, Informatics, The University of Edinburgh, United Kingdom

E-mail: L.Guo@sms.ed.ac.uk, {Jessicac, Dr}@inf.ed.ac.uk

## Abstract

*Traditional workflow management systems are normally governed by automated processes which define the flow of work throughout the organizations. However, such processes know only about themselves, they do not possess any meta-level awareness. Furthermore, they are not designed to utilize or understand ontologies; they are not capable of autonomous action, intentional communication, or deliberately cooperative behavior. In contrast, agents possess all of these capabilities. In this paper, we propose an approach to designing multi-agent system interaction protocols using a business process model as the first step in multi-agent system based workflow system enactment.*

## 1. Introduction

Business enterprises are organizations that perform collective work. In order to achieve necessary operational efficiencies, this work needs to be governed by processes that define the flow of work throughout the organization. Regulated business processes are important because they reduce transactional costs when compared to ad-hoc approaches. Workflow management systems have been widely used to support the automation of business processes. Workflow[10] is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules. A workflow Management System[10] is a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications. Workflow normally comprises a number of logical steps, each of which is known as an activity. An activity can involve manual interaction with a user or a workflow participant, or the activity might be executed using machine resources. Automating the actual work provides increases in efficiency, and

provides managers with the facilities to create Virtual Organizations.

Multiagent systems emerged as a new research area in the early 1990's. The computing paradigm of multi-agent systems (MAS) has its origin in both distributed artificial intelligence (DAI) and object-oriented distributed systems. There is no consensus on the definition of software agents or of agency. However, the prevailing opinion is that an agent may exhibit three important general characteristics: autonomy, adaptation, and cooperation. Cooperation and coordination between agents is probably the most important feature of multi-agent systems. Unlike stand-alone agents, agents in a multi-agent system collaborate with each other to achieve common goals. In other words, these agents share information, knowledge, and tasks. The intelligence of MAS is not only reflected by the expertise of individual agents but also exhibited by the emerged collective behavior beyond individual agents. From a software engineering point of view, the MAS approach also may be an effective way to develop large distributed systems. Agents are relatively independent pieces of software interacting with each other only through message-based communication. So system development, integration, and maintenance might become easier and less costly. For instance, it should be easy to add new agents into the agent system when needed. Also, the modification of legacy applications might be kept to a minimum when they are to be brought into the system. Aside from adding communication capabilities to a legacy application, nothing else should need to change. In order to realize this potential benefits, cooperation and coordination of agents in a MAS requires agents to be able to understand each other and to communicate effectively with each other. The infrastructure that supports agent cooperation in a multi-agent system is thus seen to include at least the following key components.

- A common agent communication language and protocol.
- A common format for the content of communication.
- A shared ontology.

From a technical perspective, WFMSs bring together principles, methodologies and technologies from various areas of computer science and management science. For example, workflow techniques involve database management, client-server computing, heterogeneous distributed computing, graphical user interfaces, application and subsystem integration, messaging, document management, simulation, and business practices and re-engineering. However, the current generation of WFMS have some shortcomings:

- Relying on one central control: WFMSs have a central workflow server that defines and controls all business processes. In distributed enterprises or large companies, however, the business processes cannot be managed by a central point. Independent companies make their own decisions on how to do their pieces of business.
- Lack of automation: WFMSs only determine the process logic, but most of the activities currently are still fulfilled by human.
- Lack of reactivity[1, 2]: WFMSs require a pre-defined representation of a business process and all potential deviations from that process.
- Lack of resources management [1, 2]: WFMSs do not control the resources of a business process, and so rely on a business process being dimensioned beforehand.
- Lack of semantics[1, 2]: WFMSs lack an appreciation of the content of a business process and do not make decisions based on the nature of the information generated by a business process.
- Lack of generic interfaces: WFMSs need to exchange data between activities or interface to other applications. Currently, these operations depend on API calls. There should be some generic interfaces to eliminate the effort to develop interfaces between WFMSs and other applications.
- Lack of interoperation [1, 2]: independent vendors now develop WFMSs. Though Workflow Management Coalition has presented its work to enable the interoperation, this does not help much.

By comparing methods from multi-agent system with WFMC, we may able to address these shortcomings in the following ways:

- Distributed system architecture: For the scenario concerning multiple workflow systems, agent technology provides loose coupled distributed system structure for integrating distributed business process management systems.
- Automation: The inherent autonomy of software agents can fulfill activities as human substitution. Moreover, agents can start a workflow based on

event trigger or more complex reaction to environment changes.

