Using Multi-agent Platform For Pure Decentralised Business Workflows

Li Guo, Dave Robertson and Yun-Heh Chen-Burger

CISA, Informatics, The University of Edinburgh, United Kingdom,
L.Guo@sms.ed.ac.uk, {dr,Jessicac}@inf.ed.ac.uk

Abstract. This paper describes the development of a distributed multi-agent workflow enacting mechanism starting from a BPEL4WS[BPE03] specification. Our work demonstrates that a multi-agent protocol (Lightweight Coordination Calculus)[Rob04] can be derived from a BPEL4WS specification to enable pure decentralised business workflows. The key difference between our work and other existing multi-agent based systems is that our approach gives a pure distributed architecture (has no centralised controller by any means) for deploying workflow system in an open environment (internat). Moreover, with out approach, existing workflow construct methodologies and business process models can be adopted in as much as possible for the MAS development.

1 Introduction

Workflow processes within organisations often involve a large number of resources, people and tools distributed over a wide geographic area. Workflow management systems are used for the purpose of automating the coordination of these diverse elements. Thus, in order to suit the nature of the application environment and the technology adapted, workflow applications are becoming distributed [GAK95][JGM98][Yan00]. Problems remain unsolved in current research of distributed workflow system due to the centralised system architecture, i.e, bad performance, vulnerability to failures, poor scalability, user restrictions, and unsatisfactory system openness. Moreover, as web services and Grid services become the more popular as the reference model for business resources, workflow can provide a powerful framework for composing individual services into complete solutions. However, the client-server architecture is not suitable for such areas where workflow technology is used in conjunction with services. This is because the client-server architecture is rather closed to facilitate external services (web services) available on the internet. Thus, it is better to have an open architecture which allows external services to be used.

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. Cooperation and coordination between agents is probably the most important feature of multi-agent systems. Unlike those 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 among themselves. 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 is also proven to be an effective way to develop large distributed systems. Since agents are relatively independent pieces of software interacting with each other only through message-based communication, system development, integration, and maintenance may in some cases become easier and less costly. For instance, it is easy to add new agents into the agent system when needed. Also, the modification of legacy applications can 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 is required to change.

However, cooperation and coordination of agents in a MAS requires agents to be able to understand each other and to communicate with each other to achieve their common goals. This thus requires control mechanisms to ensure the agents in a MAS always behave properly and effectively towards the final goal. A multi-agent interaction protocol is used for this purpose, which defines both the sequences of the messages that the agents must follow and the constraints associated with the messages. There are many interaction protocol languages that have been defined for describing the protocols, such as, FIPA-ACL[FIP00], KQML[FF94], LCC. However, when applying MAS to the domain of business workflow management, an obvious problem is that it is almost impossible to get the overall view of the underlying business processes involved in the workflow, since the protocol only specifies the message passing between different participants at implementation level. In the workflow management world, users care about not only the automation of their work, but also the underlying processes' objective understanding and analysis. For those analytical purposes, interaction protocol based system specifications are not enough since they involve too much system level information with high level business requirements hidden.

Business process modelling languages, in contrast, are well recognized for their value in organizing and describing a complex, informal domain in a more precise semi-formal structure that is intended for more objective understanding and analysis. Based on these advantages, they have been wildly used in conventional workflow management system and there are many mature techniques and tools which have been developed for supporting the business process model based workflow system development. However, such languages and tools are designed particularly for conventional workflow management architecture, they can not easily be adapted for new system architectures like multi-agent systems. Therefore, when building MAS based workflow management systems, almost all the existing techniques and tools for supporting conventional workflow management system's development are wasted as well as the business process models that are described by some formalised business process modelling languages, which means huge amount of repeat work has to be done during the course of MAS development. For example, verification and validation of formalised system specifications has to be re-performed even when the existing business process models have been verified and validated for a conventional system architecture. In addition, business process modelling languages used in workflow management systems sometimes are built with specifical features, for instance, BPEL4WS[BPE03] is designed for web services based

distributed workflow system. By using such languages, new platform can be combined with existing technologies.

In this paper, we discuss how to use business process models for pure decentralised workflow system deployment on a MAS platform. In section 2, we carry a in depth survey of some of existing MAS based workflow enacting approaches and discuss the features and problems of them. The necessary background knowledge of the concrete demonstrating languages both for describing business process models and MAS interaction protocols are given in section 3. Our MAS based decentralised workflow system enacting mechanism is explained in detail in section 4 including: research problem analysis; a syntax based language mapping between BPEL4WS and LCC and most of the important algorithms that we developed. Section 5 describes the design rationale of the agents in our system. Conclusions of our work and some possible future work are given in section 6.

2 Literature Review

2.1 Conventional Distributed Workflow Systems Based On a Client-server Architecture

Many research efforts have been undertaken on the topic of workflow distribution in conventional distributed workflow environment. The importance of associating workflow management with distribution has been emphasised, e.g., [PMG98][EP99][PHM99]. Also, some conceptual approaches and research prototypes have been proposed, which aim at tackling these problems by making conventional distributed workflow management systems more sophisticated.

ADEPT stands for Application Development based on Encapsulated pre-modelled Process Templates [MRD03]. One important facet in the ADEPT project is to investigate distributed workflow control in order to avoid overloading of the workflow servers and of the communication network. To address these problems, ADEPT reduces the network load by partitioning workflow definitions and by migrating the control of workflow instances from one server to another during run-time, i.e., a workflow instance may no longer be controlled by only one workflow server. Furthermore, ADEPT supports both static and variable server assignments [BD99]. The former means appropriate workflow servers are chosen for various partitions of a workflow definition. On the contrary, variable server assignment allows for dynamic workflow server assignment at run-time, which may improve the system performance significantly. As web services become more and more popular and widely used as organisations' interfaces, several approaches have been proposed to deploy web service based distributed workflow systems in which web services are clients and a centralised workflow engine is used to control the whole process that is carried between different web services. Two major approaches for such system are business process execution language for web services (BPEL4WS)[BPE03] and OWL-S[OWL01].

2.2 Pure Decentralised Workflow Approaches Based On Multi-agent/Peer-to-peer

While conventional distributed workflow approaches fail to properly address the problems caused by the client-server architecture, the emergence of computing technologies such as multi-agent and peer-to-peer have provided new platforms for process support solutions. Some research effort, although limited, has been put into investigation of using these collaborative and decentralised frameworks to support workflow management systems.

Little-JIL [AWS00], a language for programming the coordination of agents, is an executable, high-level process programming language with a formal (yet graphical) syntax and rigorously defined operational semantics. Little-JIL is based on two main hypotheses. The first is that the specification of coordination control structures is separable from other process programming language issues. The second is that processes can be executed by agents who know how to perform their tasks but can benefit from coordination support. Another ongoing p2p-based workflow project is conducted at Manchester Metropolitan University. This project presents a p2p architecture for dynamic workflow management, which is based on concepts such as Web Workflow Peers Directory (WWPD) and Web Workflow Peer (WWP)[FK]. The WWPD is a centralised feature of the system, which provides a peer registration service and maintains a list of active peers and their profiles. During the execution of workflow instances, Workflow process administration is achieved by employing a notification mechanism. It is claimed that such an approach is adaptive, easily scalable and flexible. SwinDeW[J.Y04] is a pure peer-to-peer based system for workflow management. It removes both the centralised data repository and the centralised workflow engine from the system. workflow participants are facilitated by automated peers which are able to communicate and collaborate with one another directly to fulfil both build-time and run-time workflow functions. With SwinDeW, performance bottlenecks in workflow systems are likely to be eliminated whilst increased resilience to failure, enhanced scalability, better user support and improved system openness are likely to be achieved. Its extended system SwinDeW-S also supports web services based service composition based on OWL-S.

2.3 Discussion

The systems that are based on conventional distributed workflow architecture (client-server) do add some distribution to workflow systems and bring benefits such as improved performance, increased failure resiliency and enhanced scalability as they claimed. However, these approaches mainly address distribution instead of decentralisation. A common characteristic of these approaches is that they are still based upon and limited by the client-server architecture. Thus, these approaches either address the problems partly, or require complicated languages and/or complex algorithms. The remaining centralised services like centralised process instantiation and work assignment also make them relatively inflexible in some application domains. Moreover, the aspects of user support and system openness are hardly ever considered. To summarise, the problems related to the centralised system architecture have not been and probably cannot be

addressed thoroughly when the whole workflow system is still based on a client-server architecture.

