Enacting a Decentralised Workflow Management System on a Multi-agent Platform

Li Guo

Doctor of Philosophy

Artificial Intelligence Applications Institute

School of Informatics

University of Edinburgh

2006

Abstract

This thesis presents sets of technologies for enacting multi-agent based decentralised workflow systems. Its purpose is to tackle some of the existing problems in the conventional workflow research from the system architectural and engineering point of view. Some of the problems addressed at the beginning of this thesis have affected the wide deployment of workflow management system in an open environment (internet). This thesis argues that most of these problems are caused by the huge conceptual gap and design rationale between high level application requirements and low level system design/implementation. Specifically, it is argued that the conventional system architecture of workflow management system (client-server) could be replaced by a multi-agent based platform which is more open, collaborative and can better reflect workflow's distributed features in the open environment.

Combining existing workflow design rationale and multi-agent computing technology, a multi-agent based decentralised workflow approach is proposed in this thesis. The architecture of the intended system removes both the centralised data storage and the centralised workflow engine from the system. To achieve this goal, approaches that bridge the gap between business process modelling and multi-agent interaction protocol production are proposed using three different techniques (namely functional properties based specifications verification, syntax based language mapping and interpretation based communication) according to the different types of business process models used. Based on such approaches, the mechanisms for decentralised process execution are explored. Moreover, our system is also able to be extended to support incompletely/particially specified processes in a distributed manner. The approach for handling such incomplete/particially specified processes at run-time are presented in this thesis

The main contributions of this research is to provide approaches for enabling decentralised workflow systems in an open environment based on a multi-agent platform without changing the conventional workflow design rationale and with maximum use of existing process models and tools.

Acknowledgements

I sincerely express my deepest gratitude to my supervisor, Dr. Dave Robertson and Dr. Yun-Huh Chen-Burger, for their seasoned and valuable supervision and continuous encouragement throughout the course of this work, and for their careful reading and appraisal of drafts of this thesis. Without their consistent support, I would not have been able to complete my research and this manuscript.

I thank the University of Edinburgh and the School of Informatics for offering me full research facilities throughout my doctoral program. I also thank the Centre for Intelligent Systems and their Applications of School of Informatics for research publication funding support and for providing me with financial support to attend conferences.

My thanks also go to staff members, research students and research assistants at SSP and CISA for their help, suggestions, friendship and encouragement, in particular, Adam Barker, Paolo Besana, and Jarred McGinnis in Office 4.15. Last but not least, I am deeply grateful to my parents for their love, understanding, patience, encouragement, sacrifice and help.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Li Guo*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the limitations of conventional workflow management systems based on the dominant client-server distributed system architecture. The research reported in this thesis develops a new framework and coordination technologies for a decentralised workflow systems based on a multi-agent platform, rather than a conventional client-server based distributed system architecture[AWS02]. An innovative workflow management system development approach based on a multi-agent/peer-to-peer architecture, is presented in this thesis. A system prototype implementation based on Sun Microsystems JXTA[JXT] is developed for demonstrating purposes. Moreover, the corresponding system mechanisms to support complete and incomplete processes are designed.

The background, motivations and key issues of this research are introduced in this chapter. First, an introduction to workflow management is given in Section 1.1. Section 1.2 then addresses the key issues of this research. At last, Section 1.3 presents an structure overview of this thesis.

1.1   Introduction to Workflow Management

At the heart of any organisations is a more-or-less formalised set of processes, which reflects the way in organisations coordinate and organise work activities, information and knowledge to produce products or to provide services [LL02]. Typical processes' examples are credit card application processing process, student registration process and so on, for example. Support for processes has become crucial to the success of the organisation as a whole [JBR99]. Over the past years of process support research, paradigms have changed from (hard-wired) office automation systems to work-

flow management systems. With the success use of internet, Workflow Management (WfM), as an enabling technology for Business Process Management (BPM), is widely used by different organisations and becoming an important part of them.

A workflow represents the operational/functional features of a underlying process of an organisation. It reflects the order of activities and the performers (roles) of them; the information flow that is used to support all the activities defined within the process; and the monitoring and reporting mechanisms that measure and control them[Yan00]. A workflow is formally defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules to achieve, or contribute to, an overall business goal"[Coa99]. Although workflow may be constructed manually, in real life, most workflows are better constructed using a computational system to support the process automation. Such computer based systems, which also are called Workflow Management Systems (WfMSs), are designed to improve the effectiveness and efficiencies of the underlying processes by supporting the automation of the following aspects of the workflow [Coa99]:

- performing work in a proper sequence.

- providing sufficient access to the resources required by the individual work performers, and

- monitoring all aspects of the processes' execution.

To achieve these, a workflow management system "consists of software components to store and interpret process definitions, create and manage workflow instances as they are executed, and control their interaction with workflow participants and applications" [Coa99]. At the highest level, all workflow management systems may be classified as providing support in two functional categories[Yan00]:

- the design-time functions, considered with defining and representing the workflow process and its constituent tasks, and data storage issues, and

- the executing-time functions, concerned with creating and managing the workflow instances in an operational environment.

Currently, workflow management systems have undoubtedly become the kernel of organisations, as they are capable of integrating various kinds of resources, such as heterogeneous systems, existing applications, human beings, and so on [Moh98,

Sch99]. It is observed that the proper use of workflow management systems can help to make procedures more efficient, to reduce costs and flow times, and to increase the quality of service and productivity.

From the point of views of both research and practical areas, workflow management has been one of the most important areas of interest since its appearing. Many theoretical approaches and research prototypes have been presented and also lots of contributions have been published [DGS95, JB96, FCP96, Moh97, vdAvH02, Fis02, DGCIS95]. A huge number of commercial workflow management products are available, such as ActionWorkflow (Action Technologies Inc., http://www.actiontech.com), FlowMark (IBM, http://www.ibm.com), InConcert (InConcert Inc., http://www.inconcertsw.com), METEOR (Infocosm Inc., http://www.infocosm.com), Visual WorkFlo (FileNet, http://www.filenet.com), and so on[Yan02b].

1.2   Key Issues of This Thesis

Although workflow research and practice is quite mature, some problems are still recognised. The state-of-the art in workflow management has been determined by the functionality provided in workflow systems so far[GAM97]. Problems like system performance, reliability, scalability, system openness and incomplete process support are not considered enough in the development of existing workflow systems [DGS95, AS96, JGM98, Yan02b]. Therefore, workflow systems often suffer from deficiencies in these areas, such as bad system performance, one point failure of system, unsatisfactory system openness, and lack of incomplete process support. Some significant work has been done [GAK95, EGD97, ASHT98, SJS02, YY01, LMCM01] to address some of these problems. In addition, from practical point of view, as E-commerce becomes more and more complex, the collaboration between different enterprises to provide appropriate services is required. During the collaboration, each participant should be able to join the collaboration, contribute its services to it and quit from it on the bases of on its own needs. Some standards, such as BPEL4WS [BPE03, OWL01] has been proposed for this purpose. However, with current approaches which are mainly based on the conventional workflow architecture (client-server), open scale collaboration can not easily be achieved and most importantly, the participants of the collaboration lose their own initiative, which means they can only be invoked and required for certain services passively by the server as shown below:

As many researchers have noted, most of the above problems are mainly, if not
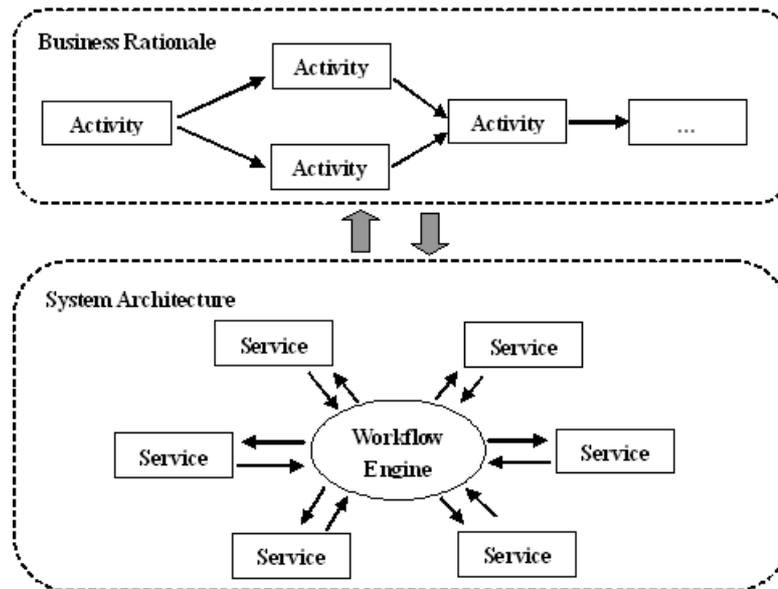
Figure 1.1: Conventional system architecture for business workflows

completely, caused by the adoption of a client-server architecture in most conventional workflow management systems [JGM98, Coo02]. Hence, in order to tackle these problems properly, a centralised system architecture based on client-server is expected to be replaced by an open, collaborative, and decentralised architecture that can reflect increasingly open and distributed features of current workflow more naturally. This thesis aims at addressing the above problems fundamentally from a point of view of system architecture without affecting the upper level business rationale, as shown in Figure 1.1, using the multi-agent based system architecture, which is depicted in Figure 1.2. Although some work has been done in this area [SPJC97, FK, Yan02b], this thesis proposes some new solutions to add to and improve on the existing approaches.

A fundamental contribution of this thesis is to adopt new system architecture (multi-agent) for deploying distributed workflow system in the open environment without changing existing business rationales that are used by business users. This research work mainly focuses on the issues of workflow as addressed in Section 1.1, i.e., performing pre-defined tasks in the proper sequence in a decentralised environment through coordination. Particularly, this thesis starts with the discussion of a coordination mechanism using process models for complete processes on multi-agent platforms. The proposed approaches mainly try to address problems related to performance, reliability, scalability, and system openness with existing design rationales, tools and models of workflow system kept untouched. This research is then extended to discuss the
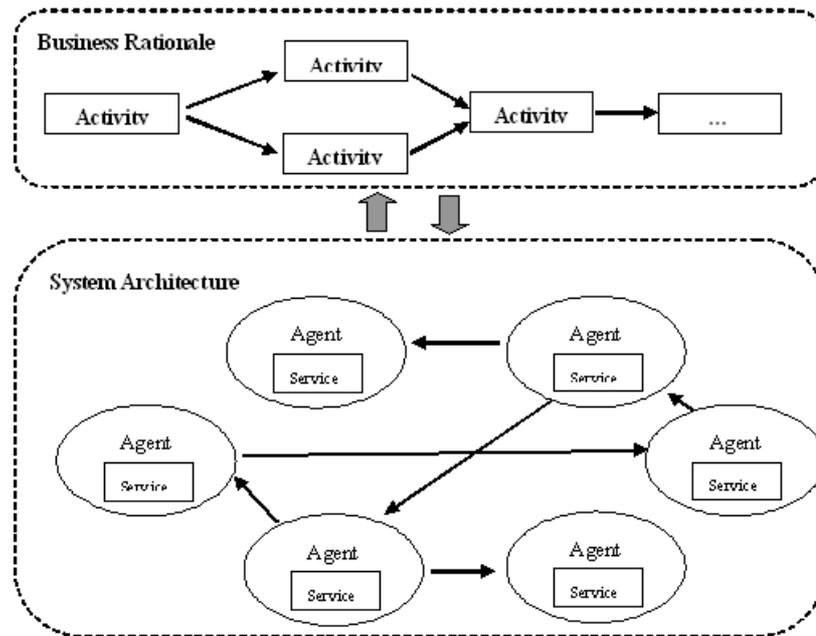
Figure 1.2: Multi-agent based system architecture for business workflows

technologies that support incomplete processes under the proposed multi-agent decentralised architecture. The proposed approaches are expected to be useful for adoption of workflow systems in some new, non-conventional application domains. The significance result of our research is to provide a better solution to the existing problems of conventional workflow research as described above. This can be considered as a paradigm change because the multi-agent technology provides a new platform.

1.3 Overview of This Thesis

In Chapter 2, research problems existing in conventional workflow system are analysed and discussed in detail. Some of the main related work is also reviewed, including workflow management approaches under aspects of client-server based distribution, decentralised workflow approaches based on other computing technologies, and research regarding incomplete process support. Chapter 2 also analyses the requirements for workflow management based on a multi-agent architecture.

The design of of a framework for bridging formalised high level business process models down to MAS interaction protocols (IPs) is illustrated in chapter 3. Checks of temporal orders between functional properties are performed to make sure that the functional requirements defined in the formalised high level business process model

are achieved by system level specification (MAS interaction protocols), using linear temporal logic as an intermediate specification.

Chapter 4 describes the language mapping technique that we perform on the translation between an executable business process modelling language BPEL4WS[BPE03] and a multi-agent interaction protocol description language LCC[Rob04]. The MAS interaction protocols thus can be generated automatically from given business process models and they, later on, can be used directly for multi-agent based workflow enactment.

Chapter 5 presents a novel approach for using an executable business process model directly for guiding multi-agent coordination. With this approach, no extra work is needed once the business process models are constructed. The interaction protocol is not used directly to guide the agents' coordination but used as a language interpreter to tell agents during their coordination what to do according to the attached business process model.