- Interaction: Software agents enable organizations to interact with each other.
- Resource management: Agents can represent resources. Task assignments and resource allocations are done through negotiation among these agents.
- Reactivity: Agents react to changing circumstances and have the ability to generate alternative execution paths. This ability normally involves agents intelligent features.
- Interoperation among heterogeneous systems: Agents can be heterogeneous. The interactions rely on semantic messages for exchanging plans and service definitions. That makes interoperation more feasible than API calls.
- Intelligent decision-making: Some high-level features of agents are also very helpful in workflow management, though they are not matured techniques nowadays.

However, implementation of business process management systems using only agent technology has the following problems:

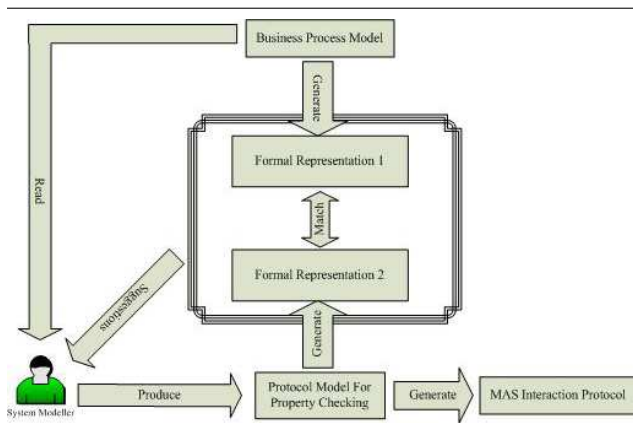
- Business process optimization is difficult due to the lack of explicit definitions and representations of the business processes.
- It is not easy to ensure that all the business process level properties are properly used in a multi-agent system.

In this paper we propose a business process model based interaction protocol engineering framework to facilitate the development of multi-agent system interaction protocol. In section 2, we describe our framework that supports MAS protocol production in details including: brief introductions to a business process modeling language and a MAS protocol language that we used as demonstrating example in this paper and also the basic notations of temporal logic; the definitions of our property checking model; how to represent the business process model and our property checking model in temporal logic and how to translating our property checking model to a MAS protocol language etc. We also address some problems we come across and some possible future work in section 3.

## 2. BPM Based MAS Interaction Protocol Engineering

A business process model normally only describes business requirement whereas a MAS protocol defines both business level and system level requirements. Thus, the

workflow enactment by a multi-agent system is difficult since business process models don't include any system level information. Normally, the MAS interaction protocol must be produced manually. Then, how can we make sure that all the properties defined in a business process model are presented by the MAS interaction protocol properly, since depicting a complex protocol is always error prone. The traditional way of solving such a problem is to do model checking or simulation after the MAS protocol is completed. The shortcoming of such testing approaches is that even if we can discover exactly what the problem is and where it is in the protocol, it is still very difficult to figure out why such a problem exists and how to solve it. In contrast, if we can do things right from the very beginning, and can make sure that we can always do the right thing in the process of protocol modeling, then we may save a lot of time and effort. To do this, we propose a business process model based interaction protocol engineering framework which is shown in figure 1.



**Figure 1. BPM based Interaction Protocol Engineering Framework**

In our framework a business process model only describes business logic level information and is used purely as a requirement for protocol modeling. In figure 1 the "business process model" is represented by sets of temporal logic facts(KB) which express the relationships between the properties. The "system modeler" reads the "business process model" and produces the "property checking model". The "property checking model" produced is also represented by temporal logic facts(KB1) so that we can check, when we employ a business logic property(P1) in the protocol, if  $KB1$  can imply  $P1(KB1 \models P1)$ . If not, a transformation  $\varphi$  has to be applied to  $KB1(\varphi(KB1))$  to produce  $KB2$  such that  $KB2$  implies  $P1(\varphi(KB1) \models KB2) \wedge (KB2 \models P1)$ . Thus, a MAS interaction protocol

modeling process eventually becomes the process of finding out a groups of actions  $\varphi_i$  which lead to  $\bigwedge_{i=0} \varphi_i(KB1) = KB$  where  $KB1$  is a knowledge base generated by protocol model and  $KB$  is a knowledge base got from given business process model. Once the "property checking model" is accomplished, it can be rapidly deployed by any other standard MAS interaction protocol language.

Temporal logic is chosen to build knowledge bases for both the business process model and MAS protocol. Although our approach doesn't rely on any particular workflow description language/protocol language, we will use Fundamental Business Process Modeling Language[3] and Lightweight Coordination Calculus[4] as demonstration example to show how our framework works.