The few attempts at combining multi-agent/p2p computing technology with workflow technology that we discuss have opened new ground in workflow, and in the process support area in general. The distinguished features of multi-agent/p2p technology make it suitable to address the problems related to the client-server architecture ultimately. The potential of multi-agent based/p2p-based workflow, which offers significant value to organisations, is revealed in existing approaches. However, it is evident from the literature that research on implementing workflow in a multi-agent/p2p environment is still at a very initial stage with many problems addressed insufficiently. Fakass work on WWPD and WWP, only reports conceptual ideas about linking workflow. Other approaches, mainly focus on decentralised enactment of process instances at run-time in order to remove potential performance bottlenecks, increase fault tolerance and offer enhanced scalability. However, some issues which are essential to decentralised enactment have not been specified clearly in these approaches. For example, it is not clear in these approaches how the process definition data are managed so that decentralised agents/peers are able to access task information at run-time. Subsequently, problems of process instantiation are not addressed by these approaches. Issues such as dynamic participants, work allocation and better user support also have not been addressed sufficiently.

SwinDeW addresses most of the problems and offers a good platform for purely decentralised workflow management. However, the problem for SwinDew is that it builds everything from scratch. The language it uses is a process oriented language with agents' coordinating mechanisms embedded consequently. It blurs the business level requirements and system level requirements. When new technologies come out, their existing work can not be easily incorporated. It also ignores all the existing technologies that are used for supporting workflow management system development and all the existing models that have been created for conventional workflow system, which means repeating established work. Little-Jil falls into the same problem category as SwinDeW.

3 Background Knowledge

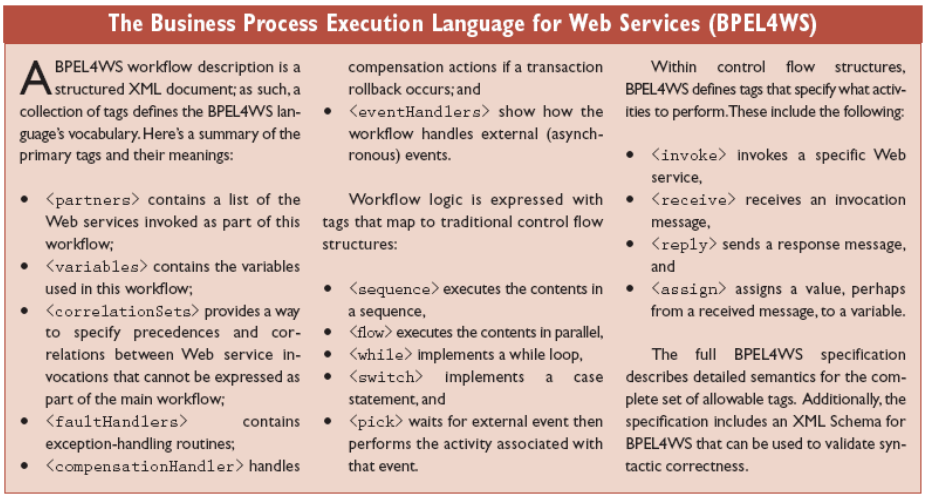
To demonstrate our idea more clearly, we use two concrete languages from both business workflow and MAS world, which are BPEL4WS LCC. In this section, necessary background knowledge of them is explained.

3.1 Business Process Execution Language For Web Services

As introduced in literature review, the Business Process Execution Language for Web Services (BPEL4WS) is an XML-based language for describing workflow in a distributed environment using web services. With the support from IBM and Microsoft, it has become the de facto standard for workflow description. As an executable process implementation language, the role of BPEL4WS is to define a new Web service by composing a set of existing services. Thus, BPEL4WS is basically a language to

implement such a composition. The interface of the composite service is described as a collection of WSDL portTypes, just like any other Web service. The composition (called the process) indicates how the service interface fits into the overall execution of the composition. Figure 1 illustrates this outer view of a BPEL4WS process.

A workflow described in BPEL4WS details the flow of control and any data dependencies among a collection of web services being composed. When enacted, the composition itself becomes available as a meta-web service, eligible for inclusion in other compositions. BPEL4WS requires that all web services be described with available WSDL descriptions. Due to the industry's increased focus on business process man-



The Business Process Execution Language for Web Services (BPEL4WS)

A BPEL4WS workflow description is a structured XML document; as such, a collection of tags defines the BPEL4WS language's vocabulary. Here's a summary of the primary tags and their meanings:

- <partners> contains a list of the Web services invoked as part of this workflow;
- <variables> contains the variables used in this workflow;
- <correlationSets> provides a way to specify precedences and correlations between Web service invocations that cannot be expressed as part of the main workflow;
- <faultHandlers> contains exception-handling routines;
- <compensationHandler> handles compensation actions if a transaction rollback occurs; and
- <eventHandlers> show how the workflow handles external (asynchronous) events.

Workflow logic is expressed with tags that map to traditional control flow structures:

- <sequence> executes the contents in a sequence,
- <flow> executes the contents in parallel,
- <while> implements a while loop,
- <switch> implements a case statement, and
- <pick> waits for external event then performs the activity associated with that event.

Within control flow structures, BPEL4WS defines tags that specify what activities to perform. These include the following:

- <invoke> invokes a specific Web service,
- <receive> receives an invocation message,
- <reply> sends a response message, and
- <assign> assigns a value, perhaps from a received message, to a variable.

The full BPEL4WS specification describes detailed semantics for the complete set of allowable tags. Additionally, the specification includes an XML Schema for BPEL4WS that can be used to validate syntactic correctness.

Fig. 1. Basic BPEL4WS Syntax[JVS04]

agement and acceptance of BPEL4WS, vendors are producing new software tools for workflow design, specification, and enactment. An example of one such tool is IBM's BPEL4WS Java Runtime (BPWS4J) (http://www.alphaworks.ibm.com/tech/bpws4j) platform.

3.2 Lightweight Coordination Calculus

The lightweight Coordination Calculus(LCC) is a language for representing coordination between distributed 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) (r1 represents the role of the agent and a1 is the ID of it) 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.

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

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:

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

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 by 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.

An 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 issue 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 [Rob04].

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.

4 From BPEL4WS Based Conventional Workflow System to LCC Based Multi-agent Platform

4.1 Problem Analysis

If we consider the interactions described in a BPEL4WS process model from the multi-agent point of view, it involves two sorts of agents: service providing agents (substitutes/proxies of web services) that is in the role of $< myRole >/< partnerRole >$ and a coordinating agent (workflow server) that is defined implicitly in BPEL4WS. The responsibility designated on the coordinating agent (workflow server) as analysed in previous sections is too heavy, which is understandable because BPEL4WS was initially designed for the coordination of web services which only have very limited computing capabilities. However, with software agents that have stronger computing capabilities, the burden of the coordinating agent can be shared. If we can dispatch the tasks that were performed by the workflow server (coordinating agent) to service providing agent, the process models that are used in conventional workflow system can be used in multi-agent based platform. Thus, to enable the MAS based distributed workflow system, the first step is to decide what sorts of tasks are performed by workflow server and how they can be dispatched to agents.

The most widely used technique for connecting two different specification based systems is syntax based language mapping. After the two languages that are used for describing the specifications in different system are mapped, any specification that is written in one language can be translated into another automatically. The two system can thus be ensured to be functionally equivalent to each other. This is illustrated in figure 2:

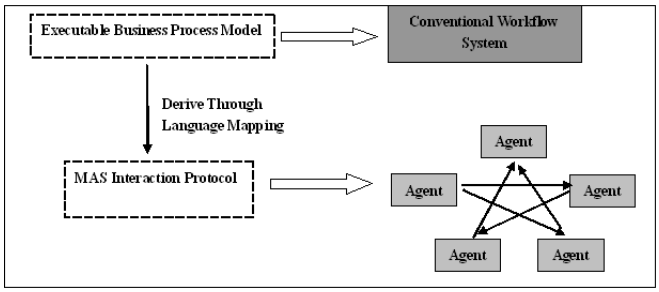


Fig. 2. Connecting Workflow System and Multi-agent Via Language Mapping

A BPEL4WS process model defines four main concepts which are:

– Partners: define the roles that participate the interaction. It should be noticed, partner notation in BPEL4WS defines the partner from the point of view of centralised workflow server. All the participants that can interact with workflow server are defined as partners ($< partnerRole >$) of it and workflow server is able to change its role ($< myRole >$) in order to interact with different participants.

