A Generic Multi-agent System Platform For Business Workflows Using Web Services Composition

Li Guo, David Robertson and Yun-Heh Chen-Burger
CISA, Informatics, The university of
Edinburgh, United Kingdom
L.Guo@sms.ed.ac.uk, {Jessicac,dr}@inf.ed.ac.uk

Abstract

This paper describes the development of a distributed multi-agent workflow[5] enactment mechanism from a BPEL4WS[1] specification. This work demonstrates that a multi-agent protocol (LCC protocol)[10] can be derived from a BPEL4WS specification to enable business workflows using web services[2] composition. The key difference between our system and other existing multiagent based web service composition systems is that our approach starts from a business process model which gives us an overview of the task being performed. All the participants in our system are generic agents that have no knowledge of any particular web service. The only knowledge that they have is how to execute the interaction protocol and invoke the web services properly. In addition, our approach makes it possible to avoid the single point of failure problem associated with a centralized workflow engine as it is based on decentralized computing paradigm.

1. Introduction

Composition of web services has received much interest as a means of supporting Business-To-Business or enterprise application integration. Currently, there are three approaches that address the problem: an industrial approach: BPEL4WS; a semantic web approach: OWL-S[6]; and an agent based approach. The first two approaches for web services composition are mainly based on static workflow technology. Using such methods, web services are described as activity/atomic process in a business process model. A Workflow engine is used to run the whole business process model, web services thus could be invoked as the business process executes.

Another approach proposed recently is employing multi-agent system for web services coordination[10, 13]. The basic architecture of this approach is shown in Figure 1 In this

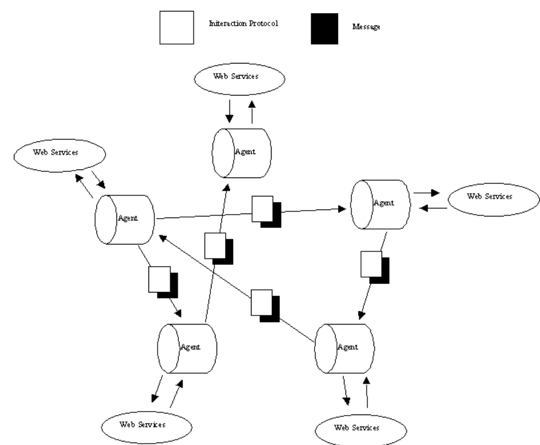

Figure 1: The infrastructure of MAS based web services composition system

infrastructure, the whole multi-agent system is wrapped in a single web service and each agent in the system is associated with a web service which contains the external behaviours for the participant (agent). The control flow logic is defined in the multi-agent system protocol and each agent decides what to do next by passing the protocol around. The Web Service Definition Language (WSDL)[3] specifications are automatically generated according to the MAS protocol and model checking is the main technique that is used to do verification and validation. A tool MagentA[4] has been developed based on the architecture shown in figure 1.

However, a shortcoming of current multi-agent based web service composition systems is that it is almost impossible to get the overall view of a business process describing by a dialogue protocol, since the protocol only specifies the message passing that takes place between different participants at implementation level, which mixes both

the business and technique requirements. Therefore the requirements are not the same as system design, which requires more effort on the validation and verification of system specification to make sure that it strictly consistent with requirements.

In this paper, we propose a generic MAS platform for business workflow using web services composition, addressing the problem above. In section 2, the framework of our platform is given as well as the necessary background introduction to BPEL4WS and a multiagent interaction protocol LCC[10]. The principles for syntax-directed translation from a BPEL4WS specification to a LCC protocol are presented in section 3 using a concrete example. A tool developed that is based on the translation principles is also shown in this section. In section 4, we explain the design of two types of agents that are used in our system and how they adopt the LCC protocol in order to coordinate. Some problems and possible future work are addressed in section 5.