Chapter 6 further extends the approach that is proposed in chapter 5 to enable multi-agent based workflow management system for incomplete processes. How to handle incomplete processes incrementally at run-time is presented for different sorts of incomplete activities.

In chapter 7, three case studies are given to prove the soundness of our system and corresponding mechanisms from the point of view of both a conventional workflow domain (complete process) and a non-conventional workflow domain (incomplete process).

The prototype implementations for the ideas presented in each of the chapters are also given at the end of each.

In chapter 8, we discuss the advantages and disadvantages of the proposed approaches, and the suitable application domains of the multi-agent based decentralised workflow management system proposed in this thesis. The potential tradeoffs of the proposed approaches are also discussed in this chapter.

The last chapter, Chapter 9, summarises the ideas discussed in this thesis, the main contributions of this research, and future research work.

Chapter 2

Literature Review and Problem Analysis

In this chapter, first, an analysis of the research problems existing in conventional workflow is given in section 2.1. Then section 2.2 introduces conventional distributed workflow approaches. Section 2.3 introduces some earlier research on decentralised workflow based on other computing technologies. Section 2.4 introduces research work related to incomplete process support. Finally, justification is given for choosing a multi-agent based decentralised workflow system as the most effective line to follow, and requirements for designing a multi-agent based decentralised workflow system are given.

2.1 Research Problems Analysis

Workflow processes within organisations often involve a vast number of resources, people and tools that are distributed over a wide geographic area. Workflow management systems are used to coordinate these elements automatically. Thus, in order to suit the nature of the application environment and the technology adopted, workflow applications are becoming distributed [GAK95, JGM98, Yan00, CBL05]. Problems remain unsolved in current research of distributed workflow system however.

These problems are mainly categorised into two groups as addressed in[Yan00]. In the first group are those directly related to the centralised system architecture, i.e, bad performance, vulnerability to failures, poor scalability, user restrictions, and unsatisfactory system openness. The second group concerns flexibility, i.e, support for the incomplete workflow process.

Several problems of centralisation are as follows:

- **Performance:** A workflow management system based on a client-server architecture heavily relies on a centralised data server for data storage and a centralised workflow server for control the process logic while the computing capability at the client end is seldom used. Such a workflow system is heavy-weight. When the system load increases, for example, when many process instances are executed at the same moment, the centralised server can be overloaded with heavy computation and mass amount of communication between participants, hence becoming a potential bottleneck of the whole system. System performance can be hugely affected under such circumstances.

- **Robustness:** A workflow system based on a client-server architecture lacks robustness. The centralised server is normally regarded as a single point failure of the system. The whole system can become completely dead if the server is unavailable due to hardware or software failures. Although server backup can be used to improve reliability, it inevitably increases the complexity of the already sophisticated workflow system since additional mechanisms have to be designed to backup all the states of a running workflow server on an another server.

- **Openness:** A workflow system based on a client-server architecture offers limited capability of openness due to the existence of the centralised workflow server. As a consequence, lack of openness prevents a workflow system from coping with the changing workflow environment. In addition, it raises difficulties in system extension. Any extension to the system, (e.g., joining of new participants) requires modifying and updating the centralised workflow server, which is inconvenient and inefficient.

Moreover, as web services and Grid services become more popular as the reference model for business resources, workflow plays a powerful role in composing individual services into complex ones. However, a client-server architecture is not suitable for such applications where workflow technology is used together 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 more easily.

Besides the above problems that are caused by the centralised system architecture, lack of ability to support incomplete processes is also a major problem for conven-

tional workflow management system. Workflow research was initially founded on two assumptions.

- First, a workflow process model is accomplished completely at design-time before the execution of workflow instances.

- Second, the running instances of process have to remain unchanged during their execution.

These two assumptions were reasonable originally since workflow technology was traditionally used in those domains which were classified by pre-determined, routine based processes. These processes are functionally predictable and repetitive. Recently, the latter assumption has been undermined, with the argument that workflow processes are subject to both inside and outside changes [AJ00]. As a result, points of dynamic workflow change, exception handling and workflow adaptation (some of the today's major research topics) have been addressed widely [AJ00, HA00, SLS99]. More recently, the former assumption that workflow processes are always modelled completely at build time has also been challenged [Wes02, SSO01]. "There is substantial evidence of workflow processes for which trying to define (or prescribe) every step may compromise the process goal. In many cases, the work practices themselves would not fit into a prescriptive framework and introducing a technology which imposes it would result in decreased productivity."[Yan00]. In other words, the processes do not exclusively belong to the pre-defined class of processes, which are generally not repetitive (depending on instance data), and either represent an administrative level of complexity or a very high level complexity which is hard to be fully modelled) [SSO01]. Certain application areas such as health care, insurance claim and customer relation management have increased possibility of workflow processes that have both ad-hoc and prescriptive process requirements. However, most of today's workflow management approaches lack the capability to support the processes for such application domains.

Research on multi-agent systems emerged as a new area in the early 1990's. The computing paradigm of multi-agent systems (MAS) has its origin in both distributed artificial intelligence (DAI) and object-oriented distributed systems. There is no consensus on the definition of software agents or of agency. However, the prevailing opinion is that an agent may exhibit three important general characteristics: autonomy, adaptation, and cooperation. Cooperation and coordination between agents is probably the most important feature of multi-agent systems. Unlike 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 emergent collective behaviour beyond individual agents. From software engineering point of view, the approach of MAS 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 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 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 certain mechanisms to ensure that the agents in MAS always behave properly and effectively towards the final goal. An interaction protocol is used for this purpose, which defines the sequences of message passing that the agents must follow as well as the constraints on message passing. Many interaction protocol languages have been defined for describing such protocols, such as, FIPA-ACL[FIP00], lightweight coordination calculus (LCC)[Rob04]. However, an obvious problem is that it is almost impossible to get the overall view of the business processes described by a dialogue protocol since the protocol only specifies the message passing between different participants at implementation level. For certain MAS based application areas (for instance, auction systems) interaction protocol oriented approaches have few disadvantages. But for other application areas; for example, workflow management systems, the 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 being hidden. Business process modelling languages are in contrast recognised for their value in organising and describing a complex, informal domain in a more precise semi-formal structure that is intended for more objective understanding and analysis. Because of these advantages, they have been widely used in conventional workflow management systems and many mature techniques and tools been developed for supporting business process model based workflow system development.

However, business process modelling languages are designed specifically for con-

ventional workflow management architecture, they can not easily be adapted for new system architectures like multi-agent systems. Therefore, when building a MAS based workflow management systems, almost all the existing techniques and tools for supporting conventional workflow management system development are wasted as well as business process models that are described in 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, the business process modelling languages used in workflow management systems sometimes are built with specific features. For instance, BPEL4WS[BPE03] is designed for a web services based distributed workflow system. By using such specific languages, new platforms can be used to support existing technologies.

2.2 Conventional distributed workflow approaches

Many research efforts have been undertaken on the topic of distributed workflow in conventional workflow environment. The importance of connecting "workflow management" with "distribution" has been addressed in [PMG98, EP99, PHM99]. Some conceptual approaches and research prototypes have been proposed and developed, which try to solve these problems by making conventional distributed workflow management systems more sophisticated.

2.2.1 METUFlow

METUFlow[EGD97] is a distributed workflow management system developed at Middle East Technical University. This project tries to design a distributed workflow service which involves several schedulers on different nodes of a network. Each scheduler executes parts of process instances. Therefore, such a system could well fit to the distributed environments, enhance robustness and improve system performance.

The approach proposed in METUFlow is based on the observation that controlling the occurrence of events provides the coordination of the tasks. This means dependencies between tasks are represented by event dependencies. To enable distributed execution of workflow computations, each event in METUFlow is made responsible for controlling its execution to decide on the right time to occur. Required information

for this operation is treated as a guard, which is a temporal expression defined on an event. Occurrences of events are permitted only if their guards are true. Thus, each node in the process tree is implemented as a CORBA object with an interface for the guard handler to receive and send messages. Workflow is deployed by these CORBA objects, with computed guards controlling distributed execution.[Yan00]

2.2.2 ADEPT

ADEPT stands for Application Development based on Encapsulated pre-modelled Process Templates. This project started in 1996 at University of Ulm and the goal of this project is to build the next generation workflow technology for enterprise-wide and cross-enterprise workflow management [MRD03]. One important aspect of the ADEPT project is to perform distributed workflow control in order to avoid over-loading of the workflow servers and of the communication network. To address the problems, ADEPT reduces the system 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 but by several shared ones. When performing such a migration, a description of the instance states is transferred between different servers. This description contains information about activity states as well as workflow relevant data. To avoid unnecessary message transfer between servers, ADEPT allows control of concurrent execution of workflow instances independently from each other. Its communication actions can be further enhanced if variable server assignment expressions are used. These expressions could be decided at design-time, allowing the selection of a suitable workflow server to keep most of the communication inside it, and require very limited additional effort at run-time. Moreover, ADEPT supports both static and variable server assignments [BD99]. The former means appropriate workflow servers are picked for various partitions of a workflow definition. In contrast, assignment of variable server allows for dynamic workflow server assignment in real time, which may improve the system performance hugely.

2.2.3 Web service based approaches

As web services become more and more popular and widely used as organisational interfaces, several approaches have been proposed to deploy web services 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 languages for web services (BPEL4WS)[BPE03] and OWL-S[OWL01].

2.2.3.1 Business Process Execution Language for Web Services (BPEL4WS)

Business Process Execution Language for Web Services[BPE03] provides a means to formally specify business processes and interaction protocols.

BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL4WS defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

2.2.3.2 OWL-S

OWL-S is a OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in an unambiguous, computer-intepretable form. OWL-S markup of Web services facilitates the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation. Following the layered approach to markup language development, the current version of OWL-S builds on the Ontology Web Language (OWL)[OWL04] recommendation produced by the Web-Ontology Working Group at the World Wide Web Consortium. OWL-S consists of four main classes that specific services should instantiate. (Alternatively, service providers may create subclasses of the OWL-S classes and instantiate those instead).

- Service, with some basic concepts that tie the parts of an OWL-S service description together and holds a textual description of the service.

- Profile, which has properties used to describe what the service does; what it provides clients, and what it requires of them.

- Process, which has properties used to describe how the service works, i.e. what happens when the service is used. Services can be described as a collection of atomic or composite processes, which can be connected together in various ways, and the data and control flow can be specified.

- Grounding, with properties used to specify how the service is activated, including details on communication protocols, message formats, port numbers, etc.

2.2.4 Discussion

The above approaches put some distribution features on workflow systems and do bring benefits such as improved system performance, increased robustness and enhanced openness as they claimed. However, these approaches mainly address the concept of distribution instead of decentralisation. A common characteristic of these approaches is that they are still based on and confined by the client-server architecture. Therefore, these approaches either tackle the problems partly, or need complicated languages and/or complex algorithms to be defined. The remaining centralised services like centralised process instantiation and work assignment also make them relatively inflexible in some application domains. In addition, the openness of system are barely concerned. As a result, the problems that are relevant to the centralised distributed system architecture have not been or cannot be addressed completely if the whole workflow system is built on a client-server architecture.

2.3 Workflow approaches based on multi-agent/peer-to-peer platforms

The appearance of novel computing technologies such as multi-agent system have provided new platforms to support process management, while conventional distributed workflow approaches fail to properly address the problems in the first group described in Section 2.1, some limited research effort has been put into using these collaborative and decentralised platform to support workflow management systems.

2.3.1 Little-Jil

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. Little-JIL provides a rich set of control structures while relying on separate systems for support in areas such

as resource, artifact, and agenda management. The second is that processes can be executed by agents who know how to perform their tasks but can benefit from coordination support. Accordingly, each step in Little-JIL is assigned to an execution agent (human or automated): agents are responsible for initiating steps and performing the work associated with them. This approach has so far proven effective in allowing us to clearly and concisely express the agent coordination aspects of workflow.

The main features of the language and their justifications are the following:

- Four non-leaf step kinds provide control flow. These four kinds are "sequential", "parallel", "try" and "choice".

- Requisites are a mechanism to add checks before and after a step is executed to ensure that all of the conditions needed to begin a step are satisfied and that the step has been executed correctly when it is completed.

- Exceptions and handlers augment the control flow constructs of the step kinds.

- Messages and reactions are another form of reactive control and greatly increase the expressive power of Little-JIL.

- Parameters passed between steps allow communication of information necessary for the execution of a step and for the return of step execution results.

- Resources are representations of entities that are required during step execution. Resources may include the step's execution agent, permissions to use tools, and various physical artifacts.

2.3.2 PeCo

PeCo, which stands for Peer Collaboration, is proposed by Proteus Technologies, LLC. It is a Java-based collaborative workflow management system that is composed of peers/agents, core services, applications, and portable plug-ins and enable generic system integration. It aims at decentralising workflow management using collaborative technologies and concepts while providing a pluggable framework for combing business process applications and human contributors.

In PeCo, workflow peers are responsible for a particular application role in a workflow's enactment. Core services, i.e., group coordinator factory, role coordinator, deployment tool, data extractor factory and administrator, are used for system initiali-

sation and system administration. The important characteristics of the PeCo architecture including agent/peer discovery, fault tolerance, and peer availability awareness are supported by Jini infrastructure and services. Generally speaking, workflow peers join enactment groups through the interaction with group coordinators and then peers coordinate and interact with private applications, user inboxes, and other peers to perform workflow tasks, through the use of portable plug-ins.