– Message passing activity: defines the message that takes place between two participants. Such activities are: $< receive >$, $< invoke >$ and $< reply >$.
– Computing activity: carries the real workflow computation. Such activities are: $< assign >$, $< terminate >$ etc.
– Structure activity: controls the execution order of message passing activities and computing activities. Such activities are: $< sequence >$, $< switch >$, $< while >$ etc.

Except for $Partners$, execution of the other three sorts of activities are all carried by the workflow server. When executing a message passing activity, what the workflow server does is simply to pass and to forward the messages from/to participants (P_1 and P_2). If P_1 and P_2 can communicate directly with each other, the workflow server is not required at all. Structure activities define only the execution order of basic activities and if the IP protocol languages such as LCC has the syntax of describing such information, workflow server can also be removed since the control of time order is built in the protocol. When the protocol is passed between agents, such information is also transferred. However, problems arise for computing activities. In a BPEL4WS specification based workflow system, the computing activities must be executed by the workflow server and such activities cannot be easily dispatched to service providing agents. The following partial BPEL4WS model shows the problem. For simplicity, it is expressed using plain text:

$$
\begin{array}{l}
sequence \\
\{ \\
\quad a = b, \\
\quad P_1\ send\ M_1\ to\ P_2, \\
\quad c = d \\
\}
\end{array}
$$

Three roles are defined in the above chunk (P_1, P_2 explicitly and workflow server implicitly). If we want to eliminate the role of the workflow server in the desired MAS, the assign clauses, $a = b$ and $c = d$ have to be executed by P_1 or P_2 wholly or separately. To decide which agent should execute what computing activity is a difficult task, at least automatically, because the BPEL4WS specification is randomly defined. Therefore, in order to translate a BPEL4WS specification is ambiguous on this point, the BPEL4WS model must be specified following specific pattern, say, the computing activities must be defined before at least one message passing activity so that the sender of the message passing activity can perform the computing activities before sending a message out.

Given the above analysis, our aim is to ensure that all the tasks performed in a BPEL4WS based conventional workflow system can be realised by a MAS if the BPEL4WS specification can be completely/partially represented by LCC protocol. In the following sub-sections, we discuss in detail of how to perform the syntax based language mapping between BPEL4WS and LCC.

4.2 Simple Protocol Property Checking Language

Syntactically a BPEL4WS model is too far away from a LCC protocol for direct automatic translation in a single stage. Thus, we adopt a intermediate language, simple protocol property checking (SPPC)[LCB04], as a tool to help the translation from

BPEL4WS to LCC since syntactically it is close to BPEL4WS and LCC. Its syntaxes is as follows:

$$
\begin{aligned}
SPPC\ Model &::= \{Def, ...\} \\
Def &::= Message | Def\ then\ Def | Def\ or\ Def | Def\ par\ Def | invoke(mid) \\
Message &::= msg(mid, pre(C), mb(Term), post(C), Agent, Agent) \\
pre(C) &::= Term | pre(C) \wedge pre(C) | pre(C) \vee pre(C) \\
post(C) &::= Term | post(C) \wedge post(C) | post(C) \vee post(C) \\
Agent &::= sender(a(Type, ID)) | receiver(a(Type, ID)) \\
C &::= Term \\
Condition &::= Term \\
mb(Term) &::= Term \\
Type &::= Term \\
mid &::= Constant \\
lid &::= Constant \\
ID &::= Constant
\end{aligned}
$$

From the above SPPC syntax it can be seen that in SPPC a interaction that takes place between two agents/participants are described in a single $msg(...)$ in the similar way that BPEL4WS does. But unlike BPEL4WS, SPPC only has $msg(...)$ as its basic components but doesn't has individual clauses for representing computation units (computing units are parts of $msg(...)$). All the computation units in SPPC are defined as pre/post-conditions of message passing clauses, which is quite similar to the constraints defined by LCC. Also SPPC uses the same operator that LCC uses to control the sequence of $msg(...)$. Thus, using SPPC as a intermediation, performing language mapping between BPEL4WS and LCC is relatively easier. In the following sub-section, we will explain in detail of how to perform language mappings from BPEL4WS to SPPC and from SPPC to LCC one by one.

4.3 Performing language mapping from BPEL4WS to SPPC

Translation from partners defined in BPEL4WS to SPPC roles All the participants defined in $< partnerLinks >$ in a BPEL4WS model are directly mapped to the agents (sender and receiver) in SPPC no matter whether they are $< myRole >$ or $< partnerRole >$. An example BPEL4WS definition of participants is:

$$
\begin{aligned}
&< partners > \\
&\quad < partner\ name = "customer" \\
&\qquad\qquad serviceLinkType = "lns : loanApproveLinkType" \\
&\qquad\qquad myRole = "approver" / > \\
&< /partners >
\end{aligned}
$$

There are two roles defined in the above clauses and in the derived SPPC model two agents ($a(customer, ID)$, $a(approver, ID_1)$) directly correspond to them, where ID and ID_1 in each role are used to identify the agent. They can be anything that is identifiable such as URL.

Translation from message passing activities defined in BPEL4WS to SPPC message BPEL4WS message passing activities as classified are $< receive >$, $< invoke >$ and $< reply >$. The translating principles for them are different.

– The activity $< receive >$ in BPEL4WS means that a web service operation will not be invoked until certain requests (inputVariable of web service operations) arrive. From multi-agent point of view, the semantic of this activity is quite simple: a message sender (partnerRole) sends a message to a service provider (myRole). Thus this activity leads to a basic SPPC message that is:

$$msg\left(\begin{array}{l} mid, pre([]), mb(portType : operation : inputVariable), \\ post([update(inputVariable), store(portType : operation : inputvariable, ID)]) \\ sender(a(partnerRole, ID_1)), receiver(a(myRole), ID) \end{array}\right)$$

In a conventional BPEL4WS based workflow system, the variable values are stored in the centralised workflow server and can be used and updated at any time needed. However, in a LCC based multi-agent system, there is no centralised data store, thus, the values of all the variables have to be passed together with the LCC protocol (defined in LCC common knowledge) and messages between the agents.
The post-condition $update(inputVariable)$ defined in the above SPPC message is used to record/update the value of the variable involved in the incoming message in LCC common knowledge. It used as a constraint for all the incoming messages for receivers. We will not specify it in every SPPC message for simplicity. The post-condition $store(partnerRole : protType : operation : inputVariable, ID)$ is used to record the the ID of the message sender so that later on, a response will be sent back to the right agent (service requestor). This constraint is used to represent the coherent relation between $< receive >$ and $< reply >$ activities in BPEL4WS.

– The $< reply >$ construct allows the business process to send a message in reply to a message that was received through a $< receive >$. The combination of a $< receive >$ and a $< reply >$ forms a request-response operation on the WSDL portType for the process. The SPPC message for $< reply >$ derived is

$$mb\left(\begin{array}{l} mid, pre([fetch(partnerRole : portType : operation : variable, ID_1)]), \\ mb(portType : operation : variable), post([]), \\ sender(a(myRole, ID)), receiver(a(partnerRole), ID_1) \end{array}\right)$$

The constraint $fetch(partnerRole : portType : operation : variable, ID_1)$ is used to find the proper ID of the agent that sent request, which, together with constraint $fetch(partnerRole : protType : operation : inputVariable, ID)$, are used to keep the semantic of combination of $< receive >$ and $< reply >$ activities.