2. A Generic MAS Platform for Business Workflow Using Web Services Composition

Due to the industry's increased focus on business process management and acceptance of BPEL4WS, vendors are producing new software tools for workflow design, specification, and enactment. Think of the workflow engine as an interpreter for the workflow specification: when the engine receives a workflow description, it enacts the workflow in a centralised manner. The engine manages each web service's interaction in the workflow, ensuring that all operations are performed as specified in the BPEL4WS description. The downside to this approach is that, although the engine can execute these invocations asynchronously (thus generating some degree of parallelism), as shown in figure 2, the process is still centralised, which means it suffers from the single point-of-failure weaknesses that plague centralized designs[9] and in certain environment, centralization is not quite possible, for example, in a mobile devices based environment. Since the central workflow engine

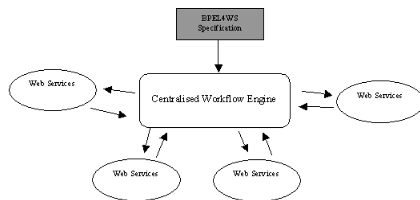

Figure 2: The infrastructure of conventional workflow based web services composition system

is only used to deal with the process logic and control the interactions between different web services based on the process logic, we can eliminate it by using multi-agent system.

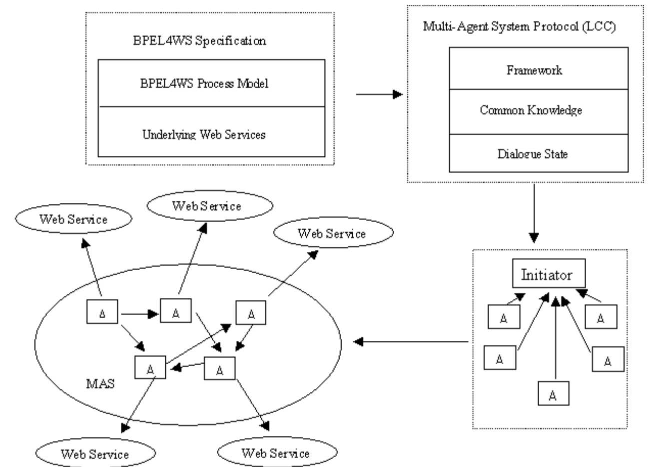

Figure 3: Revised Agents Based Architecture For Business Workflow Using Web Services

Figure 3 shows our framework for constructing a multiagent based workflow system that uses web services as external behaviours of all participants. There are basically four parts in this framework:

- BPEL4WS specification: The process model of BPEL4WS gives us a overview of the whole task and specifies how the underlying web services are used to do the real computation.
- Multi-agent system protocol (LCC): This is derived from BPEL4WS, in which the framework describes the process logic specified. The common knowledge defines all the data related information, such as information about used variables and web services etc. LCC protocol is used directly by the multiagent system, which performs the task defined in the BPEL4WS specification in a completely decentralized manner by eliminating the centralized workflow engine.
- Initiating Agent ($Initiator$): is an agent that reads the Multi-agent interaction protocol and instantiates the roles defined in the protocol to distributed workflow agents.
- Distributed Workflow Agent (A): is responsible for handling the process logic and interacting with actual web services. Each of the distributed workflow agent in our system represents a myRole/partnerRole that is defined in initial BPEL4WS specification.

2.1. BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS) is an XML-based language for describ-

ing workflow in a distributed environment using web services. With support from IBM and Microsoft, it has become the de facto standard for workflow description. 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. We will not explain the BPEL4WS notations in detail here due to the limited paper space.