## 2.1. Temporal Logic

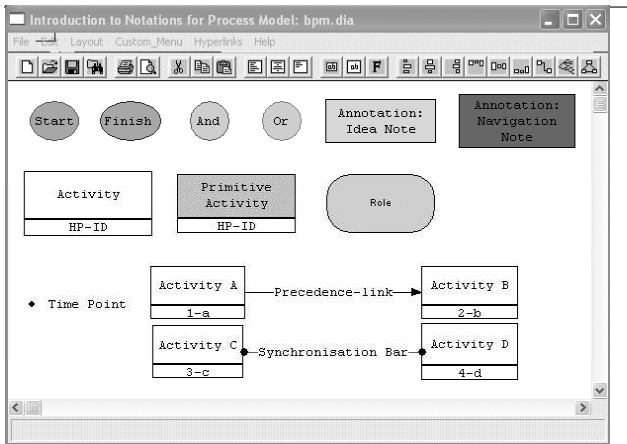
Temporal logic is a logic with a notion of time included. The formulas can express facts about past, present, and future states. Definitions of sets of typical temporal logic operators are given below:

$\Box\varphi$	$\varphi$ is true in all future moments
$\Diamond\varphi$	$\varphi$ is true in some future moment.
$\bigcirc\varphi$	$\varphi$ is true in the next moment in time
$\varphi U \psi$	$\varphi$ is true up <i>until</i> some future moment when $\psi$ is true

## 2.2. Fundamental Business Process Modeling Language

Fundamental Business Process Modeling Language(FBPML) is a logic based business process modeling language. There are three types of nodes in FBPML which are **Main Nodes**, **Junctions** and **Annotations**. Types of **Main Node** are "Activity", "Primitive Activity", "Role" and "Time Point". Types of **Junction** are "Start", "Finish", "AND" and "OR". Types of **Annotations** are "Idea Node" and "Navigation Node". The nodes are connected by **links** that are "precedence-link" and "synchronization-link". The above notations are show in figure 2, which is the palette window of AIAI KBST-EM modeling tool.

**Main Nodes:** Activity node describes a process which may be decomposed into sub-processes. A Primitive Activity is a leaf node activity that may not be further decomposed. In FBPML, an activity is uniquely identified by its name(i.e ID). It also allows the same activity to be repeated in different places in a process model. In this situation, because the repeated node may have different relation-



**Figure 2. Notations of FBPML depicted using KBST-EM**

ships to other activities, this node may be enacted differently. The semantics of an activity, therefore, are defined by three parts: location, usage ("Trigger" and "Pre-Condition") and content ("Action"). The definition of role is useful, an enabler may play a role that includes different activities and may have responsibilities for those activities. A role may be played by individuals, a group of people or software components. Time Points indicate particular points in time during the process model, the duration of a time interval is indicated by two time points. A length of time may also be defined without associations with any particular point of time.

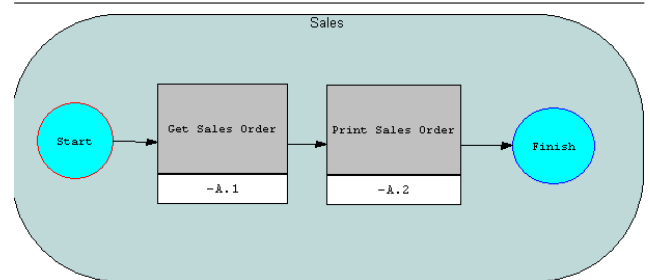
**Junctions:** The "Start" and "Finish" junctions indicate the beginning and end point of a process. They may also separate a sub-processes. "AND" and "OR" junctions describe one-to-many relationships constructing process flow and constraints between activities. They can be used in two ways: "join" and "split". "Join" indicates multiple paths merging to one path. "Split" describes a process splits into multiple paths. "AND\_Split" indicates that all the activities after it should be normally executed, whereas "OR\_Split" requires only one. "AND\_Joint" means it will not execute the next activity until all the activities before it finish, while "OR\_Joint" requires at least one activity before it completes in order to continue the next process.

**Annotations:** An "Idea Node" records information which is related to the processes but not part of the process model. A "Navigation Note" records the relationships between diagrams in a model. Neither of them contributes directly to the formal semantics of the process model, instead, they are used to help users to understand the model more clearly from an intuitive point of view.

**Links:** A "Precedence-Link" indicates a temporal constraint between two processes. If activity A is preceded by

activity B, a "Precedence-Line" may be drawn from A to B. It means that activity B cannot start until activity A finishes. A "Synchronisation-Bar" also places a temporal constraint between two time points. This notation enables any time points to be made equivalent and therefore enables the process operations to be synchronized.

To support the building of a business model, a knowledge-based support tool is provided:KBST-EM. Figure3 shows a screen shot of KBST-EM which is a part of a process model. This process model is written in FBPML that has been developed as a part of AKT[11].