– The $< invoke >$ construct allows the business process to invoke a one-way or request-response operation on a portType offered by a partner. Thus, the corresponding SPPC clauses can be one message or two message units connected by "then", which depends on weather or not the $outputVariable$ is defined. If it isn't defined, the corresponding SPPC message is:

$$mb\left(\begin{array}{l} mid, pre([fetch\_variable(inputVariable)]), mb(portType : operation : inputVariable), \\ post([]), sender(a(myRole, ID)), receiver(a(partnerRole), ID_1) \end{array}\right)$$

If the $outputVariable$ is defined, the SPPC messages are:

$$mb\left(\begin{array}{l} mid, pre([fetch\_variable(inputVariable)]), mb(portType : operation : inputVariable), \\ post([]), sender(a(myRole, ID)), receiver(a(partnerRole), ID_1) \end{array}\right)$$
$$then$$
$$mb\left(\begin{array}{l} mid, pre([fetch\_variable(outputVariable)]), \\ mb(portType : operation : inputVariable : outputVariable), \\ post([update(outputVariable)]), sender(a(partnerRole, ID)), receiver(a(myRole), ID_1) \end{array}\right)$$

From the above analysis, it is clear that all the message passing activities in BPEL4WS can be translated to SPPC clauses.

Translation from computing activities defined in BPEL4WS to SPPC constraints The computing activities defined in BPEL4WS are $< assign >$, $< wait >$ etc. Since the translation principles for all of them are the same, we only discuss the translation of $< assign >$ in detail. The activity $< assign >$ in BPEL4WS specification defines the internal variable assignation of the BPEL4WS workflow engine and it gives the BPEL4WS computational ability. In SPPC, constraints (post-conditions and pre-conditions) are the only places where we can do computation. Therefore, the computation carried by the centralised BPEL4WS server, as addressed earlier, should be dispatched to the agents in the multi-agent system as constraints for them to satisfy. Eventually, which agent executes what constraints doesn't matter too much since the execution of constraints is not role specific in BPEL4WS. The only issue is how the execution order between the computing activities and the other activities is kept in the generated SPPC model, which requires the consideration of structure activities also.

Translation from structure activities defined in BPEL4WS to SPPC model BPEL4WS structure activities control the execution order between the activities (message passing activities, computing activities and structure activities) that are nested within them. SPPC, however, uses operators to control the execution orders of basic message clauses. The temporal order between computing clauses and message passing clauses are represented by the relation between messages and their constraints. Therefore, a BPEL4WS structure activity might be represented by two SPPC notations together: $1 >$SPPC operators and $2 >$combinations of constraints and messages. The principles of when the content of a structure activity should be represented by SPPC using operators and when they should be represented by the combinations of messages and pre-conditions/post-conditions in SPPC, are quite different for different BPEL4WS structure activities.

– The $< sequence >$ activity allows us to define a collection of activities to be performed sequentially in lexical order in BPEL4WS. Using the SPPC "then" operator, the relation between message passing activities and structure activities can be kept without changing anything. To represent relations between message passing activity/structure activity and computing activity in a $< sequence >$, the following re-write rules have to be applied

$(A_1\, then\, A_2) \Rightarrow (E_1 \rightarrow C)$ $\quad if\ (A_1\, is\, a\, message\, passing\, activity) \wedge (A_1 \rightarrow E_1) \wedge (A_2\, is\, a\, computing\, activity) \wedge (A_2 \rightarrow C)$ $\quad (rule_1)$

$(A_1\, then\, A_2) \Rightarrow (C_1 \wedge C_2)$ $\quad if\ (A_1\, is a computing\, activity) \wedge (A_1 \rightarrow C_1) \wedge (A_2\, is\, a\, computing\, activity) \wedge (A_2 \rightarrow C_2)$ $\quad (rule_2)$

$(A_1\, then\, A_2) \Rightarrow (C_1 \rightarrow E_2)$ $\quad if\ (A_1\, is\, a\, computing\, activity) \wedge (A_1 \rightarrow C_1) \wedge (A_2\, is\, a\, structure\, activity) \wedge (A_2 \rightarrow E_2)$ $\quad (rule_3)$

$(A_1\, then\, A_2) \Rightarrow (E_1\, or\, E_2)$ $\quad if\ (A_1\, is < while > activity) \wedge (A_1 \rightarrow E_1) \wedge (A_2\, is\, not\, a\, computing\, activity) \wedge (A_2 \rightarrow E_2)$ $\quad (rule_4)$

$(C_1 \rightarrow (E_1\, or/par ... or/par\, E_n))$ $\quad if\ (C_1 \rightarrow E_1) \Rightarrow E_i, ..., (C_1 \rightarrow E_n) \Rightarrow E_{i+n}$ $\quad (rule_5)$
$\Rightarrow (E_i\, or/par ... or/par\, E_{i+n})$

$((E_1\, or/par ... or/par\, E_n) \rightarrow C_1)$ $\quad if\ (E_1 \rightarrow C_1) \Rightarrow E_i, ..., (E_n \rightarrow C_1) \Rightarrow E_{i+n}$ $\quad (rule_6)$
$\Rightarrow (E_i\, or/par ... or/par\, E_{i+n})$

$C_1 \rightarrow A_1$ and $A_1 \rightarrow C_1$ in the above rules means C_1 is used as the precondition/post-condition of A_1. $Rule_1$ and $rule_2$ means that a computing activity that is defined

before/after a message passing activity in a $< sequence >$ can be used as the pre-condition/post-condition of the SPPC message that is derived from the message passing activity. The re-write rules for dealing with the computing activity defined before/after a structure activity are expressed in $rule_3$. $Rule_4$ is used to deal with a special case where a $< while >$ activity is involved in a $< sequence >$. The time relation between $< while >$ and the activity defined after it in $< sequence >$ is not a sequential order but is a exclusive "or" order. The $condition$ specified for the $< while >$ activity actually controls the execution of it. If the $condition$ holds, the $< while >$ activity is executed repeatedly and only when the $condition$ fails, the activity specified after $< while >$ can get executed. $Rule_5$ and $rule_6$ are used to assign the pre-condition/post-condition to the SPPC messages which are connected by "or"/"par".

The algorithm for translating a $< sequence >$ into SPPC clauses is shown in figure 3. it should be noticed that a computing activity can never be used as the last

procedure $translate\_sequence(Sequence,\ CL\ SPPC)$
input: $Sequence,\ the\ BPEL4WS\ <sequence>\ activity$
$CL,\ a\ list\ that\ stores\ all\ un-assigned\ conditions$
output: $SPPC,\ SPPC\ clauses\ derived\ from\ given\ <sequence>\ activity$
$initiate\ a,\ pointer\ (P_1),\ and\ let\ it\ point\ to\ the\ first\ element\ Sequence\ and\ CL$
while $(P_1\ is\ not\ pointing\ to\ the\ last\ element\ Sequence)$
$fetch\ the\ activity(A)\ that\ P_1\ is\ pointing\ to\ in\ Sequence$
if $(A\ is\ a\ computing\ activity)$
$translate\ A\ into\ conditions\ and\ put\ it\ at\ the\ end\ of\ CL$
$make\ P_1\ point\ to\ next\ activity$
else if $(A\ is\ a\ message\ passing\ activity)$
$fetch\ all\ the\ condition\ in\ CL\ use\ them\ as$
$pre-conditions\ of\ the\ SPPC\ message(S)\ drived\ from\ A$
$empty\ CL\ and\ make\ P_1\ point\ to\ next\ elment\ of\ CL$
$SPPC = SPPC\ then\ S$
else if $(A\ is\ a\ structure\ activity)$
$translate\_structure\_activity(A, CL, SPPC_1)$
$SPPC = SPPC\ then\ SPPC_1$

Fig. 3. Algorithm for deriving a SPPC model from a BPEL4WS $< sequence >$ activity

element of a $< sequence >$ activity as discussed earlier. Otherwise, the automatic translation is very hard to achieve. Therefore, not all the existing BPEL4WS models can be directly translated into SPPC models using this algorithm .