2.2. Multiagent System Protocol (LCC)

The lightweight Coordination Calculus(LCC)[10] 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) 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:

$$\begin{array}{rcl} Framework & := & \{Clause, ...\} \\ Clause & := & Agent :: Def \\ Agent & := & a(Type, id) \end{array}$$

$$\begin{array}{rcl} Def & := & Agent | Message | Def\, then\, Def \\ & & | Def\, or\, Def | Def\, par\, Def \\ Message & := & M \Rightarrow Agent | M \Rightarrow Agent \leftarrow C \\ & & | M \Leftarrow Agent | M \Leftarrow Agent \leftarrow C \\ C & := & Term | C \wedge C | C \vee C \\ type & := & Term \\ id & := & Constant \\ Constant & := & Term \end{array}$$

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 an 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 an 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 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 [10].

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.

3. From BPEL4WS to LCC

BPEL4WS was not designed for a pure distributed workflow system (no centralized workflow engine) nor was it designed for multiagent enactment and, therefore, lacks explicit instructions about how agents should coordinate, while multiagent interaction protocol languages, like LCC, are more amenable to multiagent enactment. However, BPEL4WS is well recognized for its 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. In addition, it gives us the ability to use web services for the business workflows being described. Furthermore, using BPEL4WS, we can exploit existing workflows and tools. As such, we first develop methods for performing the BPEL4WS-to-multiagent-enactment mapping in order to build a multiagent platform for business workflows using web services properly and easily.

Our first task when mapping from BPEL4WS to a multiagent enactment is to make sure that the new enactment is functionally equivalent to the centralized version. The primary principles for the syntax-based mapping between some of main BPEL4WS notations to LCC framework are given below:

- The activity $receive$ in BPEL4WS means that a web service operation will not be invoked until certain requests (inputVariable of web service operations) arrive. The corresponding LCC dialogue for it from the point of view of $a(myRole, A_2)$ is

$$\begin{array}{l} a(myRole, A_2) :: inputVariable \Leftarrow a(partnerRole, A_1) \\ then \\ portType : operation(inputVariable) \Rightarrow a(myRole, A_2) \end{array}$$

  in which the $receive$ activity is represented by two LCC dialogues that are carried out in sequential order. The first clause means the receive of a request and the second clause means once a request is received, an operation should be invoked. The receiver of the second clause is $a(myRole, A_2)$ itself, which means that the message it receives causes a internal operation and the message is in the form of portType:operation(inputVariable), which is actually a signature of a web service operation. $PartnerRole$ and $myRole$ represent the role defined in partnerLinkType in BPEL4WS specification.

- The activity $reply$ simply means that a message is sent out to the customer. The LCC dialogue for the activity $reply$ defined in is

$$variable \Rightarrow a(partnerRole, A_1)$$

- The activity $invoke$ indicates a invocation on a web service operation and the receiving of a result from that operation if there is one between two participants.

$$\begin{array}{l} portType : operation(inputVariable) \\ \Rightarrow a(partnerRole, A_1) \leftarrow C_1 \, and \, C_2 \, and ... \\ then \\ C_1 \, and \, C_2 \, and ... \leftarrow portType : operation(inputVariable) : \\ outputVariable \Leftarrow a(partnerRole, A_1) \end{array}$$

  The first message indicates a invocation on a web service operation and second message is corresponding to the output of the web service that is invoked. Term C_i means the constraints associated with the dialogues, which is mainly derived from three kinds of BPEL4WS notations: $assign$, $condition$ and $links$.

- The activity $assign$ in BPEL4WS specification defines the internal variables assignation of BPEL4WS workflow engine and it gives the BPEL4WS computational ability. In LCC, constraints are the only place where we can do computation. When translating a $assign$ to a LCC constraint, the $assign$ is used as the constraint for the sender of activity $receive/invoke/reply$ that is immediate defined after it. For those $assigns$ that are putted in BPEL4WS specification randomly, we believe automatic translation is not possible.