**Figure 3. Sales Order Printing Process Model (screen shot from KBST-EM)**

### 2.3. Lightweight Coordination Calculus

The lightweight Coordination Calculus(LCC) is a language for representing coordination between agents. We can use it to describe distributed dialogue in multi-agent systems. A distributed dialogue is an interaction among a group of agents which can be described as a collection of dialogue sequences between agents. In a multi-agent system the speech acts conveying information between agents are performed only by sending and receiving messages. For example, suppose a dialogue allows an agent  $a(r1, a1)$  to send a message  $m1$  to agent  $a(r2, a2)$  and agent  $a(r2, a2)$  is expected to reply with message  $m2$ , assuming each agent operates sequentially, the sets of possible dialogue sequences we wish to allow for the two agents in the example are as given below, where  $M1 \Rightarrow A1$  denotes a message,  $M1$ , send to  $A1$ , and  $M2 \Leftarrow A2$  denotes a message,  $M2$ , received from  $A2$ .

$$a(r1, a1) :: (m1 \Rightarrow a(r2, a2) \text{ then } m2 \Leftarrow a(r2, a2))$$

$$a(r2, a2) :: (m1 \Leftarrow a(r1, a1) \text{ then } m2 \Rightarrow a(r1, a1))$$

We refer to this definition of the message passing behavior of the dialogue as the *dialogue framework*. Its syntax is as follows, where *Term* is a structured term and *Constant*

is constant symbol assumed to be unique when identifying each agent:

$$\begin{aligned}
Framework & := \{Clause, \dots\} \\
Clause & := Agent :: Def \\
Agent & := a(Type, id) \\
Def & := Agent | Message | Def \textit{ then } Def \\
& \quad | Def \textit{ or } Def | Def \textit{ par } Def \\
Message & := M \Rightarrow Agent | M \Rightarrow Agent \leftarrow C \\
& \quad | M \Leftarrow Agent | M \Leftarrow Agent \leftarrow C \\
C & := Term | C \wedge C | C \vee C \\
type & := Term \\
id & := Constant \\
Constant & := Term
\end{aligned}$$

A dialogue framework defines a space of possible dialogues determined by message passing, so the protocols allow constraints to be specified on the circumstances under which messages are sent or received. Two forms of constraints are permitted:

- Constraints under which message, M, is allowed to be sent to agent A. We write  $M \Rightarrow A \leftarrow C$  to attach a constraint C to output message.
- Constraints under which message, M, is allowed to be received to agent A. We write  $M \Leftarrow A \leftarrow C$  to attach a constraint C to input message.

For the earlier example above, to constrain agent  $a(r1, a1)$  to send message  $m1$  to agent  $a(r2, a2)$  when condition  $c1$  holds in  $a(r1, a1)$  we could write:  $m1 \Rightarrow a(r2, a2) \leftarrow c1$ .

Agent dialogue may also assume *common knowledge*, either as an inherent part of the dialogue or generated by agents in the course of a dialogue. This knowledge could be expressed in any form, as long as it can be understood by appropriate agents. We recognise the importance of preserving a shared understanding of knowledge between agents but cannot cover this issues in the current paper. As a dialogue protocol is shared among a group of agents it is essential that each agent when presented with a message from that protocol can retrieve the *state* of the dialogue relevant to it and to that message.

Pulling all the above elements together, we describe a LCC dialogue protocol as the term:

$$protocol(S, F, K)$$

Where S is the dialogue state; F is the dialogue framework (sets of dialogue clauses); and K is a set of axioms defining common knowledge assumed among the agents.

## 2.4. Representing A FBPML Business Process Model Using Temporal Logic

In a business process model, every activity is represented by sets of properties. Some of these properties are functional for the system implementation while some of them

are not but only are defined to facilitate human users. In addition, any two activities can not have complete same functional properties. Thus all the activities in a business process model can be distinguished only by using their functional properties without considering their non-functional properties such as IDs.

$$\begin{aligned}
& Primary\_Activity \left( \begin{array}{l} ID, Role, Preconditions, \\ Triggers, Inputs, \\ Outputs, Postconditions \end{array} \right) \\
& :: \\
& \square \left( \begin{array}{l} associate \left( \begin{array}{l} Role, \\ Preconditions \wedge \\ Triggers \wedge Inputs \end{array} \right) \\ \rightarrow \diamond associate \left( \begin{array}{l} Role, \\ Postconditions \wedge Outputs \end{array} \right) \end{array} \right)
\end{aligned}$$

FBPML notation *Primary\_Activity* above defines a primary activity in a business process model. *ID, Role, Preconditions, Triggers, Inputs, postconditions, outputs* are properties associated with it. Symbol "::<" means that the notations from its left hand side can be expressed by the temporal logic from its right hand side. Predicate *associate(Role, Properties)* defines the association of role and properties in a business process activity. The temporal logic clause here indicates that the conjunction of *preconditions, Triggers and Inputs* can always imply the conjunction of *postconditions and outputs* at some future time, which actually describes the task of the FBPML activity.