2.3.3   An architecture based on WWPD and WWP

Another ongoing p2p-based workflow project is conducted at Manchester Metropolitan University. This project shows 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 centralised feature of the system is called WWPD, which provides a peer registration service and maintains a list of active peers and their profiles. With support of this architecture, peers are allowed to register with the system and offer their services and resources to other peers. 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.

2.3.4   SwinDeW

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. Moreover, SwinDeW is further extended to support incompletely-specified processes in the decentralised environment. New technologies for handling incompletely-specified processes at run-time are presented. 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 with support for both completely- and incompletely-specified processes. As a consequence, workflow systems will be expected to be widely deployable to real world applications to support processes, which was infeasible before. Its extended system SwinDeW-S also supports web services based service composition based on OWL-S[OWL01].

2.3.5 Discussion

The above approaches give up conventional client-server architecture and adopt a novel and decentralised architecture to support workflow process management. Especially, the few efforts that combine multi-agent computing paradigm with existing workflow technology have opened new ground in workflow, and in the process support area in general. The distinguished features of multi-agent computing paradigm make it suitable to tackle the problems that relate to the client-server architecture. These work reveal the potential of multi-agent based workflow.

However, it is obvious from the literature review of those research on implementing workflow in a multi-agent platform is still quite immature with many problems addressed inadequately. The work reported on WWPD and WWP, is only some conceptual ideas about linking workflow with p2p system without any concrete analysis of the potential system. Approaches like PeCo, mainly concentrate on decentralising workflow process instances in real time in order to remove potential performance bottlenecks of system and offer enhanced system openness. However, some aspects that are crucial to decentralised workflow enactment have not been addressed sufficiently by these approaches. For example, it is not really clear that in these approaches how the data of process definition are managed so that decentralised agents are able to access task information in real time. Also, how the processes are instantiated are not addressed by these approaches. Issues such as dynamic participants selection, work allocation also have not been addressed sufficiently. Moreover, incomplete process support in a decentralised environment is only addressed by SwinDeW.

SwinDeW addresses most of the above 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 in which agents' coordinating mechanisms are embedded. It thus blurs the business level requirements and system level requirements. When new technologies come out, they 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 systems, which means repeating established work. This is against the basic software engineering principle. Little-Jil falls into the same problem category as SwinDeW.

2.4 Research related to incomplete process support

Flexible workflow support is one of the important research area in the development of workflow management systems. But only very few work has been performed so far due to the difficulties inherited from the conventional workflow architecture and not many approaches can be discovered in the literature.

2.4.1 WASA

WASA workflow [Wes98] is a research project developed at a German University, Potsdam. This project tries to apply the workflow technology in the domain of scientific application and engineering. A formal language, conceptual design, and prototype implementation of flexible distributed workflow management systems based on object-oriented middleware was developed in the WASA project, .

Flexibility is considered as an important research issue in WASA which uses a hierarchical workflow execution approach based on a set of states and accompanying state transitions of workflow instances. A composite activity can have a nested structure, and activity models are created using sets of activity modelling operations. It is also identified that some unpredictable aspects cannot be modelled completely at design time. Therefore, incomplete process support should be provided as a new functionality for a workflow system. Some operations are thus presented to help the workflow meet the flexible requirements, which contain operations for user intervention and operations for dynamic change.

2.4.2 WORKWARE

WORKWARE[Hav01]is a project that aims at human-centred solutions. Havard believes that interactive enactment should be adopted more strongly as a framework to support flexible workflow modelling. Incomplete workflow models are thus allowed to emerge.

Their approach shows that the execution of a workflow model should be changed from completely automated to interactive enactment based, and that interaction can be a suitable framework for understanding and designing flexible workflow management systems. Interactive enactment allows intervened control and activation of an changing online model so that at the design time the model needs not be completely accomplished and doesn't has to be consistent. A general architecture of workflow

management system is presented, which has three layers:

- shared workflow models,

- a number of model activators

- and an integrated user interface.

According to the architecture, the model activators adopt the shared workflow models to provide connecting and activating services using the user interface. This research also shows the WORKWARE prototype developed, which attempts to re-interpret the concept of workflow to contain processes with emerging structure.

2.4.3 Pockets of Flexibility

Researchers at the University of Queensland, Australia proposes a concept of "Pockets of Flexibility"[SSO01, MS02]. Based on this concept, a process modelling and enactment approach was presented, which allows capture of both complete and incomplete process requirements using the same framework. Flexibility in this research is regarded as the capability of the workflow process to be executed on the basis of a loosely, or partially specified model, which means that the full specification of the model can be made in real time, and may vary according to different process instances. In order to provide a modelling framework that provides real flexibility, the issues that affect the paths of different process instances together with the process definition are considered. An approach that tries to make the flexible parts of the workflow process is developed. With their framework, the concept called open instance that is made of a core process and several pockets of flexibility are explained. The notation "pocket" is a distinguished structure within the workflow model, which is consisted of workflow fragments, that can represent a single primary business activity, or a complex sub-process; and a special activity called the build activity, which performs the rules and constraints with which those fragments can be composed together for a running instance. Thus, the build activity is the key point of the research and provides the functionality to realise incomplete activities that are defined in the process model at design time into concrete executable activities for different running process instances.

2.4.4 Discussion

From the above literature, we can see that research on the support of flexible workflow is at an early stage. The existing approaches discussed above tackle the problems

of flexible workflow support mainly from the model construction perspective but say nothing from the system coordinating point of view. In addition, these approaches are all based on conventional workflow architectures. Therefore, research in decentralised workflow environments from the point of view of system coordination might help.

2.5 Requirement Analysis

After analysis of the existing problems in the conventional workflow area and review of some of the current approaches for workflow management, we believe that solving the problems that relate to centralised workflow architecture and incomplete process support has become crucial for the development of future workflow management system. Also, "industry trends such as virtual enterprises and flattening of organisational structures indicate that the future image of business will include distributed groups of collaborating teams that combine talents and skill sets to create new methodologies and processes. Therefore, there is growing need for the next generation of workflow systems to be built in a truly decentralised manner, providing support for both complete and incomplete processes."[Yan02a].

The emergence of multi-agent technology provides a good opportunity for the decentralisation of workflow systems. The few efforts that replace the client-server architecture with collaborative and decentralised framework of multi-agent/P2P platform have shown potential benefits. More recently, multi-agent/P2P based workflow systems have also been considered as one of the most important future directions for workflow research[MS02]. Therefore, decentralised workflow that is based on multi-agent/p2p platform might be a valuable solution for future workflow process support. However, to have a cost-effective and decentralised workflow system based on multi-agent/P2P, we would expect to adopt the existing work that has been widely used for conventional workflow systems as much as possible and although a centralised server is expected to be eliminated, the services conventionally provided by the centralised data repository and workflow server should remain as shown in Figure 2.1.

To achieve these requirements, a decentralised workflow system should:

- adopt a multi-agent based, loosely-coupled architecture and provide a flexible framework for integrating workflow process applications and end users with the needs of neither a centralised workflow engine for coordination nor a centralised data storage,
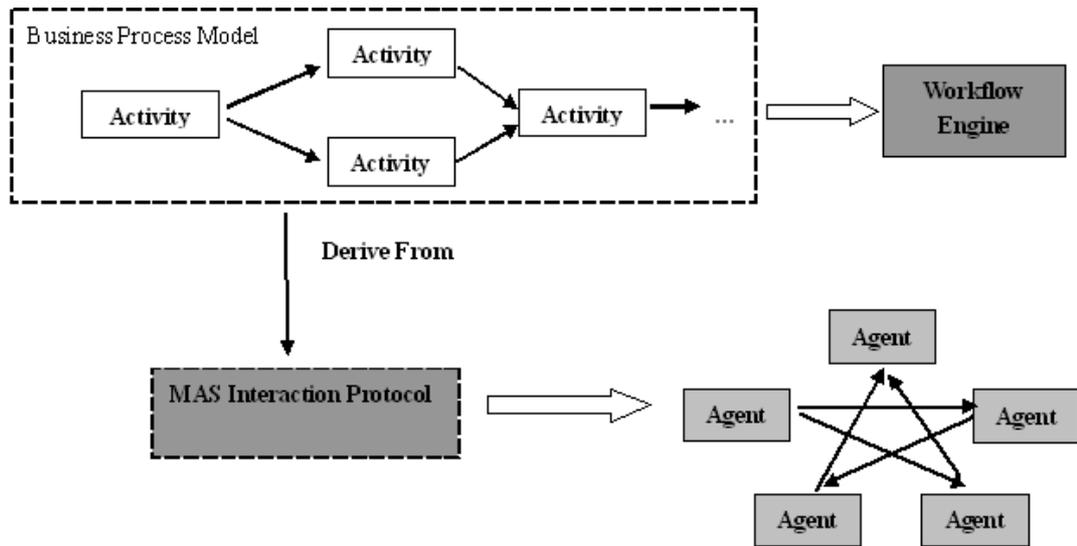
Figure 2.1: From conventional workflow architecture to multi-agent architecture

- let the distributed nodes (participants) use data that are conventionally stored in centralised workflow engine when needed,

- distribute the services that are provided uniquely by conventional workflow server to different participants so that the functionalities of the system would be the same after the change of system architecture,

- provide ways to help service providers and service consumers to communicate directly to reduce the network traffic,

- try to adopt service-oriented applications, which is the current standard for open application systems, as much as possible, and

- provide sufficient support for incomplete workflow process, which allows incomplete processes to be designed at build-time, and instantiated and executed at run-time.

In addition, from the engineering point of view, when attempting to achieve the above requirements, it is wise to use existing technologies, tools and formal business process models as much as possible to reduce repeated work. It is also easier for the acceptance of the new system by end users when the whole system is shifted from the conventional architecture to new architecture seamlessly. As addressed in the literature review, some existing multi-agent systems can satisfy most of the requirements listed above. However, the weakness of almost all of the current approaches is that they ignore the useful

work already done on conventional workflow systems.

This research, therefore, builds a pure decentralised workflow management system starting from existing business process models. It connects the workflow management world and the multi-agent world together in several different ways. In the three layer conceptual model given in Figure 2.2, we can see that the business process model and interaction protocol may or may not be at the same conceptual level. A formalised business process model that describes high level abstract information sits in the logic layer. A more detailed process model sits in the implementation layer. However, the interaction protocol for multi-agent system is always located in the implementation layer. Therefore, there are three possibilities for the production of interaction protocols for enacting a multi-agent based workflow management systems according to the framework.
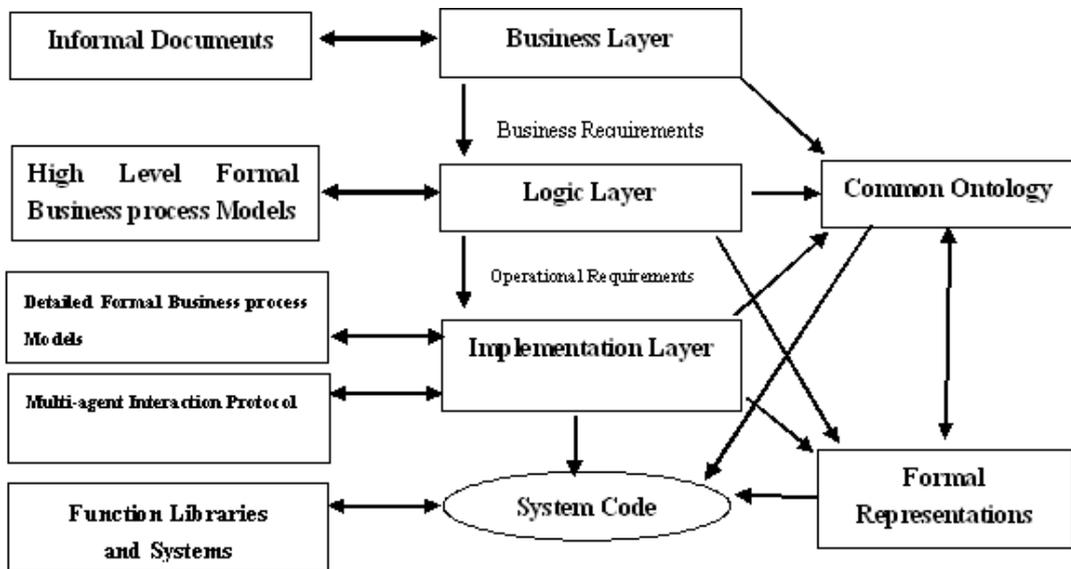
Figure 2.2: Three conceptual layers based framework

- Deriving the interaction protocol directly from the informal business requirements that is from the top layer in the framework.

- Deriving the interaction protocol from the formally defined abstract business process model at the logic layer in the framework.

- Deriving the interaction protocol from the formally defined detailed business process model at the implementation layer in the framework.

Since formally defined business process models are widely used in conventional workflow management systems, the first possibility is ignored in this research and I focus on the remaining two. I assume all the given formal business process models are correct models and are coherent with the informal business requirements.

The key issues of this research are therefore:

- How to use the formal business process models that are defined at different abstract levels for the construction of multi-agent system based workflow management systems.