- we also can use constraints in LCC to represent the synchronization links defined in BPEL4WS specification. $SourceLink$ is used as effect of receiving a message and $targetLink$ is used as a precondition of sending out a message in LCC. In order to distinguish the source and target for a same link. We use two predicates to do this: $create(sourceLink)$ and $exist(targetLink)$. $Create(sourceLink)$ creates a flag $sourceLink$ in the protocol and $exist(targetLink)$ checks if $targetLink$ flag exists in the protocol. For a $invoke$ activity that has a $sourceLink$ and a $targetLink$ defined as its child elements and a $outputVariable$, the LCC dialogues are

$$\begin{array}{l} portType : opertion(inputVariable) \\ \Rightarrow a(partnerRole, ID) \leftarrow exist(targetLink) \\ then \\ create(sourceLink) \leftarrow portType : \\ operation(inputVariable) : outputVariable \\ \Leftarrow a(partnerRole, ID) \end{array}$$

- The control structures $sequence$ and $flow$, they can easily be represented by LCC dialogues, with all its child elements connected by operator $then/par$ based on different roles. However, for the control structure $switch$ and $while$, things get more complex because of the existence of $condition$. We need to decide where to put the condition in as a constraint for a sending out message. In $switch$structure, each $case$ can possibly has four kinds of direct child elements: all the basic activities, $sequence$ structure, $switch$ structure, $flow$ structure and $while$ structure. For the first two cases, the $case\, condition$ can be used as the constraint for the basic activities or the constraint for the first element of $sequence$ structure if it is a basic activity since the temporal orders of properties are same and clear to see (in a sequential order). However, for $switch$ and $flow$ structure, the $case\, condition$ has to be used as constraint for all the first elements (if they are basic activities) defined in $switch/flow$.

- Translating a BPEL4WS $while$ structure to LCC dialogue is even more complex since in LCC, there is no direct notation corresponding to it. In LCC, the only way to represent loops is to use role's changing in the form as follows:

$$\begin{array}{l} a(Role, ID) :: M \Rightarrow a(Role_1, ID_1) \, then \\ \quad a(Role, ID) \leftarrow Condition \end{array}$$

$$\begin{array}{l} a(Role, ID) :: M \Rightarrow a(Role_1, ID_1) \, then \\ \quad a(Role(loop), ID) \leftarrow Condition \end{array}$$

$$\begin{array}{l} a(Role(loop), ID) :: M_1 \Rightarrow a(Role_2, ID_2) \, then \\ \quad a(Role(loop), ID) \leftarrow Condition \end{array}$$

  The first LCC clause represent a loop in which an agent $a(Role, ID)$ keeps sending message M to $a(Role_1, ID_1)$ as long as the $Condition$ is true. The second and third clauses means that an agent a(Role,ID) sends a message M to $a(Role_1, ID_1)$ and after that keeps sending message M_1 to $a(Role_2, ID_2)$. Thus, for the $while$ struc-

ture, the LCC dialogue is in the form of above clause.

- The another thing that should be noticed for a generic multiagent platform is, since there are several roles defined in LCC protocol, we need to indicate where the process should start (the agent that initiates the whole conversation process). We use a special keyword $Initiator$ in LCC to express it, for our example, this looks like:

$$Initiator :: a(Role, ID)$$

- Since the design goal of our approach is to build a generic multi-agent platform for business workflows using existing web services. The agent in the system should not be associated with any particular web service in advance. Therefore, all the information about where to invoke a web service defined BPEL4WS should be declared in common knowledge of LCC. Furthermore, all the variables' declarations and how they are shared by different roles also should be written in common knowledge as well as a data set that records the value of each variable when the workflow is being executed.

Due to the limited paper space, we can not illustrate all our mapping principles here. We use an example (shipping service process) to show how a LCC protocol can be derived from a BPEL4WS specification. The operation of the process is very simple, and is represented first in pseudo code. The formal specification of this example encoded in BPEL4WS follows. It has been simplified by removing attributes that do not help clarify the example[1].