– The <**switch**> construct allows us to select exactly one branch of activity from a set of choices. A $< switch >$ activity can be represented using SPPC "or" notation in the following manner,

$$(C_1 \rightarrow A_1)\ or\ (C_2 \rightarrow A_2)\ or\ ...$$

where C_i means the conditions defined for $< case >$ in $< switch >$ and A_i represents the activities that are defined as the content for each $< case >$ which could be basic activities or structure activities. The translation of the content of each $< case >$ relies on the types of them. The $condition$ defined for each $< case >$ can be translated using the following re-write rule as well as the rules defined for $< sequence >$:

$$(C_1 \rightarrow A_1) \Rightarrow (C_1 \rightarrow E)\ if\ A_1 \Rightarrow E\ (rule_7)$$

The algorithm for translating a BPEL4WS $< switch >$ activity to a SPPC model is given in figure 4.

procedure $translate\_switch(Switch,\ CL,\ SPPC)$
input: $Switch,\ the\ BPEL4WS\ < switch > activity$
$CL,\ a\ list\ that\ stores\ all\ un-assigned\ conditions$
output: $SPPC, SPPC\ clauses\ derived\ from\ given\ < switch > activity$
for $(each\ branch(B)\ of\ switch)$
$extract\ the\ conditions\ defined\ for\ each\ branch\ and\ put\ in\ CL$
$extract\ the\ contents(C)\ of\ B$
$translate\_structure\_activity(C, CL, SPPC_1)$
$SPPC = SPPC\ or\ SPPC_1$

Fig. 4. Algorithm for deriving a SPPC model from a BPEL4WS $< switch >$ activity

– The $< flow >$ construct allows us to specify one or more activities to be performed concurrently. It creates a set of concurrent activities directly nested within it. It further enables expression of synchronization dependencies between activities that are nested directly or indirectly within it. The link construct is used to express these synchronization dependencies. A link has a name and all the links of a flow activity must be defined separately within the flow activity. The standard source and target elements of an activity are used to link two activities. The source of the link specify a source element specifying the link's name and the target of the link specify a target element specifying the link's name. The following example shows that links can cross the boundaries of structured activities. There is a link named "CtoD" that starts at activity C in sequence Y and ends at activity D, which is directly nested in the enclosing flow. This synchronisation link confines the execution order of activity C and activity D. Under its control, activity D must be executed after the execution of activity C.

$< flow >$
$< links >$
$< link\ name = "CtoD" / >$
$< /links >$
$< sequence\ name = "Y" >$
$< receive\ name = "C" \ldots >$
$< source\ linkName = "CtoD" / >$
$< /receive >$ $< invoke\ name = "E" .../ >$
$< /sequence >$
$< invoke\ partnerLink = "D" \ldots >$
$< target\ linkName = "CtoD" / >$
$< /invoke >$
$< /flow >$

In a conventional client-server based workflow system, the execution of concurrent activity and control of the synchronization link are possible because the workflow server could control the state of all the branches nested in a concurrent activity.

However, in a multi-agent based open environment, the centralised coordinator is eliminated. Thus the only way for agents to coordinate with each other is again, through message passing, which means all the synchronisation links have to be controlled by message passing between agents as well. Figure 5 shows the algorithm that we use to turn all the synchronisation links defined in a $< flow >$ activity into SPPC messages. Using the above algorithm, a $< flow >$ activity can

procedure $translate\_flow(Flow, CL, SPPC)$
  input: $Flow$, $the\ BPEL4WS < switch > activity$
    CL, $a\ list\ that\ stores\ all\ un-assigned\ conditions$
  output: $SPPC, the\ SPPC\ clauses\ derived\ from\ given < flow > activity$
  $initiate\ a\ public\ list(L)$ $//public\ list\ can\ be\ accessed\ by\ any\ procedure$
  $extract\ all\ links\ defined\ in\ Flow\ and\ put\ them\ in\ L$
  for $(All\ the\ links(L_1)\ in\ L)$
    $scan\ the\ whole\ Flow\ and\ find\ out\ the\ activity(A)\ that\ defines\ the < source > of\ L_1$
    $scan\ the\ whole\ Flow\ and\ find\ out\ the\ activity(A_1)\ that\ defines\ the < target > of\ L_1$
    $replace\ A\ with\ ''A\ then\ A_2''$
    $in\ which\ A_2\ is\ a\ message\ sended\ from\ the\ receiver\ of\ A\ to\ sender\ of\ A_2$
    $replace\ A_2\ with\ ''A_3\ then\ A_2''$
    $in\ which\ A_3\ is\ a\ message\ sended\ from\ the\ receiver\ of\ A\ to\ sender\ of\ A_2$
  for $(All\ the\ branches(B)\ of\ FLow)$
    $translate\_structure\_activity(B, CL, SPPC_1)$
    $SPPC = SPPC\ par\ SPPC_1$

Fig. 5. Algorithm for deriving a SPPC model from a BPEL4WS $< flow >$ activity

be represented by a SPPC model. It should be noticed that when a SPPC model is translated into a LCC protocol, the SPPC messages generated for synchronisation links are only partially translated. The message that is derived for the "source" of a synchronisation link in SPPC is only used for the message sender's LCC protocol generation. In contrast, the message that is derived for the "target" of a synchronisation link in SPPC is only used for the message receiver's LCC protocol generation. Thus the algorithm for generating LCC protocols from SPPC models have to be revised to be used for dealing with SPPC models derived from BPEL4WS specifications.

– The $< while >$ construct allows us to indicate that an activity is to be repeated until a certain success criteria has been met. The notation that is used in SPPC for repeated execution of messages is the combination of $invoke(mid)$ and the SPPC message (M_1) that the $invoke$ points to. Whether the loop is executed is controlled by the precondition defined for the M_1. However, a BPEL4WS $< while >$ activity represented loop might start with a message passing activity, a computing activity or a structure activity and the execution of the its content is controlled by the $condition$ associated with it. Therefore, the translation from a $< while >$ activity to SPPC clauses involves two parts: the translation of $condition$ defined and the re-write of content of $< while >$. The principle of processing the $conditions$ defined for $< while >$ structure activity is complete same with that of $< switch >$. The re-writing of $< while >$ highly replies on the first and last elements nested in

it. Several re-write rules are thus defined for its different sorts of child elements.

$$(A_1\ then ... then\ A_i) \Rightarrow (E_1\ then ... then\ E_n) \quad if\ A_1 \Rightarrow E_1, A \Rightarrow E_i\ ((E_i \rightarrow generate\_invoke(E_1)) \Rightarrow E_n \quad (rule_8)$$

$$(A_1\ or/par ... or/par\ A_i) \Rightarrow (E_{i+1}\ or/par ... or/par\ E_{i+n}) \quad if\ A_1 \Rightarrow E_1, ..., A_i \Rightarrow E_i,\ ((E_1 \rightarrow generate\_invoke(E_1)) \Rightarrow E_{i+1}, ...,\ ((E_i \rightarrow generate\_invoke(E_i)) \Rightarrow E_{i+n} \quad (rule_9)$$

$$(E_i \rightarrow generate\_invoke(E_1)) \Rightarrow (E_i\ then\ E_2) \quad if\ generate\_invoke(E_1) \Rightarrow E_2) \quad (rule_{10})$$

$$generate\_invoke(E_1) \Rightarrow \begin{pmatrix} generate\_invoke(E_2), \\ or/par ... or/par \\ generate\_invoke(E_n) \end{pmatrix} \quad if\ the\ first\ element\ of\ E_1\ is\ (E_2\ or/par ... or/par\ E_n) \quad (rule_{11})$$

$$generate\_invoke(E_1) \Rightarrow invoke(E_2) \quad if\ the\ first\ element(E_2)\ of\ E_1\ is\ a\ single\ SPPC\ message \quad (rule_{12})$$