- 1 >  $link(ID, Precedence\_Link, A_1, A_2) :: A_1 \wedge \diamond A_2$ .
- 2 >  $link(ID, or\_Split, A_1, \{A_2, A_3, \dots, A_i\}) :: A_1 \wedge \diamond(A_2 \vee A_3 \vee \dots, \vee A_i)$
- 3 >  $link(ID, and\_Split, A_1, \{A_2, A_3, \dots, A_i\}) :: A_1 \wedge \diamond(A_2 \wedge A_3 \wedge \dots \wedge A_i)$
- 4 >  $link(ID, or\_joint, \{A_2, A_3, \dots, A_i\}, A_1) :: (A_2 \vee A_3 \vee \dots \vee A_i) \wedge \diamond A_1$
- 5 >  $link(ID, and\_joint, \{A_2, A_3, \dots, A_i\}, A_1) :: (A_2 \wedge A_3 \wedge \dots \wedge A_i) \wedge \diamond A_1$

Predicate *link* indicates how the activities from a business process model are connected. Different link types are defined in FBPML, such as *Precedence\_Link* which represents a sequence order between two activities. Above temporal logic clauses define time relations between sets of FBPML activities. For simplicity, all the FBPML activities are represented by symbol  $A_{i, i \in I}$ .

A very simple example is used here to show how a business process model can be expressed by temporal logic.

Business process model in figure 4 is defined from the view of *sales* agent. There are two primary activities in this model which are *input sales order* and *print sales order* and the functional properties associated with both them are given in table 1:

The above example business process model is represented

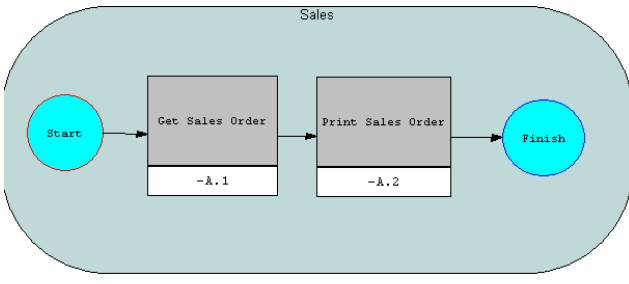


Figure 4. Sales Order Printing Process

Property	Input Sales Order
Precondition	getInput(X), validInput(X)
Input	null
Postcondition	null
Output	saleOrder(X)
Property	Print Sales Order
Precondition	null
Input	saleOrder(X)
Postcondition	printed(saleOrder(X))
Output	null

Table 1. Functional Properties of Primary Activities in Sales Order Printing Process

as follows using temporal logic facts:

- 1 >  $\square \text{associate}(\text{sales}, \text{getInput}(X))$
- 2 >  $\square \text{associate}(\text{sales}, \text{validInput}(X))$
- 3 >  $\square \text{associate}(\text{sales}, \text{saleOrder}(X))$
- 4 >  $\square \text{associate}(\text{sales}, \text{printed}(\text{saleOrder}(X)))$
- 5 >  $\square \left( \begin{array}{l} \text{associate} \left( \begin{array}{l} \text{sales}, \\ \text{getInput}(X) \wedge \\ \text{validInput}(X) \end{array} \right) \\ \rightarrow \diamond \text{associate}(\text{sales}, \text{saleOrder}(X)) \end{array} \right)$
- 6 >  $\square \left( \begin{array}{l} \text{associate}(\text{sales}, \text{saleOrder}(X)) \rightarrow \\ \diamond \text{associate} \left( \begin{array}{l} \text{sales}, \\ \text{printed}(\text{saleOrder}(X)) \end{array} \right) \end{array} \right)$
- 7 >  $\square \left( \begin{array}{l} \left( \begin{array}{l} \text{associate}(\text{sales}, \\ \text{getInput}(X) \wedge \\ \text{validInput}(X)) \\ \rightarrow \diamond \text{associate}(\text{sales}, \\ \text{saleOrder}(X)) \end{array} \right) \\ \wedge \diamond \left( \begin{array}{l} \text{associate}(\text{sales}, \\ \text{saleOrder}(X)) \\ \rightarrow \diamond \text{associate}(\text{sales}, \\ \text{printed}(\text{saleOrder}(X))) \end{array} \right) \end{array} \right)$

## 2.5. Building Property Checking Model

Several agent protocol languages are available for multi-agent system development, such as FIPA-ACL[5], KQML-KIF[6], LCC etc. But none of them are suitable for property checking. For example, in FIPA-ACL, message constraints are defined in BDI model, which makes it impossible to be checked in the process of protocol modeling. LCC protocol is defined from the view of single agent, which makes the time information contained in the protocol implicit. Therefore, In order to facilitate protocol property checking and separate the checking method from particular protocol language, we define a very simple MAS interaction protocol modeling language: Simple Protocol Properties Checking(SPPC) Language. A SPPC protocol model is built based on the message passing between two agents and the constraints associated with this message. Once the SPPC protocol model is decided, the time relationships among all the messages can also be derived. Thus the SPPC model can be represented by temporal logic directly and easily.