- How to make sure that a multi-agent system that uses the business process model as its requirement behaves strictly coherently with the BPM.

- How to adapt a multi-agent architecture to solve some of the problems in conventional workflow management systems, for instance, supporting incomplete processes.

For highly abstract business process models, even for those that are formally defined, it is not always possible to derive interaction protocols from them automatically. Human intervention is needed. The first part of my research considers how to help the human modeler produce the interaction protocols given a high abstract and formally defined process model. A framework for this is defined and a temporal logic is used as the main tool to ensure the functional equivalence between the input process model and output interaction protocol of the framework.

For business process models that are defined at the implementation level in the conceptual framework, because they normally give enough information for system implementation, automatic derivation of an interaction protocol is possible. I use two approaches to achieve this.

- One is to perform a mapping between the two languages that are used for describing business process models and interaction protocols.

- Another is to use the business process model (BPEL4WS) directly in the potential multi-agent system to tell agents what they need to do and the interaction protocol (LCC[Rob04]) is also used to tell agents how they perform the required tasks defined in the business process model.

2.6 Summary

The motivations of this research have been proposed in this chapter. Some of the main problems in conventional workflow research, such as bad performance, single point failure of system, unsatisfactory system openness, and insufficient support for incomplete process, as well as causes of these problems have been analysed. The literature on these problems has been reviewed. Based on the problem analysis and the given literature review, a multi-agent based workflow architecture is suggested to support both stable and flexible workflow. Detailed requirements for multi-agent based workflow have also been analysed. To conclude the chapter, we describe the new platform and approaches that should be able to use existing workflow models and tools as much as possible and also address possible solution for such requirements according to a three layer conceptual framework.

Chapter 3

Using High Level Formal BPMs For MAS Development

As discussed in chapter 2, using formal business process models as requirements to establish the initial social order of multi-agent system can be performed in several different ways. In this chapter, the framework for bridging formalised high level business process models down to MAS interaction protocols (IPs) is presented. Based on the three layer conceptual framework, high level business process models and IPs are located at different conceptual levels (one is at the logic layer and one is at the implementation layer) as shown in Figure 3.1:
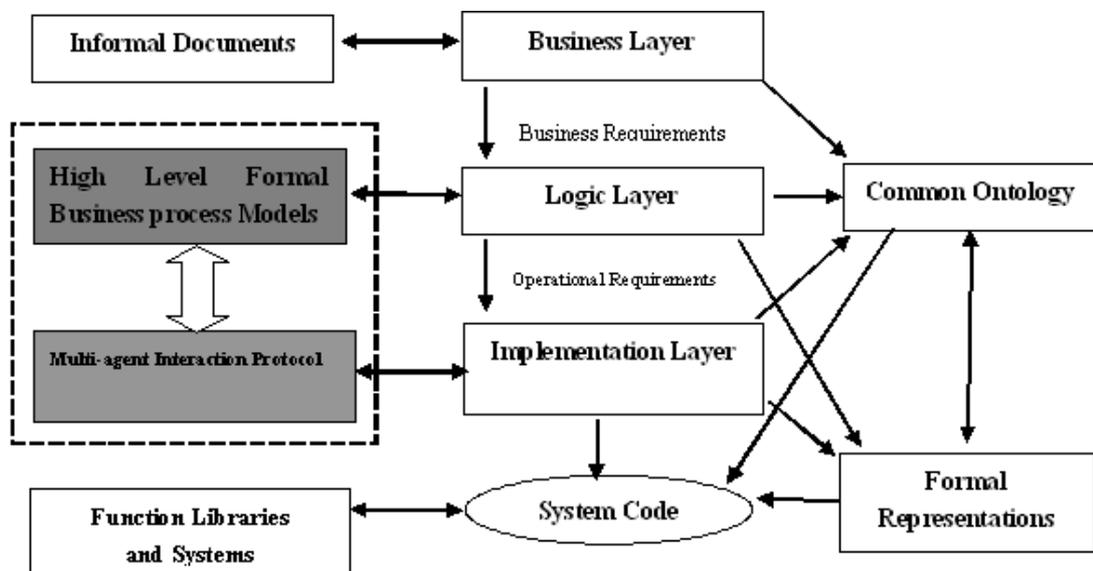
Figure 3.1: Bridging high level formal BPMs to IPs

This approach is based on linking requirements(business process models) to IPs by matching the formal specifications derived from both of them to make sure that the IPs designed manually meet the functional requirements strictly and to make sure they will remain consistent as requirements change.

In section 3.1, we describe our framework in detail including all the components in our framework and showing how they cooperate with each other in the IP modelling task. Components of our framework are explained in section 3.2 and through to section 3.6. In section 3.7, we demonstrate the algorithm for verifying the two linear temporal logic clauses derived. The algorithm for generating a concrete IP (LCC) from a verified simple protocol properties checking (SPPC) model is discussed in section 3.8. Based on the mechanism discussed in the earlier sections, a case study is used in section 3.9 to illustrate how the framework supports real word IP production. Implementation and discussion are given in section 3.10 and 3.11

3.1 Process Model Based MAS Interaction Protocol Modelling Framework

Conventionally, a high level process model describes high level requirements whereas a MAS protocol is a detailed system specification which should be consistent with both business level and system level requirements. Thus, automatic workflow enactment by a multi-agent system is difficult since high level process models don't necessarily contain any system level information. For example, in a business process model, there might be an activity called *print papers*. At requirement level, this only means some papers need to be printed out, but says nothing about what actions should be performed in unexpected circumstances, say, when a printer is out of paper. Normally, a MAS interaction protocol is produced manually. How can we make sure that all the properties defined in a process model are preserved by the MAS interaction protocol properly, since writing a complex protocol by hand is usually error prone. One way of solving this problem is to undertake model checking or simulation after the MAS protocol is completed [Wal04a, NOW05]. With the support of formally specified business process model, we may be able to verify the MAS protocol automatically and thus reduce the effort and time that we normally spend during the model checking/simulation process. We propose a process model based interaction protocol modelling framework which is shown in Figure 3.2. There are six main parts of this framework as shown below. We
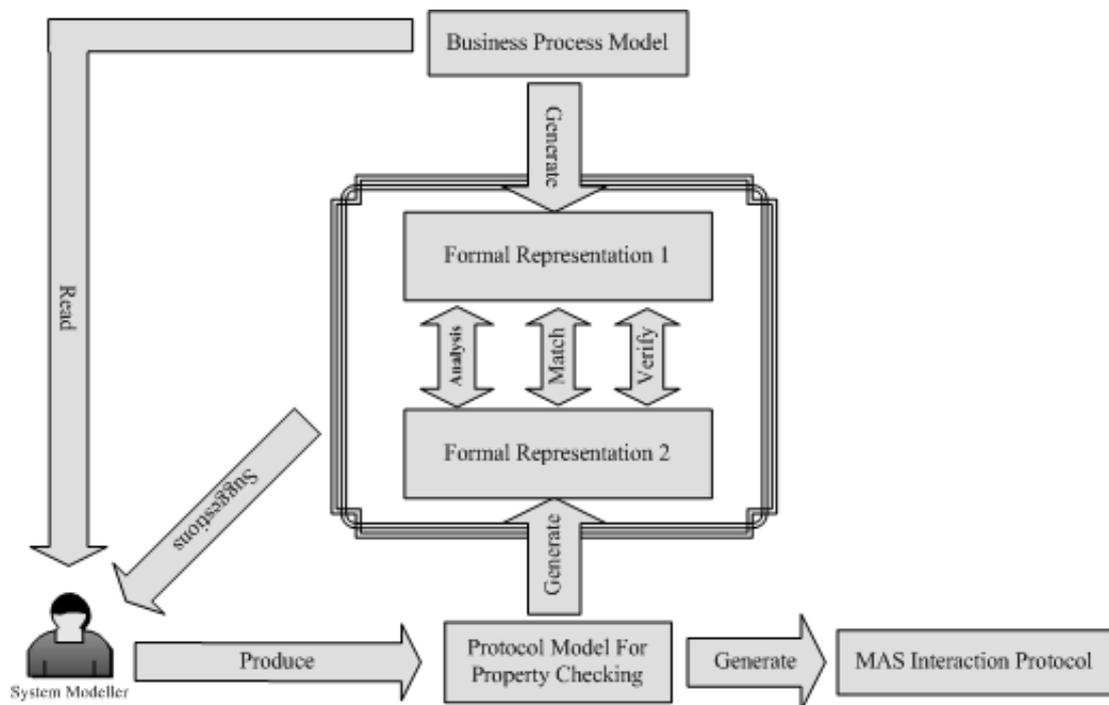
Figure 3.2: BPM based interaction protocol modelling framework

will go through each of them in later sub-sections.

- Business Process Model (depicting a high level business process)

- System Modeller (person who builds the IP)

- Property Checking Model (built by the system modeller)

- MAS Interaction Protocol (define the manner of agents' interactions)

- Formal Representation 1 (linear temporal logic, FR 1)

- Formal Representation 2 (linear Temporal logic, FR 2)

3.1.1 How our framework works for IP's modelling task?

With our framework, the IP modelling task consists of the following steps:

- The business process model (high level formal representation) is loaded and then all the temporal relationships of functional properties defined in it are translated into a temporal logic (formal representation 1/FR1). These are the temporal business requirements.

- The system modeller uses the process model (which in practice may be diagrammatical, textual or formal descriptions) and produces a property checking model for the MAS interaction protocol that is composed of sets of dialogues. The temporal relations of functional properties defined in the process model contained in each dialogue are translated into a temporal logic representation (formal representation 2/FR2) later on.

- If FR1 implies FR2($FR1 \models FR2$), then the system modeller will get some suggestions from system, which indicate proper actions that the system modeller should take next or if $FR2 \models FR1$ fails, suggestions from the system about why problems exist(what is the discrepancy between temporal orders of properties) so that the system modeller can fix the problem as early and quickly as possible.

- After property checking of the model is properly accomplished, it can then be translated into to a standard MAS protocol.

3.2   High level Process Model

A formalised high level business process model gives a logical description of business processes that obeys and keeps track of business principles and requirements that have been described in informal business requirements. It depicts the conditions and actions of processes, the relationships and constraints between them and the data upon which the processes operate. This formal representation can be used to check for errors in the model, and it provides a basis for offering advice and a foundation for forecasting organisational behaviour. Processes described at this abstract level are relatively independent of the deployed technologies, including software and hardware systems, and therefore are more robust compared to more detailed business process models (at implementation level). Furthermore, the business process changes less rapidly than the system specified to support that process. People usually don't want to re-write their requirements only because of the adoption of new technology. Therefore, in our framework, we use process models as a starting point to avoid unnecessary requirement re-capture work when we adopt MAS technology for process management.

There are many high level business process modelling languages available to fit different desires of different organisations. But some common features are required by almost all business process modelling languages. A high level business process model is normally composed of sets of activities, which define the basic tasks that

are undertaken in the process, and sets of links which define the different relationships (sequential, parallel, etc) between the activities. Each activity has several properties: *ID, Role, Input, Preconditions, Postconditions, Outputs, textual descriptions*, which describe both functional information (*ID, Role, Input, Predoncitions, Postconditions, Output*) for execution of the activity and non-functional information (*textual descriptions*) for other purposes. The fundamental business process modelling language (FBPML)[CBR98], for example, is such a high level process modelling language.

3.3   MAS Interaction Protocol

A MAS interaction protocol is the product of our framework, which is ensured consistent with the given process model(describing the temporal business requirements). Although any standard protocol language is applicable for this framework, the lightweight coordination calculus(LCC)[Rob04] is used for our work.

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:

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

All the notations in this thesis are defined using BNF meta symbols which is explained in [MM96]. A dialogue framework defines a space of possible dialogues determined by message passing, so the protocols allow constraints to be specified on the circumstances under which messages are sent or received. Two forms of constraints are permitted:

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

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

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

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.

To enable an 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 [Rob04]) to expand the dialogues state. This works as follows:

- An 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 agent extracts from $\mathcal{P}$ the dialogue clause, C_i, determining its part of the dialogue.

- Applying the rewrite rules in [Rob04] 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:

$$\left\langle C_i \xrightarrow{M_i, M_{i+1}, \mathcal{P}, O_i} C_{i+1}, C_{i+1} \xrightarrow{M_{i+1}, M_{i+2}, \mathcal{P}, O_{i+1}} C_{i+2}, \quad ..., C_{n-1} \xrightarrow{M_{n-1}, M_n, \mathcal{P}, O_n} C_n \right\rangle$$

- The original clause, C_i, is then replaced in $\mathcal{P}$ by C_n to produce the new protocol, $\mathcal{P}_n$

- The agent can then send the messages in set O_n, each accompanied by a copy of the new protocol $\mathcal{P}_n$.

3.4 System Modeller

A system modeller is someone who performs the IP modelling task based on the given process models. He/She is responsible for understanding the process models and designing appropriate IPs. Since it is difficult to have automatic translation from a process

model (at least for a high level abstract one) to an IP because the properties defined in the process model is a subset of that in the IP as we noted earlier and the gap between the process model and the IP is too huge to be bridged, human intervention in the process of IP modelling is crucial in this framework.