By applying the above re-write rules, a $< while >$ activity can be represented using SPPC notation. The algorithm for the $< while >$ activity's translation is given in figure 6

procedure $translate\_while(While,\ CL,\ SPPC)$
- **input:** $While$, the $BPEL4WS\ < switch >\ activity$
  CL, a list that stores all $un-assigned$ conditions
- **output:** $SPPC$, $SPPC$ clauses derived from given $< while >\ activity$
- extract the conditions defined for $While$ and put it at the end of CL
- extract the contents(C) of $While$
- $translated\_structure\_activity(C, CL, SPPC_1)$
- $invoke\_generator(SPPC_1, Invoke)$
- $loop\_generator(SPPC_1, Invoke, SPPC_2)$
- $SPPC = SPPC_2$

procedure $invoke\_generator(SPPC, Invoke)$
- **input:** $SPPC$, $SPPC$ clauses
- **output:** $Invoke$, a invoke or sets of invokes connected by $''or/par''$
- extract the first element(E) of $SPPC$
- **if** $(E$ is a $SPPC$ message$)$
  - $Invoke = invoke(E)$
- **else**
  - **for** $(each\ branch(B)\ of\ E$ connected by $''or/par'')$
    - $invoke\_generator(B, Invoke_1)$
    - $Invoke = Invoke\ or/par\ Invoke_1$

procedure $loop\_generator(SPPC_1, Invoke, SPPC_2)$
- **input:** $SPPC_1$, $SPPC$ clauses
  $Invoke$, a invoke or sets of invokes connected by $''or/par''$
- **output:** $SPPC_2$, $SPPC$ clauses that represent loop
- $SPPC_2 = SPPC_1$
- extract the last element(E) of $SPPC_2$
- **if** $(E$ is a $SPPC$ message$)$
  - replace it with $''E\ then\ Invoke''$
- **else**
  - **for** $(each\ branch(B)\ of\ E)$
    - $loop\_generator(SPPC_2, Invoke, SPPC_3)$
    - $SPPC_2 = SPPC_3$

Fig. 6. Algorithm for deriving a SPPC model from a BPEL4WS $< while >$ activity

4.4 Generating a MAS Interaction Protocol (LCC) From a SPPC Model

Although the main components of both SPPC and LCC are *messages and constraints* the same, they are built based on different concepts. With LCC, the MAS interaction

protocol is defined from the views of different agents where each agent has its own definitions, whereas with SPPC, the protocol model is built based on the messages passing, which means that 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 that we need to construct a 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*.

One important issues about SPPC modelling is that role dependency between SPPC clauses must be addressed. Role dependency means that two adjacent SPPC clauses connected by a $then$ operator at least need to have a same role defined in them as shown below:

$$
\begin{array}{l}
msg(MID, mb(...), sender(a(\textbf{Role}, ID)), receiver(a(Role_2, ID_1))) \\
then \\
msg(MID_1, mb(...), sender(a(Role_3, ID_3)), receiver(a(\textbf{Role}, ID))) \\
...
\end{array}
$$

In contrast, clauses shown below is not translatable, although it might be rational right:

$$
\begin{array}{l}
msg(MID, mb(...), sender(a(Role_1, ID_1)), receiver(a(Role_2, ID_2))) \\
then \\
msg(MID_1, mb(...), sender(a(Role_3, ID_3)), receiver(a(Role_4, ID_4))) \\
...
\end{array}
$$

To deal with such cases, the SPPC model is pre-processed before being translated to the LCC framework by correcting the role dependency. For example, after pre-processing, the above SPPC clauses become:

$$
\begin{array}{l}
msg(MID, mb(...), sender(a(Role_1, ID_1)), receiver(a(Role_2, ID_2))) \\
then \\
msg(MID_2, mb(\textbf{run\_this}), sender(a(Role_2, ID_2)), receiver(a(Role_3, ID_3))) \\
then \\
msg(MID_1, mb(...), sender(a(Role_3, ID_3)), receiver(a(Role_4, ID_4))) \\
...
\end{array}
$$

The message **run_this** defined above only serves as a connector. Thus, each message has role dependencies with its adjacent siblings. For different SPPC junctions, the pre-processing mechanism is different, the algorithm for pre-processing a SPPC model for later translating is given in figure 7

```
procedure seuqence_precessor(M1, M2, M3)
    inputs: M1, M2, two element conencted by a ''then'' operator
    outputs: M3, a processed model with all of its clauses role dependent on each other
    if (M1 is a message)&&(M2 is a message)
        if (M1 and M2 doesn't contain at least one same role)
            generate a new message(TM) using recevier of M1 as sender and sender of M2 as receiver
            M3 = M1 then TM then M2
        else
            M3 = M1 then M2
    else if (at least one of M1 and M2 is a or/par structure)
        or/par_precessor(M1, M2, M4)
        M3 = M4 then M2
    else if (M1 is a message)&&(M2 is a ''invoke(mid)'')
        fectch the SPPC message(M4) that identified by mid
            if (agents involved in M1 don't contain the sender of M4)
                insert a new message(M5) between M1 and invoke(mid)
                using receiver of M1 as its sender and sender of M4 as its receiver
                M3 = M1 then M5 then invoke(mid)
```

```
procedure or/par_precessor(M1, M2, M3)
    inputs: M1, M2, two SPPC element which can be a message or a or/par structure
    outputs: M3, a processed SPPC element with all of its clauses role dependent on each other
    if (M1 is a message)&&(M2 is a or/par structure)
        process_msg_or/par(M1, M2, M3)
    if (M1 is a or/par structure)&&(M2 is a message)
        process_or/par_msg(M1, M2, M3)
    if (M1 is a or/par structure)&&(M2 is a or/par structure)
```

```
procedure process_msg_or/par(M1, M2, M3)
    inputs: M1, M2, two SPPC elements which is a message and a or/par structure repectively
    outputs: M3, a processed SPPC element from M1 with all of its clauses role dependent on each other
    initiate a list(L)
    for (the first element(E) of each branch of M2)
        if (M1 and E doesn't contain at least one same role)
            generate a new message(TM) using recevier of M1 as sender and sender of E as receiver
            put TM in L
    M3 = M1
    while (L is not empty)
        fetch the first element(M5) in L
        M3 = M4 then M5
```

```
procedure process_or/par_msg(M1, M2, M3)
    inputs: M1, M2, two SPPC elements which is a or/par structure and a message repectively
    outputs: M3, a processed SPPC element from M1 with all of its clauses role dependent on each other
    for (each branch(B) of M1)
        pre − precessor(B, B1)
        retrieve the last element(E) of B1
        if (E is not a invoke(mid))&&(E and M2 doesn't contain at least one same role)
            generate a new message(TM) using recevier of E as sender and sender of M2 as receiver
            replace E in B1 with ''E then TM''
        put B1 in L
    fetch the first element(M5) in L
    M3 = M5
    while (L is not empty)
        fetch the first element(M6) in L
        M3 = M3 or/par M6
```

```
procedure process_or/par_or/par(M1, M2, M3)
    inputs: M1, M2, two SPPC elements which are two or/par structures
    outputs: M3, a processed SPPC element from M1 with all of its clauses role dependent on each other
    initiate two lists(L, L1)
    for (each branch(B) of M1)
        pre − precessor(B, B1)
        retrieve the last element(E) of B1
        for (the first element(E1) of each brach of M2)
            if (E is not a invoke(mid))&&(E and E1 doesn't contain at least one same role)
                generate a new message(TM) using recevier of E as sender and sender of E1 as receiver
                put TM in L
            M6 = E
        while (L is not empty)
            fetch the first element(M5) in L
            M6 = M6 then M5
        replace E with M6 in B1
        put B1 in L1
    fetch the first element(B2) in L1
    M3 = B2
    while (L1 is not empty)
        fetch the first element(B3) in L
        M3 = M3 or/par B3
```

Fig. 7. Algorithm For Pre-processing a SPPC model

The underlying principles of the above algorithm are:

$1>$ If two SPPC messages (A and B) are connected by "then" and they don't have at least one same role defined, then a new message (C) is inserted after A and before B (A then C then B) using run_this as its message body , the receiver of A as its sender and the sender of B as its receiver.

$2>$ If a SPPC message (A) and a SPPC or/par structure (B) are connected by "then", the first basic SPPC message(C_i) of each branch of B are compared with A and according to the roles defined in them, a list of new SPPC messages (M_n) are generated and put after A ($A\,then\,M_1\,then\,...\,then\,M_n$).

$3>$ If a SPPC or/par structure (A) and a SPPC message (B) are connected by "then", the last basic SPPC message(C_i) of each branch of A are compared with B and according to the roles defined in them, a list of new SPPC messages (M_n) are generated and put after C_i ($C_i\,then\,M_n$) and all the branches of A are also processed using the algorithm.