**Representing an message in SPPC:** Any message defined in SPPC model may be defined by a tuple of five characteristics underpinned by a formal representation. The tuple is:  $\text{msg}([\text{preconditions}], \text{message body}, [\text{postcondition}], \text{sender and receiver})$  where a *message body* only can be sent out from its *sender* when its *precondition* holds and can cause *postcondition* when it is received by its *receiver*.

**Temporal order between messages:** *Then* expresses a temporal constraint between two messages. In SPPC, it means that once the passing of message A is finished, the message B connected with it by *then* becomes "Temporal Qualified" and can thus be considered for passing.

**Junctions:** A junction is a control point in SPPC model. There are two types of junctions: "Par" and "Or". The two junctions define a one-to-many relationship between connected messages and indicate conjunction and disjunction points of a SPPC model.

The syntax of SPPC is as follows:

$$\begin{aligned}
 \text{SPPCModel} &:= \{ \text{Def}, \dots \} \\
 \text{Def} &:= \text{Message} | \text{Def then Def} | \\
 &\quad \text{Def or Def} | \text{Def par Def} | \\
 &\quad \text{invoke} \\
 \text{Message} &:= \text{msg}(\text{mid}, \text{pre}(C), \text{mb}(\text{Term}), \\
 &\quad \text{post}(C), \text{Agent}, \text{Agent}) \\
 \text{pre}(C) &:= \text{Term} | \text{pre}(C) \wedge \text{pre}(C) | \\
 &\quad \text{pre}(C) \vee \text{pre}(C) \\
 \text{post}(C) &:= \text{Term} | \text{post}(C) \wedge \text{post}(C) | \\
 &\quad \text{post}(C) \vee \text{post}(C) \\
 \text{Agent} &:= \text{sender}(a(\text{id}, \text{Type})) | \\
 &\quad \text{receiver}(a(\text{id}, \text{Type})) \\
 C &:= \text{Term} \\
 \text{mb}(\text{Term}) &:= \text{Term} \\
 \text{Type} &:= \text{Term}
 \end{aligned}$$

$$\begin{aligned} mid &:= Constant \\ id &:= Constant \end{aligned}$$

The predicates  $pre(C)$ ,  $mb(Term)$  and  $postcondition(C)$  defined above are the preconditions, message body and postconditions of the message being passed.

It should be kept in mind that SPPC is developed only for the purpose of MAS protocol property checking. In other words, it is not a valid protocol language that can be used by agents directly. However, the SPPC protocol can be translated into other formal protocol languages such as LCC etc. A possible SPPC model for our example business process model shown in figure 4 is given in figure 5.

It might be noticed that in the SPPC model shown in figure 5, there are several agents which are not defined in the example business process model. The reason for this is that when we build a multi-agent system we should try to use the existing agents as much as possible instead of building new ones to fit the input business process models every time. Thus, we have to consider the availabilities of the agents we needed and the capabilities of those agents. The scenario we try to handle is that all the agents that coordinate to perform the system are already available. In our example, agents *Inputdevice*, *Sales*, *Printer*, *Admin* are picked up by MAS protocol modeler to perform the intended business process model. Also, we ignore all the ontology issues in this paper.

## 2.6. Representing SPPC Using Temporal Logic

In a SPPC model, each message may have preconditions or postconditions or both. The time relationship between them is clear. The preconditions of a message must hold before the message can be sent out and the postcondition cannot hold until the message is received. The relationship between two messages is exactly same as with the relationship between two activities in FBPMML model. Thus, a SPPC model can be represented by temporal logic using the following conceptual mapping principles:

$$\begin{aligned} 1 &> msg \left( \begin{array}{l} mb(A), sender(Role), \\ receiver(Role1) \end{array} \right) :: \\ &\quad \square associate(Role, A) \\ 2 &> msg \left( \begin{array}{l} pre(A), mb(B), sender(Role), \\ receiver(Role1) \end{array} \right) :: \\ &\quad \square \left( \begin{array}{l} associate(Role, A) \\ \rightarrow \diamond associate(Role, B) \end{array} \right) \\ 3 &> msg \left( \begin{array}{l} mb(B), post(C), sender(Role), \\ receiver(Role1) \end{array} \right) :: \\ &\quad \square \left( \begin{array}{l} associate(Role, B) \\ \rightarrow \diamond associate(Role1, C) \end{array} \right) \end{aligned}$$

$$\begin{aligned} 4 &> msg \left( \begin{array}{l} pre(A), mb(B), post(C), \\ sender(Role), receiver(Role1) \end{array} \right) :: \\ &\quad \square \left( \begin{array}{l} associate(Role, A) \rightarrow \\ \diamond \left( \begin{array}{l} associate(Role, B) \\ \rightarrow \diamond associate(Role1, C) \end{array} \right) \end{array} \right) \\ 5 &> msg(A) then msg(B) :: A \wedge \diamond B \\ 6 &> msg(A) par msg(B) :: A \wedge B \\ 7 &> msg(A) or msg(B) :: A \vee B \end{aligned}$$