3.5 Property Checking Model

A property checking model is a product that is produced by a system modeller which can then be translated to concrete IPs automatically. The reason why a system modeller needs to first produce a property checking model rather than producing concrete IPs directly is because that not all the protocol describing languages could express properties or temporal relations between dialogues that are taking place among sets of agents clearly. For instance, LCC describes dialogues based on the viewpoint of each agent, which makes it very hard to discover the temporal orders of properties defined in requirements that often specify temporal relations between agents. In order to facilitate protocol property checking and separate the checking method from a particular protocol language, we define a simple MAS interaction protocol modeling language: Simple Protocol Properties Checking(SPPC) Language to help. A SPPC protocol model is built based on the message passing taking place between two agents and the constraints associated with the message. The temporal orders of messages can also be expressed by SPPC.

- **Representing a message in SPPC:** Any message defined in SPPC model is defined by a tuple:

  - *msg(ID,preconditions, message body, postcondition, sender, receiver)*

  where a *message body* only can be sent out from its *sender* when its *precondition* holds and can cause certain effects(*postcondition*) when it is received by its *receiver*.

- **Temporal order between messages:** *A then B* means *B* occurs after *A*. *Invoke(A)* means that *A* occurs while being invoked, which is used to represent loops.

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

The syntax of SPPC is as follows:

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

SPPC is developed only for the purpose of MAS protocol property checking. In other words, it is not a protocol language that is intended to use by agents directly. However, the SPPC protocol can be translated into concrete protocols that are described by other formal protocol description languages such as LCC etc.

3.6   Formal Representations: FR1 and FR2

Formal representation 1 and formal representation 2 are linear temporal logic representations derived from the process model and property checking model respectively and are identical in the phase of SPPC model verification. If they are temporally identical, we can conclude that the property checking model(system specification) is strictly consistent with process model(requirement) functionally.

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

φ	φ is true in all future moments
$\diamond\varphi$	φ is true in some future moment.
$\bigcirc\varphi$	φ is true in the next moment in time
$\varphi\,\mathcal{U}\,\psi$	φ is true up *until* some future moment when ψ is true

Table 3.1: Basic Syntaxes of Linear Temporal Logic

3.6.1   Deriving representation 1 (FR1) from the process model

$$\left[primary\_activity \left(\begin{array}{l} ID, Role, [Precondition_1, ...Precondition_m], [Input_1, ..., Input_n], \\ [Output_1, ..., Output_x], [Postcondition_1, ...Postcondition_y] \end{array} \right) \right]$$

$$\Downarrow$$

$$A_i = \left[\begin{array}{l} (\bigwedge_{i=0}^{m} associate(Role, Precondition_i)) \wedge (\bigwedge_{i=0}^{n} associate(Role, Input_i)) \\ \rightarrow \diamond(\bigwedge_{i=0}^{x} associate(Role, Postcondition_i)) \wedge (\bigwedge_{i=0}^{y} associate(Role, Output_y)) \end{array} \right]$$

The notation *primay_activity* above defines a primary activity in process model.

- *ID*, *role*, *preconditions*, *inputs*, *postconditions*, *outputs* are properties associated with it.

- The symbol $\Downarrow$ means that the term above it can be expressed by the temporal logic clauses below it.

- Predicate *associate(Role,Properties)* defines the association of role and properties in an activity.

- $\bigwedge_{i=0}^{m} associate(Role, Property)$ represents the association of an activity's property and the role that should perform this activity.

- For simplicity, all the activities from a business process model are represented by the symbol A_i where $i \in [0, \infty)$.

The temporal logic clause shown above indicates that the conjunction of *preconditions and Inputs* can always imply the conjunction of *postconditions and outputs* at some future time.

Beside the temporal relations between the properties defined for individual activities, a process model also defines the temporal order of different activities. The notation *link* indicates how the activities from a process model are connected. Different

Link Notations	**Linear Temporal Logic Clauses**
$link(ID, Precedence\_Link, A_1, A_2)$	$A_1 \land \Diamond A_2$
$link(ID, or, A_1, \{A_2, A_3, ..., A_i\})$	$A_1 \land \Diamond (A_2 \lor A_3 \lor ..., \lor A_i)$
$link(ID, and, A_1, \{A_2, A_3, ..., A_i\})$	$A_1 \land \Diamond (A_2 \land A_3 \land ... \land A_i)$

Table 3.2: Representing link notations with linear temporal logic

link types are given in Table 3.2, such as *Precedence_Link* which represents a sequential time order between two activities. Table 3.2 also shows how the temporal relations defined by links in a business process model can be expressed as temporal logic clauses.

The linear temporal logic clauses derived for representing the temporal order between two activities are normally in relatively complex forms as follows, for simplicity, we use $\bigwedge_{i=0}^{n} \mathcal{P}_i$ to represent the properties defined for an activity, where $\mathcal{P} \equiv$ *assoicate(Role, Properties)*:

$$(\bigwedge_{i=0}^{n} \mathcal{P}_i \rightarrow \Diamond \bigwedge_{j=0}^{m} \mathcal{P}_j) \land \Diamond (\bigwedge_{k=0}^{o} \mathcal{P}_k \rightarrow \Diamond \bigwedge_{l=0}^{p} \mathcal{P}_l)$$

In the above clause, $\bigwedge_{i=0}^{n} \mathcal{P}_i \rightarrow \Diamond \bigwedge_{j=0}^{m} \mathcal{P}_j$ on the left hand side of $\land \Diamond$ represents an activity (A_1) and $\bigwedge_{k=0}^{o} \mathcal{P}_k \rightarrow \Diamond \bigwedge_{l=0}^{p} \mathcal{P}_l$ on the right hand side represents the activity that is defined in a process model right after A_1 in a sequential order. However, the above clauses can not be easily used for automated reasoning as basic units. In order to facilitate the reasoning process, the rules shown in Figure 3.3 are used to re-write the complex temporal logic clauses into simple ones. Symbols A_i and B_i used in the rules means the conjunctions of properties defined for process activities.

The rewrite $rule_1$ is used to represent the sequential temporal relation between two activities using the temporal relation between certain properties. The basic rationale underlying this rule is that the temporal order between two activities (LR) on the left hand side of $\Rightarrow$ can be expressed by the time order between the *output $\land$ postcondition* of former activity and the *input $\land$ precondition* of latter activity. The notation ID_i used above is important for run time property checking. The order of i defined for the clauses' *IDs* must be in a lexicographical order so that we know where to start when we perform the property checking. When performing property checking, the three clauses (RR) derived using $rule_1$ on the right hand side of $\Rightarrow$ are checked one by one strictly according to the ID_i associated with them. The original clause (LR) will be proved only when all of the three clauses are proved. $Rule_2$ and $rule_3$ defined are used to represent the parallel and choice relations between sets of activities using relations

$$(\mathcal{P} \rightarrow \Diamond \mathcal{P}_1) \wedge \Diamond (\mathcal{P}_2 \rightarrow \Diamond \mathcal{P}_3) \Rightarrow \left[\begin{array}{l} clause(ID_0, \mathcal{P} \rightarrow \Diamond \mathcal{P}_1), \\ clause(ID_1, \mathcal{P}_2 \rightarrow \Diamond \mathcal{P}_3) \end{array} \right] \qquad (rule_1)$$

$$(\mathcal{P} \rightarrow \Diamond \mathcal{P}_0) \wedge \Diamond ((\mathcal{P}_1 \rightarrow \Diamond \mathcal{P}_2) \wedge ... \wedge (\mathcal{P}_n \rightarrow \Diamond \mathcal{P}_{2n})) \Rightarrow \left[\begin{array}{l} clause(ID_0, \mathcal{P} \rightarrow \Diamond \mathcal{P}_0), \\ clause(ID_1, \mathcal{P}_0 \wedge \Diamond \mathcal{P}_1), \\ clause(...), \\ clause(ID_n, \mathcal{P}_0 \wedge \Diamond \mathcal{P}_n), \\ clause(ID_{n+1}, \mathcal{P}_1 \rightarrow \Diamond \mathcal{P}_2), \\ clause(...), \\ clause(ID_{2n}, \mathcal{P}_n \rightarrow \Diamond \mathcal{P}_{2n}) \end{array} \right] \qquad (rule_2)$$

$$(\mathcal{P} \rightarrow \Diamond \mathcal{P}_0) \wedge \Diamond ((\mathcal{P}_1 \rightarrow \Diamond \mathcal{P}_2) \vee ... \vee (\mathcal{P}_n \rightarrow \Diamond \mathcal{P}_{2n})) \Rightarrow \left[\begin{array}{l} clause(ID_0, \mathcal{P} \rightarrow \Diamond \mathcal{P}_0), \\ clause(ID_1, (\mathcal{P}_0 \wedge \Diamond \mathcal{P}_1) \vee ... \vee (\mathcal{P}_0 \wedge \Diamond \mathcal{P}_n)), \\ clause(ID_2, \mathcal{P}_1 \rightarrow \Diamond \mathcal{P}_2) \\ clause(...), \\ clause(ID_n, \mathcal{P}_n \rightarrow \Diamond \mathcal{P}_{2n}) \end{array} \right]$$
$$(rule_3)$$

Figure 3.3: Rules for rewriting complex linear temporal logic clauses

between properties. The re-writing principles for $rule_2$ and $rule_3$ are the same as for $rule_1$.

After performing the re-write rules, all the temporal logic clauses derived from a business process model are of the following form:

$$
\begin{array}{rcl}
Linear\,Temporal\,Representations & ::= & \{Clause, ..., Clause\} \\
Clause & ::= & clause(ID, \bigwedge_{i=0}^{n} \mathcal{P}_i) | \\
 & & clause(ID, \bigwedge_{i=0}^{n} \mathcal{P}_i \rightarrow \Diamond \bigwedge_{j=0}^{m} \mathcal{P}_j) | \\
 & & clause(ID, \bigwedge_{i=0}^{n} \mathcal{P}_i \wedge \Diamond \bigwedge_{j=0}^{m} \mathcal{P}_j) | \\
 & & clause(ID, \bigvee_{i=0}^{n} (\bigwedge_{i=0}^{n} \mathcal{P}_i \wedge \Diamond \bigwedge_{j=0}^{m} \mathcal{P}_j)) \\
\mathcal{P} & ::= & associate(Role, Property) \\
ID & ::= & start | end | Term \\
start & ::= & Term \\
end & ::= & Term \\
Role & ::= & Term \\
Property & ::= & Term
\end{array}
$$

A business process model might also contain composite activities that are composed of several primary activities by different links. Using the approach proposed above, a composite activity can also be represented by linear temporal logic clauses.

Because after a composite activity is translated into linear temporal logical clauses, the relationships between its sub-activities are processed in the same way that is used for processing primary activities.

3.6.2 Deriving formal representation 2 (FR2) from the property checking model

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

$$\Big[\ msg(pre([A_1,...,A_m]), mb([B_1,...,B_n]), post([C_1,...C_x]), sender(Role), receiver(Role_1)) \ \Big]$$

$$\Downarrow$$

$$Msg_i = \left[\begin{array}{l} \bigwedge_{i=0}^{m} associate(Role, A_i) \\[2mm] \rightarrow \Diamond \left(\left(\begin{array}{c} \bigwedge_{i=0}^{n} associate(Role, B_i)) \\ \vee \\ (\bigwedge_{i=0}^{n} associate(Role_1, B_i)) \end{array} \right) \rightarrow \Diamond \bigwedge_{i=0}^{x} associate(Role_1, C_i) \right) \end{array} \right]$$

One point that has to be mentioned here is the association of properties and roles in a SPPC model. It is clear that the properties defined in the precondition of a message should be associated with the message sender and the properties in postcondition of a message should be associated with message receiver, whereas the properties defined in the message body are associated with both message sender and receiver in the MAS protocol. Therefore, in the formal representation derived from it, we have to specify this explicitly to make sure that the right properties are associated with the right roles.

Table 3.3 shows how the temporal relations defined between messages in a SPCC model can be expressed by linear temporal logic clauses.

SPPC links	Linear Temporal Logic Clauses
Msg_1 *then* Msg_2	$Msg_1 \rightarrow \Diamond Msg_2$
Msg_1 *par* Msg_2	$Msg_1 \wedge Msg_2$
Msg_1 *or* Msg_2	$Msg_1 \vee Msg_2$

Table 3.3: Representing SPPC link notations with linear temporal logic

By applying the above rules, a SPPC model can be expressed by linear temporal logic in following forms:

$$
\begin{aligned}
Linear\,Temporal\,Logic\,Representations \quad &::= \quad C \\
C \quad &::= \quad C_1 \rightarrow \Diamond C_2 | Msg | Msg_1 \rightarrow \Diamond Msg_2 | \\
&\quad\;\; Msg_1 \wedge Msg_2 | Msg_1 \vee Msg_2
\end{aligned}
$$

3.7 Performing Property Checking

3.7.1 Issues for role checking

After representing both process model and SPPC model in linear temporal logic clauses, the relationships between properties can be derived and checked. However, another issue we have to check in the process model when performing protocol modelling is whether the right properties are associated with the right roles or not. In a conventional high level business process model, every activity can only have one role performing on it for a given business scenario. However, since a MAS interaction protocol model describes extra system information, which may bring new roles in. Intuitively, if the role specified in the predicate *associate* in requirement is also specified in protocol specification (or can be matched to the role specified in process model) and the properties associated with it are the same, we can conclude that the properties are associated with right role in the final protocol. The assumption we make here is that an ontology of roles is already available.