$4>$ If a SPPC or/par structure (A) and a SPPC or structure (B) are connected by "then", the last basic SPPC message(C_i) of each branch of A are compared with the first basic SPPC message (D_i) of each branch of B and according to the roles defined in them, for each C in C_i, a list of new SPPC messages (M_n) are generated and put after C ($C\,then\,M_1\,then\,...\,then\,M_n$).

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. For example, the following SPPC model shows a repeated message passing from $a(Role_1, ID_1)$ to $a(Role_2, ID_2)$:

$msg(mid_1, pre([]), mb(M_1), post([]), sender(a(Role_1, ID_1)), receiver(a(Role_2, ID_2)))$
$then$
$invoke(mid_1)$

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 LCC, the only way to represent loops is through use of recursion on a(Role,ID).

$$a(Role, ID) :: M \Rightarrow a(Role_1, ID_1)\,then\,a(Role, ID)$$

For example, the above LCC clause represents a repeated message sending from $a(Role, ID)$ to $a(Role_1, ID_1)$. In this way, everything defined for $a(Role, ID)$ is executed repeatedly. However, if we only want parts of the definition of an agent get executed in a loop manner, a new role must be invented for this purpose as follows:

$$a(Role, ID) :: M \Rightarrow a(Role_1, ID_1)\,then\,a(loop(Role), ID)$$

$$a(loop(Role), ID) :: M_1 \Rightarrow a(Role_2, ID_2)\,then\,a(loop(Role), ID)$$

What the above LCC protocol means is that agent $a(Role, ID)$ keeps sending a message M_1 to agent $a(Role_2, ID_2)$ after it sends a message M to agent $a(Role_1, ID_1)$. The role $a(loop(Role), ID)$ is purely defined for the purpose of repeated message sending of M_1.

In SPPC model, we use the combination of $invoke(mid)$ and $msg(mid, ...$ to represent loop, which has to be translated into a LCC compatible fashion. The loop processing algorithm in figure 8 shows how to pre-process the all the loops defined in a SPPC model. The algorithm for generating a LCC protocol from a processed SPPC

procedure $process\_loops(SM, SM_n)$
inputs: $SM, a\ original\ SPPC\ model$
outputs: $SM_n, a\ SPPC\ model\ that\ is\ with\ all\ of\ roles\ that\ are\ relative\ to\ loops\ replaced$
$find\ the\ first\ SPPC\ message(M_1)\ that\ leads\ to\ a\ loop\ and\ the\ invoke(mid)\ that\ points\ to\ it$
$extract\ the\ SPPC\ model(SM_1)\ between\ them$
$all\ the\ roles(a(R, ID))\ defined\ in\ SM_1\ have\ to\ be\ replaced\ with\ a(loop(R), ID)$
$process\_loops(SM_1, SM_n)$

Fig. 8. Algorithm For pre-processing all the loops defined in a SPPC model

model is shown in figure 9.

procedure $generator(SM, List)$
inputs: $SM, a\ SPPC\ model\ that\ is\ used\ to\ derive\ a\ LCC\ protocol$
outputs: $List, a\ LCC\ protocol\ list\ that\ stores\ generated\ LCC\ protocol\ from\ the\ given\ SPPC\ model$
$extract\ all\ the\ agents(a(R_{i,i\epsilon I}, ID_i))\ defined\ in\ the\ SM\ and\ put\ them\ in\ a\ list - \mathcal{L} = [a(R_i, ID_i)]$
while $(\mathcal{L}\ is\ not\ empty)$
$fetch\ the\ first\ element(a(R_i, ID))\ in\ \mathcal{L}$
$generate(a(R_i, ID), SM, LM_i, List_1)$
$put\ LM_i\ in\ List$
$List = merge\ List_1\ and\ List$

procedure $generate(a(R_i, ID), SM, LM_i, List)$
inputs: $a(R_i, ID), is\ an\ agent\ that\ we\ are\ going\ to\ generate\ a\ LCC\ protocol\ for$
$SM, a\ SPPC\ model\ that\ is\ used\ to\ derive\ a\ LCC\ protocol$
outputs: $LM_i, a\ LCC\ protocol\ that\ is\ generated\ from\ the\ given\ SPPC\ model\ for\ a(Role_i, ID)$
$List, a\ LCC\ protocol\ list\ that\ stores\ generated\ LCC\ protocol\ from\ the\ given\ SPPC\ model$
if $(SM\ is\ in\ the\ form\ of\ SM_i\ OP\ SM_{i+1})$
$generate(a(R_i, ID), SM_i, LM_n, List)$
$generate(a(R_i, ID), SM_{i+1}, LM_{n+1}, List)$
if $(LM_n = null\ \&\&\ LM_{n+1} \neq null)$
$LM_i = LM_{n+1}$
else if $(LM_n \neq null\ \&\&\ LM_{n+1} = null)$
$LM_i = LM_n$
else if $(LM_n \neq null\ \&\&\ LM_{n+1} \neq null)$
$LM_i = LM_n\ OP\ LM_{n+1}$
else if $(SM\ is\ a\ SPPC\ message(M_1))$
if $(M_1\ contains\ R_i)$
$LM_i = LM_n$
else if $(SM\ contains\ a\ a(loop(R_i), ID))$
$LM_i = a(loop(R_i), ID)$
$extract\ the\ invoke(mid)\ that\ points\ to\ M_1\ and\ extract\ the$
$SPPC\ model\ defined\ between\ invoke\ and\ M_1(SM_1)$
$generator(SM_1, LM_i, List)$

Fig. 9. Algorithm For Deriving a LCC protocol from a SPPC model

5 Agent Design

The agents that participate the interaction on a LCC protocol based MAS platform are dummy agents. The agents themselves don't need to make complex decision making

but simply follow what the LCC protocol asks them to do and perform some of the computational functions. Therefore, the design issues of this sort of agents are mainly about how to enable the agent conform to the protocols received and take proper actions accordingly. Figure 10 the conceptual layer of the internal structure of an agent.

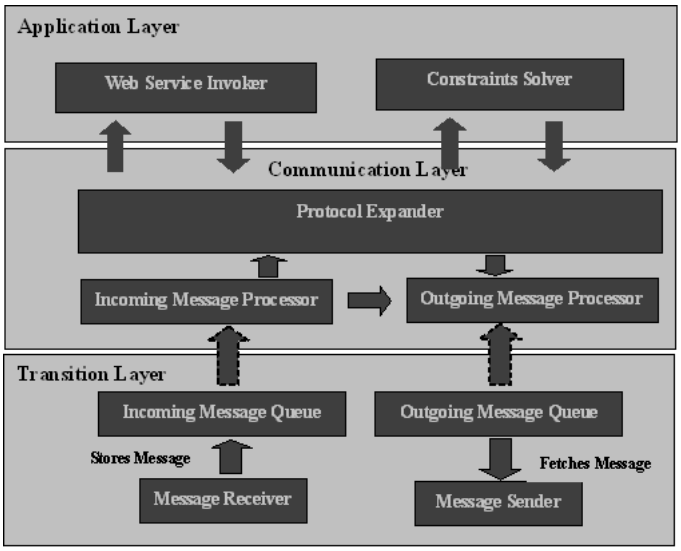


Fig. 10. Agent Internal Structure

- **Transition layer:** is responsible for the underlying message passing between different agents. It controls the messages' passing at the lowest level of our system. It retrieves the processed outgoing messages from communication layer and forwards the received messages to communication layer. The ”incoming message queue” and ”outgoing message queue” are used to store the message received and the messages that are going to be sent out. These two message queues are operated by ”message receiver” and ”message sender” in a FIFO manner and are used as a channel for the communication between the transition layer and communication layer. Once a ”message receiver” receive a message package from others, it puts it in in the end of ”incoming message queue” while ”message sender” fetches the first message in the ”outgoing message queue” and sends it out. The main task that ”message receiver” needs to perform is to filter transition level information of the received package such as if the received message is intended for it, or if the agent it represents for matches the agent's capability description attached in the message package. In contrast, ”message sender” adds transition layer to the outgoing message package according to the information derived from communication layer.
- **Communication layer**: is responsible for unpacking the received messages from transition layer and producing the outgoing messages according to the protocol attached with the received messages. ”Incoming message processor” is used to judge

whether the message that is fetched from ”incoming message queue” is the one that is required by the ”protocol expander” or not. If so, ”incoming message processor” passes it to ”protocol expander”. Otherwise, it put this message and everything that is attached with the message at the end of ”incoming message queue” for later processing. ”Outgoing message processor ” receives information from ” protocol expander” and puts them at the end of ”outgoing message queue”. Protocol expander” communicates with ”incoming message processor” and ”outgoing message processor” in following ways:

- If it doesn't hold a LCC protocol at the moment, it asks the ”incoming message processor” for message package. Once it receives it, it unpacks the message package, performs the required tasks, re-generates a new message package and sends it to ”outgoing message processor” using the following protocol expanding and re-write rules[Rob04]:

$$
\begin{array}{ll}
A :: B \xrightarrow{M_i, M_o, \mathcal{P}, O} A :: E & if\ B \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
A_1\ or\ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E & if\ \neg closed(A_2) \wedge A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
A_1\ or\ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E & if\ \neg closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
A_1\ then\ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E\ then\ A_2 & if\ A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
A_1\ then\ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} A_1\ then\ A_2 & if\ closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
A_1\ par\ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O_1 \cup O_2} E_1\ par\ E_2 & if\ A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O_1} E_1 \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O_2} E_2 \\
C \leftarrow A \Leftarrow M \xrightarrow{M_i, M_i - M \Leftarrow A, \mathcal{P}, \phi} c(M \Leftarrow A) & if\ (M \Leftarrow A) \in M_i \wedge satisfy(C) \\
M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_o, P, M \Rightarrow A} c(M \Rightarrow A) & if\ satisfied(C) \\
a(R, I) \leftarrow C \xrightarrow{M_i, M_o, P, \phi} a(R, I) :: B & if\ clause(\mathcal{P}, a(R, I) :: B) \wedge satisfied(C)
\end{array}
$$

- If it holds a protocol and is waiting for a message, it asks ”incoming message processor” for the message and blocks itself until it receives the required message.

During the process of protocol expansion, all the constraints involved are sent to ”constraints solver” at application layer for further processing.

”Outgoing message processor” simply forwards the message package that it receives from ”protocol expander” to ”outgoing message queue” currently. It is a place holder for outgoing message processing. For example, the message package may have priorities. In such case, the ”Outgoing message processor” is responsible for sorting the messages in ”outgoing message queue” accordingly.

– **Application layer**: is the place where the constraints defined in LCC protocol are solved. ”Web services invoker” takes care of all the issues of web services invocation including: invoking a web service according to the received messages; handling the returned message from invoked web service and converting them into agent's messages. ”Constraint solver” provides a container for executing the constraints that are requested by the ”protocol expander”. The way for solving the constraints might be attached in the LCC protocol or purely solved by the local methods.

6 Conclusion and Future Work

In this paper, we demonstrated our work on how to develop a pure decentralised workflow system that is deployed on protocol (LCC) based multi-agent platform using busi-

ness process models (BPEL4WS) as a starting point. Our work shows that architecture of conventional distributed workflow management system (client-server) can be changed without significantly affecting the methods, tools and models that exist in current workflow community. In addition, the re-use of existing results for new system development hugely reduces the amount of work that needs to be carried by other similar approaches, which conforms to the basic principle of software engineering strictly.

A language mapping technique is performed between business process modelling language (BPEL4WS) and IP (LCC) to generate the protocol that is used in MAS from given business process model. Since the gap between them is quite huge, we use a another modelling language (SPPC) as an intermediation. During the language mapping process, we found that although most of the main concepts from business process modelling language (BPEL4WS) and SPPC match, some particular notations from certain business process modelling language cannot be seamlessly represented by a another modelling language which is based on a different paradigm. For example, the computing activities defined at the end of a $< sequence >$ activity in BPEL4WS can not be easily translated into SPPC clauses as addressed earlier and also, the translation for the synchronisation links defined in $< flow >$ requires the revision of LCC protocol generation algorithm from SPPC. Such restriction means only some of BPEL4WS specifications (those conform to the language mapping rules) can be used for protocol based MAS development, which makes the approach discussed in this paper incomplete. Fortunately, these limitations can be overcome by revising the input business process models of our system minorly.

We are currently working on testing our system. We will then be able to determine various benefits and drawbacks of our approach when it is applied for real life application. Also, we are developing a new approach to bridge the gap between business process model and MAS interaction protocol seamlessly such that all the existing process models can be adopted for the new system.

References

[AWS00] Barbara Staudt Lerner Eric K. McCall Leon J. Osterweil Alexander Wise, Aaron G. Cass and Stanley M. Sutton. Using little-jil to coordinate agents in software engineering. In *Automated Software Engineering Conference (ASE 2000)*, pages 155–163, September 2000.

[BD99] T. Bauer and P. Dadam. Efficient distributed control of enterprise-wide and cross-enterprise workf. In *The Workshop Informatik99: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications,*, pages 25–32, Oct 1999.

[BPE03] Bpel4ws v1.1 specification. Technical report, May 2003.

[EP99] J. Eder and E. Panagos. Towards distributed workflow process management. In *Workshop on Cross-Organisational Workflow Management and Coordination*, Feb 1999.

[FF94] T Finin and R Fritzson. Kqml-a language and protocol for knowledge and information exchange. In *Proceeding of 13th International Distributed Artificial Intelligence Workshop*. July 1994.

[FIP00] Fipa acl message structure specification. Technical report, 2000.

[FK] G. J. Fakas and B. Karakostas. A peer to peer (p2p) architecture for dynamic workflow management. In *Journal of Information and Software Technology*.

[GAK95] R. Guenthoer D. Agrawal A. El. Abbadi G. Alonso, C. Mohan and M. Kamath. A persistent message-based architecture for distributed workflow management. In *IFIP WG8.1 Working Conference Decentralized Organizations,Trondheim*, August 1995.

[JGM98] J. Hosking J. Grundy, M. Apperley and W. Mugridge. A decentralised architecture for software process modelling and enactment. In *IEEE Internet Computing,*, Sep/Oct 1998.

[JVS04] P. Buhler J.M. Vidal and C. Stahl. Multiagent systems with workflows. In *IEEE Internet Computing*, volume 8(1), January/February 2004.

[J.Y04] J.Yan. *A Framework and Coordination Technologies for Peer-to-peer based Decentralised Workflow Systems*. PhD thesis, School of Information Technology, Swinburne University of Technology, 2004.

[LCB04] Dave Roberston L.Guo and Yun-Heh Chen-Burger. Business process model based multi-agent system development. In *Proceedings of The Second Workshop On Collaboration Agents: Autonomous Agents for Collaborative Environments*, September 2004.

[MRD03] S. Rinderle M. Reichert and P. Dadam. Adept workflow management system: Flexible support for enterprise-wide business processes (tool presentation). In *International Conference on Business Process Management (BPM'03)*, volume 2678, pages 371–379, June 2003.

[OWL01] Owl-s 1.0 release. Technical report, 2001.

[PHM99] S. Jablonski J. Neeb K. Stein P. Heinl, S. Horn and M.Teschke. A comprehensive approach to flexibility in workflow management systems. In *The International joint Conference on Work Activities Coordination and Collaboration (WACC99)*, pages 79–88, Feb 1999.

[PMG98] J. Weissenfels A. K. Dittrich P. Muth, D. Wodtke and G.Weikum. From centralised workflow specification to distributed workflow execution. In *Intelligent Information Systems - Special Issue on Workflow Management*, pages 159–184. Kluwer Academic Publishers, March 1998.

[Rob04] Dave Roberston. A lightweight method for corrdination of agent oriented web services. In *AAAI Spring Symposium on Sematic Web Services*, July 2004.

[Yan00] Y. Yang. An architecture and the related mechanisms for webbased global cooperative teamwork support, international. In *Journal of Computing and Informatics,*, pages 13–19, Sep/Oct 2000.