In the above temporal logical clauses, the properties *preconditions* and *message body* in a SPPC model are associated with the *sender* and *postconditions* are associated with *receiver*. Associating properties and roles in this makes it possible to do role's checking.

## 2.7. Issues For Role's Checking.

After representing both business process model and SPPC model in temporal logic, the relationships between properties can be derived and checked. However, another issue we have to check in the process of protocol modeling is if the right properties are associated with the right roles. In a business process model, every activity can only have one role for certain business scenario. However, a MAS interaction protocol model describes extra system information, which brings new roles in. Intuitively, if the role specified in the predicate *associate* in requirement is also specified in protocol specification and the properties associated with it are same, we can conclude that the properties are associated with right role in protocol. The assumption we make here is that ontology of roles is already available.

## 2.8. Generating A MAS Protocol From A SPPC Model

We will use LCC as our example to illustrate how a SPPC model can be translated into to a concrete MAS protocol. Although the main components of both SPPC and LCC are *messages and constraints*, they are built on different concepts. With LCC, the MAS interaction protocol is defined from the views of different agents where each agent has its own behavior definitions, whereas with SPPC, the protocol model is built based on the messages passing, which means the SPPC model is viewed from the aspect of messages but not agents. However, from the notations of SPPC and LCC we can see that SPPC is eventually a subset of LCC, so a SPPC model does contain all the information we need to construct a corresponding LCC protocol. *Message body, sender and receiver* from SPPC model together indicate the message being passed and direction of it in LCC. *Junctions* in SPPC can be used as LCC's *operators*.

---

```

msg(m.1, mb(getInput(X)), sender(a(i1, Inputdevice)), receiver(a(s1, Sales)))
then
msg(m.2, pre(vaildInput(X)), mb(saleOrder(X)), sender(a(s1, Sales)), receiver(a(p1, Printer)))
then
(
  msg(m.3, mb(printed(saleOrder(X))), sender(a(p1, Printer)), receiver(a(s1, Sales)))
  or
  (
    msg(m.4, pre(err(Y)), mb(askForHelp(Help)), sender(a(p1, Printer)), receiver(a(a1, Admin)))
    then
    msg(m.5, mb(response(Help)), sender(a(a1, Admin)), receiver(a(p1, Printer)))
    then
    msg(m.6, mb(failed(saleOrder(X))), sender(a(p1, Printer)), receiver(a(s1, Sales)))
    then
    invoke(m.2)
  )
)

```

**Figure 5. A SPPC Model for Sales Order Printing Process**

---

The notation *invoke* in a SPPC model indicates the ending point of the loop and the parameter of it indicates the the starting point of it. When translating a SPPC model with loops to a LCC protocol, all the messages between *invoke* and the message being invoked can be extracted to define the behaviors of a new role for loop, as long as the message invoked is not the first message defined for that agent.

In our example SPPC model, there is a loop that starts at the point *m.2* and ends at *invoke(m.2)*. We can see that the message identified by *m.2* is not the first message sent by the *a(s1, Sales)*, so all the *a(s1, Sales)* defined between *m.2* and *invoke(m.2)* should be replaced by *a(loop(X), Sales)* to deal with the loop. After the replacement, a temporal SPPC model for the translation purpose is shown in figure 6. A LCC protocol for the example SPPC model generated from the temporal SPPC model is given in figure 7:

Agent *a(loop(X), Sales)* is used to deal with the loop defined in SPPC model.

### 3. Conclusion and Future Work

In this paper, We propose a framework for modeling Multi-agent system protocol starting from a business process model as the first step in the workflow system enactment. With our framework, a business process model can be expressed by a MAS interaction protocol correctly. A simple language SPPC is defined for property checking purposes and any protocol model defined by SPPC can be translated into other standard protocol language(in this case LCC). Using our framework, lots of effort can be saved in the process of MAS protocol modeling since requirements level errors can be discovered in real time, which is different with the typical protocol modeling engineering method. Furthermore, using our approach, any revision to a existing protocol can also be checked in real time to make sure

all the business logic level changes are correct and compatible with former specification.