Pseudo code for the shipping service process:

```
receive shipOrder
switch
  case shipComplete
      send shipNotice
  otherwise
    itemsShipped := 0
    while(itemsShipped < itemsTotal)
      itemsCount := opaque
      sendshipNotice
      itemsShipped = itemsShipped + itemsCount
```

Formal BPEL4WS Specification:

```
< process name = "shippingService" >
  < partnerLinks >
    < partnerLink name = "customer"
      partnerLinkType = "shippingLT"
      partnerRole = "shippingServiceCustomer"
      myRole = "shippingService" / >
  < /partnerLinks >
  < sequence >
    < receive partnerLink = "customer"
      portType = "shippingServicePT"
      operation = "shippingRequest"
      variable = "shipRequest" >
    < /receive >
    < switch >
      < case condition =
          "getVariableProperty('shipRequest',
                  'shipComplete')" >
        < sequence >
          < assign >
            < copy >
              < from variable = "shipRequest" / >
              < to variable = "shipNotice" / >
            < /copy >
          < /assign >
          < invoke partnerLink = "customer"
            portType = "shippingServiceCustomerPT"
            operation = "shippingNotice"
            inputVariable = "shipNotice" >
          < /invoke >
        < /sequence >
      < /case >
      < otherwise >
        < sequence >
          < assign >
            < copy >
              < from expression = "0" / >
              < to variable = "itemsShipped" / >
            < /copy >
          < /assign >
          < while condition =
              itemsShipped < itemsTotal >
            < sequence >
              < assign >
                < copy >
                  < from opaque = "yes" / >
                  < to variable = "shipNotice"
                    property = "itemsCount" / >
                < /copy >
              < /assign >
              < invoke partnerLink = "customer"
                portType =
                "shippingServiceCustomerPT"
                operation = "shippingNotice"
                inputVariable = "shipNotice" >
              < /invoke >
              < assign >
                < copy >
                  < from expression =
                  itemsShipped + itemsCount/ >
                  < to variable = "itemsShipped" / >
                < /copy >
              < /assign >
            < /sequence >
          < /while >
        < /sequence >
      < /otherwise >
    < /switch >
  < /sequence >
< /process >
```

Internally, the workflow definition coordinates the interaction of the two participants named, shippingServiceCustomer, and shippingService, which can be used to represent agents in our desired multiagent system. The complete LCC protocol framework for the example BPEL4WS process model is shown in figure 4.

4. Agent Design

There are two types of agents that enact the multiagent based workflow: initiating agent and distributed workflow agent. The initiating agent processes the LCC protocol and instantiates roles defined in the protocol to distributed workflow agents. Distributed workflow agent are the proactive proxies for the web services that they represent. One of the design goals for our distributed workflow enactment mechanism was to have the ability to instantiate the roles defined in our protocol with distributed workflow agent at run time.

1 The original scenario for this example is taken from [1]

$$Initiator :: a(shippingServiceCustomer, A_1)$$

$$a(shippingServiceCustomer(loop(m_4)), A_1) ::$$
$$(shippingServiceCustomerPT : shippingNotice(shipNotice) \Leftarrow a(shippingService(loop(m_4)), A_2))$$
$$then$$
$$\left(\begin{array}{l} a(shippingServiceCustomer(loop(m_4)), A_1) \leftarrow \\ (itemsShipped < itemsTotal)\, and\, itemsShipped = itemsShipped + itemsCount \end{array} \right) or\, null$$

$$a(shippingServiceCustomer, A_1) :: shipRequest \Rightarrow a(shippingService, A_2)\, then$$
$$\left(\begin{array}{l} shippingServiceCustomerPT : shippingNotice(shipNotice) \Leftarrow a(shippingService, A_2) \\ or \\ (a(shippingServiceCustomer(loop(m_4)), A_1) \leftarrow (itemsShipped = 0)\, and\, itemsShipped < itemsTotal)\, or\, null \end{array} \right)$$