3.7.2 Temporal order checking

As explained earlier, both a business process model and a SPPC model can be expressed by temporal logic clauses. In principle, we can thus prove whether or not the temporal relationships defined between the properties in a business process model can be implied by those functional properties that are defined in a SPPC model.

In preparing a problem for our properties checking process, we need to divide our knowledge into three parts:

- A set of clauses known as the **goals**, which defines that goals that need to be proved. For our problem, the goals are the linear temporal logic clauses (LTL_1) derived from business process model.

- A set of clauses known as the **set of support (or sos)**, which defines the important facts about our problem. Every resolution step resolves a member of the set

of support against another axiom, so the search is focused on the set of support. For our domain, the set of support it the linear temporal logic clauses (LTL_2).

- A set of **rewrites** applied on SPPC generated linear temporal clauses. Rewrites are not equations, they are always applied in the left to right direction. The rewrites that are used for our problem are as follows:

$$(A \rightarrow \Diamond(A_1 \rightarrow \Diamond A_2) \xrightarrow{A \rightarrow \Box A} ((A \wedge A_1) \rightarrow \Diamond A_2) \qquad (A_i \equiv \bigwedge_{i=1}^{n} \mathcal{P}_i) \quad (rule_1)$$

$$(A \rightarrow \Diamond(A_1 \rightarrow \Diamond A_2) \xrightarrow{A_1 \rightarrow \Box A_1} (A \rightarrow \Diamond(A_1 \wedge A_2)) \qquad (A_i \equiv \bigwedge_{i=1}^{n} \mathcal{P}_i) \quad (rule_2)$$

$$(A_1 \rightarrow \Diamond((A_2 \vee A_3) \rightarrow \Diamond A_4)) \xrightarrow{LTL_2 \models A_2} (A_1 \rightarrow \Diamond(A_2 \rightarrow \Diamond A_4)) \qquad (A_i \equiv \bigwedge_{i=1}^{n} \mathcal{P}_i) \quad (rule_3)$$

$$(A_1 \rightarrow \Diamond((A_2 \vee A_3) \rightarrow \Diamond A_4)) \xrightarrow{LTL_2 \models A_3} (A_1 \rightarrow \Diamond(A_3 \rightarrow \Diamond A_4)) \qquad (A_i \equiv \bigwedge_{i=1}^{n} \mathcal{P}_i) \quad (rule_4)$$

Rule$_1$ and *rule*$_2$ eliminate parts of the time relationships between several properties. If A_1 appears after A and A_2 appears after A_1, by applying *rule*$_1$, we can conclude that after we get property A, we will get property A_1 and A_2 in some future time. On the other hand, by applying *rule*$_2$, we can conclude that after we get property A and A_1, in some future time we will have property A_2. These two rules are used to deal with the circumstances where the conjuncted properties for a activity defined in a business process model are used separately in SPPC model for different messages.

For example, if two properties (A and A_1) are defined for a primary activity as inputs and a property A_2 is defined as its output, based on our proposed translation principle in previous sections, we will get:

$$clause(ID_i, (A \wedge A_1) \rightarrow \Diamond A_2)$$

However, in a manually produced SPPC model, these three properties might be used in the manner:

$$msg(pre([...]), mb(A), post([...]), sender(...), receiver)$$
then
$$msg(msg(pre([...]), mb(A_1), post([...]), sender(...), receiver)$$
then
$$msg(pre([...]), mb(A_2), post([...]), sender(...), receiver)$$

which after translation, would give us two clauses as follows:

$$clause(ID_i, A \rightarrow \Diamond(A_1 \rightarrow A_2))$$

By applying $Rule_2$, we could say that the functional properties are used consistently in SPPC with way that they are used in a business process model. $Rule_3$ and $Rule_4$ are used to deal with the properties' roles checking.

The algorithm for the functional properties based checking is given in Figure 3.4, which works in the following manner:

- All the *clauses(goal)* derived from a business process model are processed one by one based on their *IDs*' order.

- If a clause is in a form of $goal_1 \rightarrow \Diamond goal_2$, $goal_1$ will be proved first using the *clauses* derived from the SPPC model and so will be $goal_2$ if $goal_1$ is proved.

- If a clause is in a form of $\bigvee_{i=0}^{n} goal_i$, all the $goal_i$ will be proved one by one using the *clauses* derived from the SPPC model.

```
procedure properties_checking(sos,goal)
    inputs: sos, linear temporal logic clauess derived from a SPPC model
            goal, a list that stores linear temporal logic clauess derived from a busienss process model
    output: true, false
    while (goal is not empty)
        fetch the first element(goal₁)
        if (goal₁ is in the form of (goal₂ → ◇goal₃) or (goal₂ ∧ ◇goal₃))
            prove(sos, goal₂, goal₃),
            if (goal₂ is proved)
                prove(sos, goal₃, goal₃)
                if (goal₃ is proved)
                    reture true
                else
                    return false
            else
                return false
        else if(goal₁ is in the form ⋁ⁿᵢ goalᵢ
            for (i in n)
                properties_checking(sos, goalᵢ)
```

procedure $prove(sos, goal_1, goal_2)$

    inputs: sos, *linear temporal logic clauess derived from a SPPC model*

        $goal_1$, *is in the form of* $\bigwedge_{i=0}^{n} associate(Role, P_i)$

        $goal_2$, *is used to defend the time order between properties* $(\bigwedge_{i=0}^{n} associate(Role, Q_i))$

    if $(goal_1 \neq goal_2)$

        while $(P_i \ in \ \bigwedge_{i=0}^{n} associate(Role, P_i) \ are \ not \ all \ proved)$

            if $(sos \models P_i)\&\&(sos \nvDash any \ Q_i \ in \ goal_2)$

                continue

            else

                return false

    else

        while $(Q_i \ in \ \bigwedge_{i=0}^{n} associate(Role, Q_i) \ are \ not \ all \ proved)$

            if $(sos \models Q_i)$

                continue

            else

                return false

Figure 3.4: Basic algorithm for property checking

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

Although the main components of both SPPC and LCC are *messages and constraints*, they are built on different concepts. With LCC, the MAS interaction protocol is defined from the views of different agents where each agent has its own behavior definitions, whereas with SPPC, the protocol model is built based on the message 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 corresponding LCC protocol. *Message body, sender and receiver* from SPPC model together indicate the message being passed and direction of it in LCC. *Junctions* in SPPC can be used as LCC *operators*.

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 that loop. 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.

One important issue 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 need to have a same role defined in them if such a SPPC model is expected to be used for the generation of a LCC protocol as shown below:

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

In contrast, the clauses shown below are not translatable, although they might be rational:

$$msg(MID, mb(...), sender(a(Role_1, ID_1)), receiver(a(Role_2, ID_2)))$$
then
$$msg(MID_1, mb(...), sender(a(Role_3, ID_3)), receiver(a(Role_4, ID_4)))$$
...

The issue of role dependency arises due to the coordinating mechanism of LCC and the potential system architecture that we are trying to achieve. For agents that use LCC protocol for coordination, as we have explained, they have no knowledge of what the coordinating process is until they receive the LCC protocol that contains the states of the whole system. If two messages are sent by two agents $a1, a2$ to two different agents $a3, a4$ at the same time respectively, two separate LCC protocols containing different system states are sent out also. Thus it is quite hard to keep track on the whole system states later on unless there is a centralised coordinator which is what we try to eliminate.

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

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

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 3.5

procedure $seuqence\_precessor(M_1, M_2, M_3)$

    inputs: M_1, M_2, *two element conencted by a "then" operator*

    outputs: M_3, *a processed model with all of its clauses role dependent on each other*

    if $(M_1\ is\ a\ message)\&\&(M_2\ is\ a\ message)$

        if $(M_1\ and\ M_2\ doesn't\ contain\ at\ least\ one\ same\ role)$

            generate a new message (TM) *using receiver of* M_1 *as sender and sender of* M_2 *as receiver*

            $M_3 = M_1\ then\ TM\ then\ M_2$

        else

            $M_3 = M_1\ then\ M_2$

    else if $(at\ least\ one\ of\ M_1\ and\ M_2\ is\ a\ or/par\ structure)$

        $or/par\_precessor(M_1, M_2, M_4)$

        $M_3 = M_4\ then\ M_2$

    else if $(M_1\ is\ a\ message)\&\&(M_2\ is\ a\ "invoke(mid)")$

        fectch the SPPC message (M_4) *that identified by mid*

            if $(agents\ involved\ in\ M_1\ don't\ contain\ the\ sender\ of\ M_4)$

                insert a new message (M_5) *between* M_1 *and invoke(mid)*

                using receiver of M_1 *as its sender and sender of* M_4 *as its receiver*

                $M_3 = M_1\ then\ M_5\ then\ invoke(mid)$

---

procedure $or/par\_precessor(M_1, M_2, M_3)$

    inputs: M_1, M_2, *two SPPC element which can be a message or a or/par structure*

    outputs: M_3, *a processed SPPC element with all of its clauses role dependent on each other*

    if $(M_1\ is\ a\ message)\&\&(M_2\ is\ a\ or/par\ structure)$

        $process\_msg\_or/par(M_1, M_2, M_3)$

    if $(M_1\ is\ a\ or/par\ structure)\&\&(M_2\ is\ a\ message)$

        $process\_or/par\_msg(M_1, M_2, M_3)$

    if $(M_1\ is\ a\ or/par\ structure)\&\&(M_2\ is\ a\ or/par\ structure)$

        $process\_or/par_o r/par(M_1, M_2, M_3)$

---

procedure $process\_msg\_or/par(M_1, M_2, M_3)$

    inputs: M_1, M_2, *two SPPC elements which is a message and a or/par structure repectively*

    outputs: M_3, *a processed SPPC element from* M_1 *with all of its clauses role dependent on each other*

    initiate a list (L)

    for $(the\ first\ element\ (E)\ of\ each\ branch\ of\ M_2)$

        if $(M_1\ and\ E\ doesn't\ contain\ at\ least\ one\ same\ role)$

            generate a new message (TM) *using recevier of* M_1 *as sender and sender of E as receiver*

            put TM in L

    $M_3 = M_1$

    while $(L\ is\ not\ empty)$

        fetch the first element (M_5) *in L*

        $M_3 = M_4\ then\ M_5$

procedure *process_or/par_msg*(M_1, M_2, M_3)

    inputs: M_1, M_2, *two SPPC elements which is a or/par structure and a message repectively*

    outputs: M_3, *a processed SPPC element from M_1 with all of its clauses role dependent on each other*

    for *(each branch(B) of M_1)*

        pre − precessor(B, B_1)

        retrieve the last element(E) of B_1

        if *(E is not a invoke(mid))&&(E and M_2 doesn't contain at least one same role)*

            generate a new message(TM) using recevier of E as sender and sender of M_2 as receiver

            replace E in B_1 with "E then TM"

        put B_1 in L

    fetch the first element(M_5) in L

    $M_3 = M_5$

    while *(L is not empty)*

        fetch the first element(M_6) in L

        $M_3 = M_3 \, or/par \, M_6$

---

procedure *process_or/par_or/par*(M_1, M_2, M_3)

    inputs: M_1, M_2, *two SPPC elements which are two or/par structures*

    outputs: M_3, *a processed SPPC element from M_1 with all of its clauses role dependent on each other*

    initiate two lists(L, L_1)

    for *(each branch(B) of M_1)*

        pre − precessor(B, B_1)

        retrieve the last element(E) of B_1

        for *(the first element(E_1) of each brach of M_2)*

            if *(E is not a invoke(mid))&&(E and E_1 doesn't contain at least one same role)*

                generate a new message(TM) using recevier of E as sender and sender of E_1 as receiver

                put TM in L

            $M_6 = E$

        while *(L is not empty)*

            fetch the first element(M_5) in L

            $M_6 = M_6 \, then \, M_5$

        replace E with M_6 in B_1

        put B_1 in L_1

    fetch the first element(B_2) in L_1

    $M_3 = B_2$

    while *(L_1 is not empty)*

        fetch the first element(B_3) in L

        $M_3 = M_3 \, or/par \, B_3$

Figure 3.5: 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 is 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 ($C\,then\,M_1\,then...then\,M_n$).
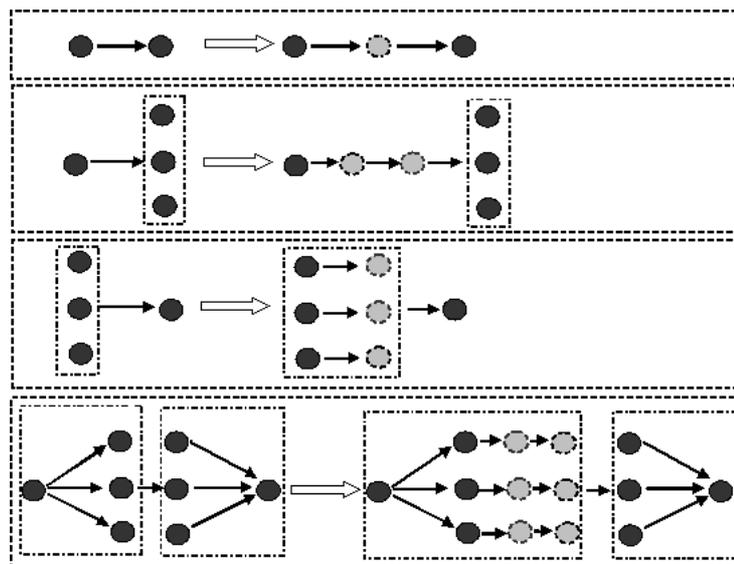
Figure 3.6: Inserting connect message for different SPPC structure

The diagram shown in Figure 3.6 illustrates the above message insert principles, where solid cycles represent the existing SPPC message node, arrows represent *then* operator and dashed cycles represent the new message nodes that need to be inserted in the SPPC model.