Future work includes: implementing a graphic modeling tool based on the framework to facilitate protocol modeling using existing diagram based modeling language like AUML; enlarging the range of property checking so we can check not only business logic level properties but also system level properties in real time; Translating SPPC model to more standard agent protocol langues such as FIPA-ACL/KQML-KIF.

### Acknowledgement

This work is partially funded by the UK Engineering and Physical Sciences Research Council under grant numbers GR/N15764/01 and Advanced Knowledge Technologies Interdisciplinary Research Collaboration. The AKT IRC research partners and sponsors are authorized to reproduce and distribute reprints and on-line copies for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of other parties.

### References

- [1] K. Trammel, Workflow without fear, Byte, April 1996.
- [2] P.D. O'Brien and M.E. Wiegand, Agent based process management: applying intelligent agents to workflow, the Knowledge Engineering Review, Vol. 13/2. 161-174, Cambridge University Press, 1998.
- [3] Yun-Heh Chen-Burger, Austin Tate, and Dave Robertson, Enterprise Modelling: A Declarative Approach for FBPML, European Conference of Artificial Intelligence, Knowledge Management and Organisational Memories Workshop.

---

```

msg(m.1, mb(getInput(X)), sender(a(i1, Inputdevice)), receiver(a(s1, Sales)))
then
msg(m.2, pre(vaildInput(X)), mb(saleOrder(X)), sender(a(loop(X), Sales)), receiver(a(p1, Printer)))
then
(
  msg(m.3, mb(printed(saleOrder(X))), sender(a(p1, Printer)), receiver(a(loop(X), Sales)))
  or
  (
    msg(m.4, pre(err(Y)), mb(askForHelp(Help)), sender(a(p1, Printer)), receiver(a(a1, Admin)))
    then
    msg(m.5, mb(response(Help)), sender(a(a1, Admin)), receiver(a(p1, Printer)))
    then
    msg(m.6, mb(failed(saleOrder(X))), sender(a(p1, Printer)), receiver(a(loop(X), Sales)))
    then
    invoke(m.2)
  )
)

```

**Figure 6. A Mid-SPPC Model For The Translation To LCC Protocol**

---

```

a(i1, Inputdevice) :: getInput(X) ⇒ a(s1, Sales)

a(s1, Sales) :: getInpux(X) ⇐ a(i1, Inputdevice) then a(loop(X), Sales)

a(loop(X), Sales) :: saleOrder(X) ⇒ a(Printer, p1) ← validInput(X) then
  (
    printed(saleOrder(X)) ⇐ a(p1, Printer) or
    (
      failed(saleOrder(X)) ⇐ a(p1, Printer) then a(loop(X), Sales)
    )
  )

a(p1, Printer) :: saleOrder(X) ⇐ a(loop(X), Sales) then
  (
    printed(saleOrder(X)) ⇒ a(loop(X), Sales) or
    (
      askForHelp(Help) ⇒ a(a1, Admin) then
      response(Help) ⇐ a(a1, Admin) then
      failed(saleOrder(X)) ⇐ a(loop(X), Sales) then a(p1, Printer)
    )
  )

a(a1, Admin) :: askForHelp(Help) ⇐ a(p1, Printer) then response(Help) ⇒ a(p1, Printer)

```

**Figure 7. LCC Protocol For Sales Order Printing Process**

---

- |   |  |
|---|--|
| <p>[4] Dave Roberston, <i>A Lightweight Method for Corrdination of Agent Oriented Web Services</i>, Proceedings of AAAI Spring Symposium on Sematic Web Services, California, USA, 2004.</p> <p>[5] FIPA. FIPA communicative act library speciation, <a href="http://www.pa.org/specs/pa00037/XC00037H.html">http://www.pa.org/specs/pa00037/XC00037H.html</a>, 2000.</p> <p>[6] KQML as an agent communication language Tim Finin, Yannis Labrou, and James Mayfield, in Jeff Bradshaw (Ed.), <i>Software Agents</i>”, MIT Press, Cambridge.</p> <p>[7] Li Guo, Yun-Heh Chen-Burger, Dave Roberston <i>Mapping a business process model to a semantic web service model</i>, Proceeding of IEEE International Conference on Web Services 2004.</p> <p>[8] Paul A. Buhler, Jose M.Vidal, Harko Verhagen <i>Adaptive Workflow = Web Services+Agents</i>, Proceeding of IEEE International Conference on Web Services 2003.</p> | <p>[9] Christopher D. Walton <i>Model Checking Multi-Agent Web Services</i>, Proceeding of AAAI Symposium of Semantic Web Services 2004.</p> <p>[10] <a href="http://www.wfmc.org/">http://www.wfmc.org/</a>.</p> <p>[11] <a href="http://www.aiai.ed.ac.uk/project/akt">http://www.aiai.ed.ac.uk/project/akt</a>.</p> |
|---|--|