$$a(shippingService(loop(m_4)), A_2) ::$$
$$\left(\begin{array}{l} shippingServiceCustomerPT : shippingNotice(shipNotice) \Rightarrow \\ a(shippingServiceCustomer(loop(m_4)), A_1) \leftarrow shipNotice = yes \end{array} \right)$$
$$then$$
$$\left(\begin{array}{l} a(shippingService(loop(m_4)), A_2) leftarrow \\ (itemsShipped < itemsTotal)\, and\, (itemsShipped = itemsShipped + itemsCount) \end{array} \right) ornull$$

$$a(shippingService, A_2) :: shipRequest <= a(shippingServiceCustomer, A_1)\, then$$
$$shippingServicePT : shippingRequest(shipRequest) \Rightarrow a(shippingService, A_2)\, then$$
$$\left(\begin{array}{l} shippingServiceCustomerPT : shippingNotice(shipNotice) \Rightarrow a(shippingServiceCustomer, A_1) \leftarrow \\ (shipNotice = itemsCount)\, and\, getVariableProperty(shipRequest, shipComplete) \end{array} \right)$$
$$or$$
$$(a(shippingService(loop(m_4)), A_2) \leftarrow (itemsShipped = 0)\, and\, (itemsShipped < itemsTotal))ornull).$$

Figure 4: LCC Protocol Framework for Shipping Service Process

Thus the distributed workflow agents are generic and are able to work with any given web services.

4.1. Initiating Agent

The initiating agent is used to read the LCC protocol and instantiates roles defined in the protocol to concrete distributed workflow agent. How it searches for the distributed workflow agent is beyond the scope this paper, so we simply assume that enough distributed workflow agents are available to initiating agent for our work. The basic working mechanism of initiating agent is it:

- reads a LCC protocol and create a unique protocol ID (PID) for this protocol. This makes it possible for distributed workflow agents to be involved in different workflow instances or even totally different types of workflows.

- processs the LCC protocol and finds out all roles ($a(Role_{i,})$) defined in the protocol and rewrite $a(Role_{i,})$ into the form of ($a(Role_i, PID : AID_i)$) in which, PID is the process instance ID and AID_i are distributed workflow agents IDs. Finally it passes the new LCC protocol with all roles instantiated with ($a(Role_i, PID : AID_i)$) to the $initiator$ defined in the protocol to start the workflow.

4.2. Distributed Workflow Agent

The distributed workflow agents are the proactive proxies for the passive web services they represent. Figure 5 reflects the basic architecture of the distributed workflow agents. The distributed workflow agents share the same functionalities (code base). On receiving different sorts of

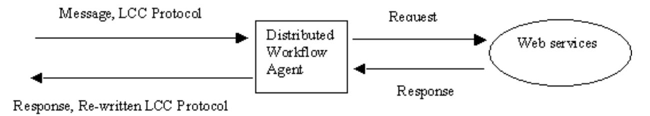

Figure 5: Basic Architecture for Distributed Workflow Agent

messages, the distributed workflow agent shows different behaviours.

- If the message is simply a request, $\mathcal{R}$, the distributed workflow agent looks it up in the protocol and find out the next move after $a(Role_i, PID : AID_i) \Leftarrow \mathcal{R}$
- If the message, $\mathcal{M}$, is in the form of $portType : operation(inputVariable)$, the distributed workflow agent: checks the common knowledge part in protocol, finds out where to invoke the operation and invokes the operation and then returns output to the sender of the message $\mathcal{M}$.
- If the message is in the form of $portType : operation(inputVariable) : outputVariable$, the distributed workflow agent updates the value of the

output of the operation by using the outputVariable in the received message.