In LCC, the only way to represent loops is through use of 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:

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

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

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 a SPPC model, we use the combination of *invoke(mid)* and *msg(mid,...* to represent a loop, which has to be translated into a LCC compatible fashion. The loop processing algorithm in Figure 3.7 shows how to pre-process the all the loops defined in a SPPC model.

---

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)

Figure 3.7: Algorithm For pre-processing all the loops defined in a SPPC model

The algorithm for generating a LCC protocol from a processed SPPC model is shown in Figure 3.8.

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 $(a(R_{i,i \in I}, ID_i))$ *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$(SM_1, LM_i, List)$

Figure 3.8: Algorithm for deriving a LCC protocol from a SPPC model

3.9 A Simple Case Study

We will use a simple example to illustrate how our framework is used to verify an IP protocol(LCC) based on a business process model. The business process model in Figure 3.9 shows an very simple printing process from the view of the *Sales* role. The
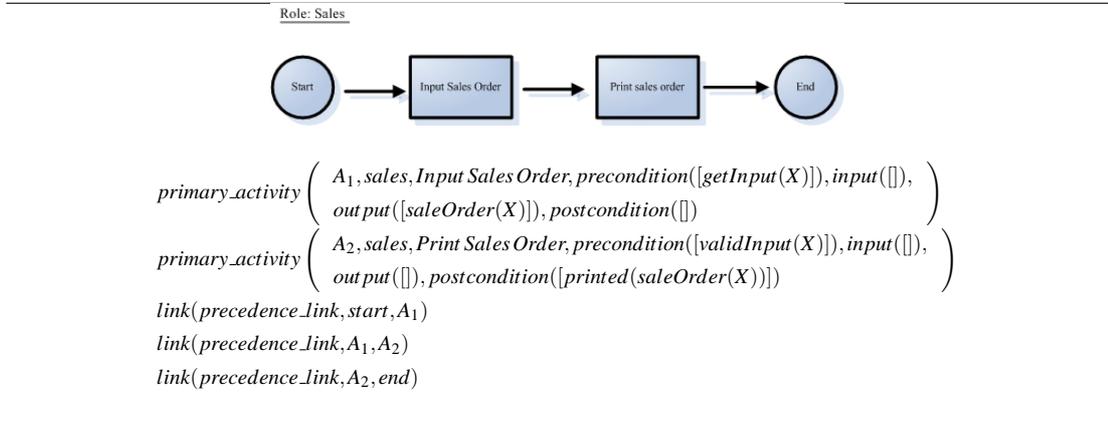
Figure 3.9: Sales order printing process

formal representation of the business process model is defined above. There are two primary activities in this model which are *input_sales_order* and *print_sales_order* and each of them has several functional properties as shown in table 3.4. The business

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

Table 3.4: Functional Properties of Primary Activities in Sales Order Printing Process

process model is represented as follows using linear temporal logic: A possible multi-

$associate(sales, getInput(X)),$

$associate(sales, validInput(X)),$

$associate(sales, saleOrder(X))$

$associate(sales, printed(saleOrder(X))),$

$clause(A_1, associate(sales, getInput(X)) \wedge associate(sales, validInput(X)) \rightarrow \Diamond associate(sales, saleOrder(X))),$

$clause(A_2, associate(sales, saleOrder(X)) \rightarrow \Diamond associate(sales, printed(saleOrder(X)))),$

agent platform that is used to realise the above business process model is illustrated
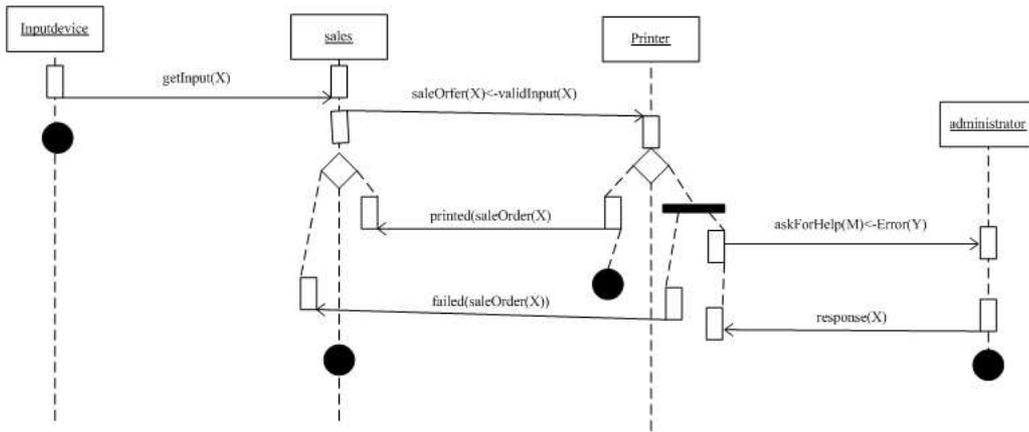
Figure 3.10: AUML model for sales order printing process

in Figure 3.10 using an AUML model which expresses the message passing sequence between several agents.

The SPPC model for this scenario is given below. It might be noticed that in the

$msg(m.1, mb(getInput(X)), sender(a(i1, Inputdevice)), receiver(a(s1, Sales)))$

then

$msg(m.2, pre(validInput(X)), mb(saleOrder(X)), sender(a(s1, Sales)), receiver(a(p1, Printer)))$

then

$$
\left(
\begin{array}{l}
msg(m.3, mb(printed(saleOrder(X))), sender(a(p1, Printer)), receiver(a(s1, Sales))) \\
or \\
\left(
\begin{array}{l}
msg(m.4, pre(err(Y)), mb(askForHelp(Help)), sender(a(p1, Printer)), receiver(a(a1, Admin))) \\
then \\
msg(m.5, mb(response(Help)), sender(a(a1, Admin)), receiver(a(p1, Printer))) \\
then \\
msg(m.6, mb(failed(saleOrder(X))), sender(a(p1, Printer)), receiver(a(s1, Sales))) \\
then \\
invoke(m.2)
\end{array}
\right)
\end{array}
\right)
$$

SPPC model shown above there are several agents which are not defined in the example business process model. The reason for this is that when we build a multi-agent system we try to use the existing agents as much as possible instead of building new ones to fit the input business process models every time. Thus, we have to consider the availabilities of the agents we needed and the capabilities of those agents. The scenario we try to handle is that all the agents that coordinate to perform the system are already available. In our example, agents *Inputdevice, Sales, Printer, Admin* are picked up by MAS protocol modeler to perform the intended business process model. The linear temporal logic clause derived from the above SPPC model is:

$$(associate(inputDevice, getInput(X)) \lor associate(sales, getInput(X))) \to \Diamond$$
$$\left(\begin{array}{l}(associate(Sales, vaildInput(X)) \to \Diamond(associate(Sales, saleOrder(X)) \lor associate(Printer, saleOrder(X)))) \to \Diamond \\ \left(\begin{array}{l}(associate(Printer, printed(saleOrder(X))) \lor associate(Sales, printed(saleOrder(X)))) \to \Diamond \\ \left(\begin{array}{l}associate(Pinter, err(Y)) \to \Diamond \left(\begin{array}{l}associate(Printer, askForHelp(Help)) \lor \\ associate(Admin, askForHelp(Help))\end{array}\right) \to \Diamond \\ (associate(Adminresponse(Help)) \lor associate(Printer, response(Help))) \to \Diamond \\ \left(\begin{array}{l}(associate(Printer, failed(saleOrder(X))) \lor associate(sales, failed(saleOrder(X)))) \to \Diamond \\ \left(associate(Sales, validInput(X)) \to \Diamond \left(\begin{array}{l}associate(Sales, saleOrder(X)) \lor \\ associate(Printer, saleOrder(X))\end{array}\right)\right)\end{array}\right)\end{array}\right)\end{array}\right)$$

The linear temporal logic representations derived from the example business process model and SPPC model are then matched using the algorithm proposed in section 3.7. The final LCC protocol derived from SPPC model after the verification is as follows:

$$a(i1, Inputdevice) :: getInput(X) \Rightarrow a(s1, Sales) then a(i1, Inputdevice).$$

$$a(s1, Sales) :: getInpux(X) \Leftarrow a(i1, Inputdevice) then a(loop(X), Sales).$$

$$a(loop(X), Sales) :: saleOrder(X) \Rightarrow a(Printer, p1) \leftarrow validInput(X) then$$
$$\left(\begin{array}{l}(printed(saleOrder(X)) \Leftarrow a(p1, Printer) then a(s1, Sales)) or \\ \left(failed(saleOrder(X)) \Leftarrow a(p1, Printer) then a(loop(X), Sales)\right)\end{array}\right).$$

$$a(p1, Printer) :: saleOrder(X) \Leftarrow a(loop(X), Sales) then$$
$$\left(\begin{array}{l}printed(saleOrder(X)) \Rightarrow a(loop(X), Sales) or \\ \left(\begin{array}{l}askForHelp(Help) \Rightarrow a(a1, Admin) then \\ response(Help) \Leftarrow a(a1, Admin) then \\ failed(saleOrder(X)) \Leftarrow a(loop(X), Sales) then a(p1, Printer)\end{array}\right)\end{array}\right).$$

$$a(a1, Admin) :: askForHelp(Help) \Leftarrow a(p1, Printer) then response(Help) \Rightarrow a(p1, Printer)$$

3.10 Prototype Implementations

The prototyping system for supporting the proposed framework is demonstrated in this section. It helps system modellers to build SPPC models, check the relationships of the functional properties defined in the input process model and then translate the SPPC model to LCC protocol automatically. The system consists of three parts which are discussed in the following sub-sections.

3.10.1   SPPC modeller

SPPC modeller is used for building a SPPC model. It provides a graphical interface, by which a SPPC model can be build using graphical notation and then can be translated into linear temporal logic clauses for verification purpose using verifier or can be translated into a concrete LCC protocol using LCC protocol generator. The snapshot of it is given in Figure 3.11. This unit is adopted from the general graphical modelling tool INGENIAS *http://ingenias.sourceforge.net/*.
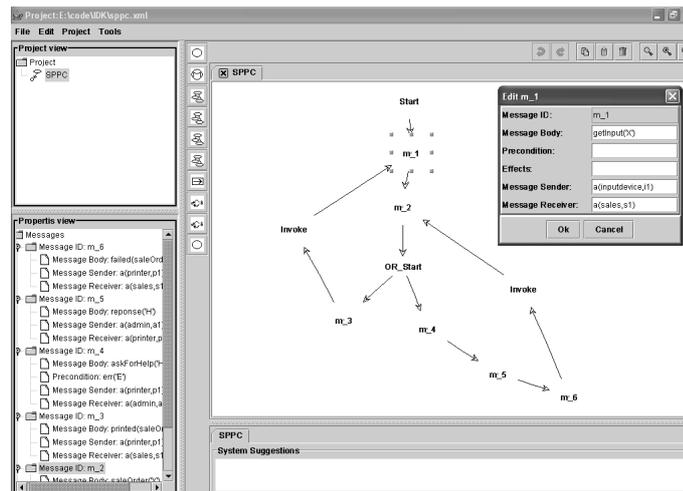
Figure 3.11: Business process model based MAS protocol IDE

A SPPC model generated as a graph by this definition tool can be converted into XML format and saved as an XML file automatically as shown in Figure 3.12

Figure 3.12: XML representation of a SPPC Model
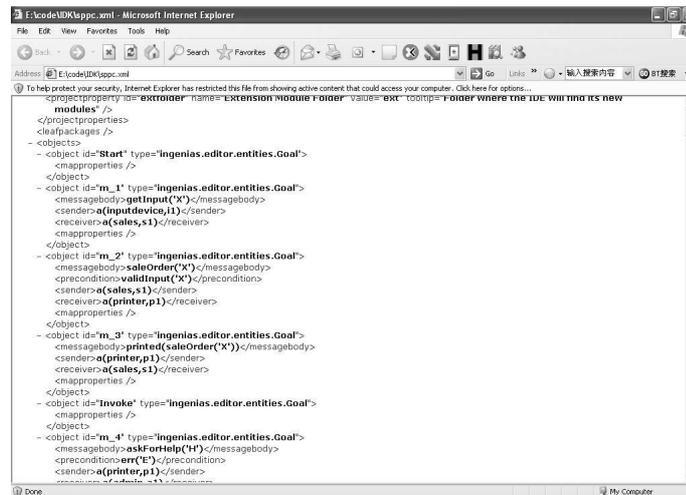
3.10.2   Verifier