4.3. Using LCC Protocols for the Coordination of Distributed Workflow Agents

To enable a distributed workflow agent to confirm a LCC protocol it is necessary to supply it with a way of unpacking any protocol it receives; finding the next moves that it is permitted to take; and updating the state of the protocol to describe the new state of dialogue. There are many ways of doing this but perhaps the most elegant way is by applying rewrite rules (more detailed re-write rules can be found in [10]) to expand the dialogues state. This works as follows:

- A distributed workflow agent receives from some other agents a message with an attached protocol, $\mathcal{P}$, of the form $protocol(S, F, K)$. The message is added to the set of messages currently under consideration by the agent-giving the message set M_i.
- The distributed workagent extracts from $\mathcal{P}$ the dialogue clause, C_i, determining its part of the dialogue.
- Applying the rewrite rules in [10] to give an expression of C_i in terms of protocol $\mathcal{P}$ in response to the set of received messages, M_i, producing: a new dialogue clause C_n; an output message set O_n and remaining unprocess messages M_n (a subset of M_i). These are produced by applying the protocol rewrite rules exhaustively to produce the sequence:
  $$\langle\ C_i \xrightarrow{M_i, M_{i+1}, \mathcal{P}, O_i} C_{i+1}, C_{i+1} \xrightarrow{M_{i+1}, M_{i+2}, \mathcal{P}, O_{i+1}} C_{i+2}, \ldots, C_{n-1} \xrightarrow{M_{n-1}, M_n, \mathcal{P}, O_n} C_n \rangle$$
- The original clause, C_i, is then replaced in $\mathcal{P}$ by C_n to produce the new protocol, $\mathcal{P}_n$
- The distributed workflow agent can then send the messages in set O_n, each accompanied by a copy of the new protocol $\mathcal{P}_n$.

5. Conclusion and Future Work

In this paper, we have represented a novel technique for constructing business workflows using existing web services composition on a generic multiagent system platform, which particularly suits the inter-operations among enterprises. By using our approach, a BPEL4WS specification can be used as a base for constructing a web services based multiagent system. In such a system, all the real operations are carried by web services that are associated with distributed workflow agents. The distributed workflow agents only deal with the flow control logic that is expressed in BPEL4WS specification and have no knowledge about any particular web service until they receive a MAS protocol which is derived from a BPEL4WS specification. Since all the agents are generic, it is possible for them to change their roles for different web services, which reduces vulnerability to single point of failure(as usually happens in a centralized environment).

Our next step is try to solve the transactional problems in a pure decentralized environment by using multi-agent coordination.

References

[1] *Business Process Execution Lanuage For Web Services specification*, http://www-128.ibm.com/developerworks/library/ws-bpel/.

[2] *W3C. Web Services reference*, http://www.w3.org/2002/ws/.

[3] *Web Service Definition Language references* http://www.w3.org/TR/wsdl.

[4] *MagentA*, http://homepages.inf.ed.ac.uk/cdw/magenta.html.

[5] *The Workflow Management Coalition*, http://www.wfmc.org/.

[6] *OWL-S*, http://www.daml.org/services/owl-s/1.0/owl-s.html.

[7] *IBM. BPWS4J*, http://www.alphaworks.ibm.com/tech/bpws4j.

[8] *Extensible Markup Language (XML)*, http://www.w3.org/XML/

[9] J.M. Vidal, P. Buhler, and C. Stahl. *Multiagent systems with workflows.* IEEE Internet Computing, 8(1):76-82, January/February 2004.

[10] D. Roberston, *A Lightweight Method for Coordination of Agent Oriented Web Services*, Proceedings of AAAI Spring Symposium on Sematic Web Services, 2004.

[11] L.Guo, Dave Roberston and Yun-Heh Chen-Burger. *"Business Process Model Based Multi-agent System Development"*. Proceedings of The Second Workshop On Collaboration Agents: Autonomous Agents for Collaborative Environments.

[12] P. A. Buhler, J. M. Vidal, H. Verhagen *Adaptive Workflow = Web Services+Agents*, Proceeding of IEEE International Conference on Web Services 2003.

[13] C. D. Walton *Model Checking Multi-Agent Web Services*, Proceeding of AAAI Symposium of Semantic Web Services 2004.