Verifier is responsible for matching the two sets of linear temporal logic clauses derived. It takes a business process model and a SPPC model as its inputs, translates them into linear temporal representations respectively and then tries to match them. The matching process is carried by automated theorem proving. We use SISCtus Prolog (http://www.sics.se/isl/sicstuswww/site/index.html) for the implementation of the verifier. All the linear temporal logic clauses derived from a business process model and a SPCC model can be directly mapped to the facts in prolog. The verifier interacts with the SPPC modeller using SISCtus Jasper. Figure 3.13 shows the verification of the SPPC model that we used in case study section.
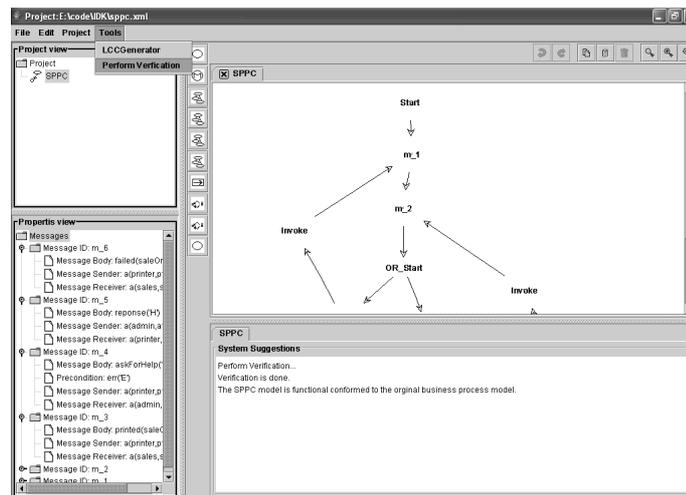
Figure 3.13: Verification of a SPPC model

3.10.3 LCC protocol generator

The LCC protocol generator is used to generate LCC protocols from given SPPC models. After a SPPC model is verified by the verifier, it can be translated automatically to a LCC protocol. The snapshot of LCC generator is shown in Figure 3.14.
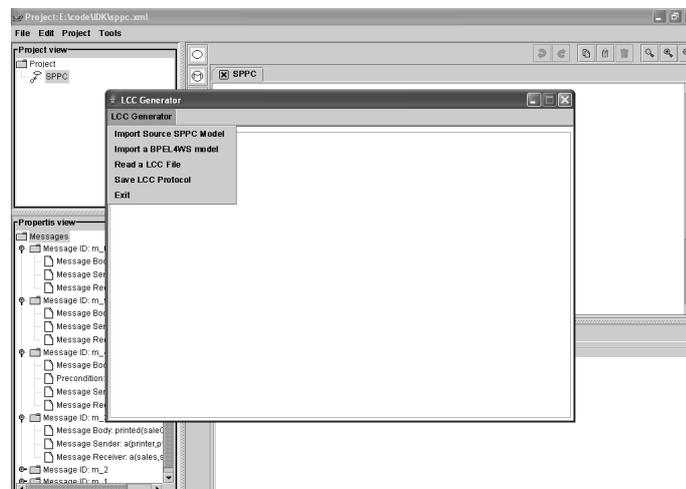
Figure 3.14: LCC protocol generator

3.11 Discussion

With our framework, the temporal relationships of all the functional properties defined in a business process model can be checked when those properties are used in the

property checking model. Therefore, the system modeller no longer needs to worry about what process the functional properties are associated with, but instead, he/she only needs to care about the temporal relations between the properties being used in his/her protocols. Although in the final protocol it is still hard to get an overall view of the business transactions defined in the given process model, the protocol does perform the task that the process model intends to do. Thus high level business process models can be bridged down to IP and can lead to many totally different IP by different system modellers according their preferences.

With our framework, the process model is used as requirement and all the properties defined in it will be verified computationally. It helps the system modeller to generate the right design at the end of the IP modelling process. Thus, in the late testing phase, we only need to concentrate on system level errors without considering any problems from the requirement level, which largely reduces the amount of properties that need to be checked and thus promotes the efficiency of IPs construction.

In an evolutional environment (where requirements change), our framework has an advantage over other software engineering approaches. Any change to the requirements(process models) will be immediately reflected in the temporal logical representation so that inconsistencies between the new process model and old property checking model can be discovered and fixed by a system modeller according to the system's suggestions.

3.12   Summary

In this section, We propose a framework for modeling multi-agent system protocols starting from a high level process model. With our framework, a process model can be used as a base for protocol properties' verification. A simple language SPPC is defined for property checking purposes and any protocol model defined by SPPC can be translated into an existing protocol language(in this case LCC). Using our framework, much effort can be saved in the process of MAS protocol modeling since requirements level errors can be discovered using automatic verification, which is different with the typical protocol modeling engineering method. Furthermore, using our approach, any revision to an existing protocol can also be checked in real time to make sure all the business logic level changes are correct and compatible with the former specification.

Chapter 4

Using Executable Formal BPMs For MAS Development Via Language Mapping

Chapter 3 discussed how to bridge from high level business process models to multi-agent interaction protocols so that existing formal business process models can be adopted in the development of multi-agent system. As well as high level business process models, executable business process models have been well developed and used in conventional workflow management system. Based on the three layer conceptual framework introduced, executable business process models and multi-agent interaction protocols are at the same conceptual level (implementation level) as shown in Figure 4.1:
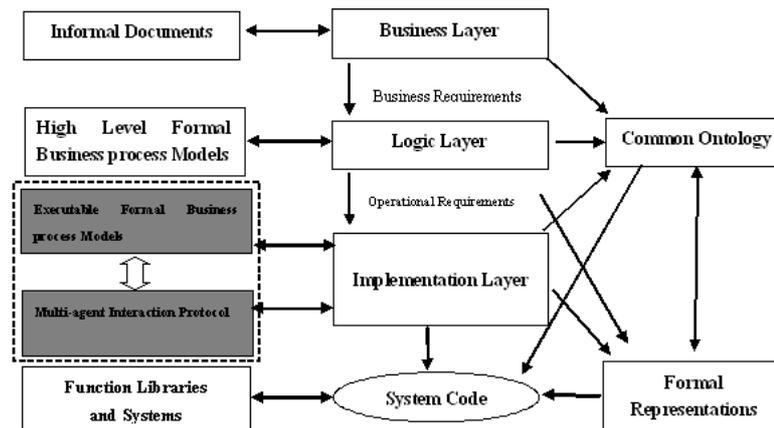

Figure 4.1: From executable formal BPMs to IPs

Since executable BPMs are at the same conceptual level as MAS IPs, they con-

tain enough information on both business level requirements and system level require-
ments to make automatic generation of MAS IPs possible. In this chapter, we will
discuss how to connect executable business process models and MAS IPs via language
mapping technique in detail by performing it between two concrete specification de-
scription languages (BPEL4WS and LCC). In section 4.1, the necessary background
knowledge of BPEL4WS is given. Section 4.2 gives the detailed syntax translation
from BPEL4WS to SPPC using language mapping techniques, in which the main con-
cepts that are involved in almost all conventional business process modelling languages
are considered. A simple case study is given in section 4.3 to help in understanding of
the approach proposed in this chapter. The problems that we encountered during the
language mapping are discussed in section 4.4.

4.1   Background Knowledge Of BPEL4WS

The Business Process Execution Language for Web Services (abbreviated to BPEL4WS)
is a notation for specifying business process behaviour based on Web Services. Pro-
cesses in BPEL4WS export and import functionality by using Web Service interfaces
exclusively. Business processes can be described in two ways. Executable business
processes model actual behaviour of a participant in a business interaction. Business
protocols, in contrast, use process descriptions that specify the mutually visible mes-
sage exchange behaviour of each of the parties involved in the protocol, without reveal-
ing their internal behaviour. The process descriptions for business protocols are called
abstract processes. BPEL4WS is meant to be used to model the behaviour of both ex-
ecutable and abstract processes. It provides a language for the formal specification of
business processes and business interaction protocols. By doing so, it extends the Web
Services interaction model and enables it to support business transactions. BPEL4WS
defines an inter-operable integration model that should facilitate the expansion of au-
tomated process integration in both the intra-corporate and the business-to-business
spaces. BPEL4WS fits into the core Web service architecture since it is built on top
of XML[XML06a], XML Schema[XML06b], WSDL[WSD01], and UDDI[UDD02].
Some of the key BPEL4WS syntax is given and explained below:

- $<$ *partners* $>$: contains a list of participants (web services) involved as part of
  the BPEL4WS workflow

- $<$ *variables* $>$: contains the variables that are used in the workflow

- $< invoke >$: invoke a particular service as requested

- $< receive >$: receive a service invocation message

- $< reply >$: reply a message to service requestor

- $< assign >$: assigns a value that might be from a received message to a variable

- $< sequence >$: executes the activities nested within it in a sequential order

- $< flow >$: executes the activities nested within it concurrently

- $< switch >$: executes the activities nested within one of the branches defined in it, when the condition for that branch holds during the execution

- $< while >$: implements a loop when the conditions defined for the loop hold

Currently, BPEL4WS is well accepted by industry and has become a de facto standard for deploying web services based distributed workflow system and that is way it is chosen for our work as we try to get our research as close as possible to real life applications. Some new softwares that are built based on BPEL4WS have been released, such as BPEL4WS Java Runtime (BPWS4J) (http://www.alphaworks.ibm.com/tech/bpws4j) platform.

We use a simple example[1] to show how a BPEL4WS execution model is constructed. In this example (see Figure 4.2), a customer sends a request for a loan; the request gets processed, and the customer finds out whether the loan was approved. Initially, the middle step will involve sending the application to a Web services enabled financial institution and telling the customer what it decided. From the customer's point of view, the process will consume his application and then send him an answer. The diagram below shows this external view of the loan request process.

BPEL4WS compositions rely heavily on WSDL (http://www.w3.org/TR/wsdl) descriptions of the involved services in order to refer to the messages being exchanged, the operations being invoked, and the portTypes these operations belong to. For any BPEL4WS process, we will need the description of the appropriate information and the process itself. After all the requirements are now available for creating the process. we begin the definition with the $< process >$ element, and include the namespaces that will allow it to refer to the required WSDL information, where the message

---

[1]This example is taken from http://www-128.ibm.com/developerworks/library/ws-bpelcol1/
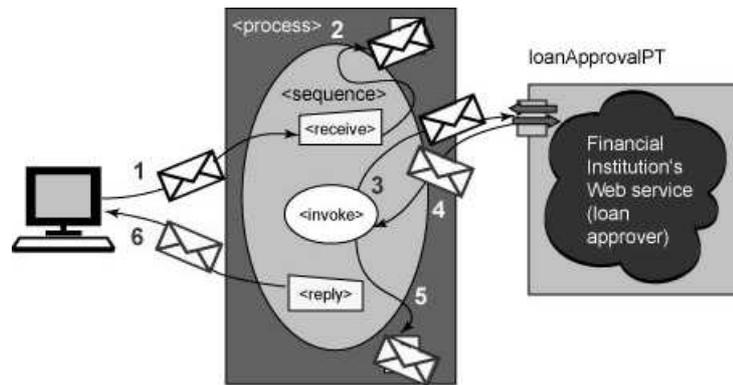
Figure 4.2: Executable loan approval process.

definitions are defined (http://..../loandefinitions), the target namespace of the loan ap-
prover (http://.../loanapprover), and the target namespace of the process's own WSDL
(http://.../loan-approval). The process is now able to use the loan approver service as
a component. The next step is to declare the parties involved. Named partners are de-
fined, each characterised by a WSDL serviceLinkType. For this example, the partners
are the customer and the financial institution. The myRole/partnerRole attribute on a
partner specifies how the partner and the process will interact given the serviceLink-
Type. The myRole attribute refers to the role in the serviceLinkType that the process
will play, whereas the partnerRole specifies the role that the partner will play. This is
illustrated in the partner definitions below. The loan approval process offers the func-
tionality of the loanApprovalPT to the customer, and the financial institution in turn of-
fers that functionality to the process. This relationship can be seen in Figure 4.2 above.
A process may contain only one activity, which in this case will be the $< sequence >$.

$< process\,name = "loanApprovalProcess"$

$\qquad targetNamespace = "http://acme.com/simpleloanprocessing"$

$\qquad xmlns = "http://schemas.xmlsoap.org/ws/2002/07/business-process/"$

$\qquad xmlns:lns = "http://loans.org/wsdl/loan-approval"$

$\qquad xmlns:loandef = "http://tempuri.org/services/loandefinitions"$

$\qquad xmlns:apns = "http://tempuri.org/services/loanapprover" >$

$\quad < partners >$

$\qquad < partner\,name = "customer"$

$\qquad\quad serviceLinkType = "lns:loanApproveLinkType"$

$\qquad\quad myRole = "approver"/ >$

$\quad < /partners >$

The sequence contains a receive activity that can take the customer's message. The

definition of a receive activity must include the partner that will send it its message, and the port type and operation of the process to which the partner is targeting this message. Based on this information, once the process gets a message, it searches for an active receive activity that has a matching partner-portType-operation triplet and hands it the message. In order to avoid confusion, the specification states that there may not be multiple received activities with the same partner-portType-operation triplet that are active at the same time. We start the sequence activity, and add the receive to it. The next step is to ask the Web services-enabled financial institution whether or not it will accept the loan. This is done with a regular Web services invocation, defined in the process by an Invoke activity. When this activity runs it will make the specified invocation to the Web service using the message in its input container, put the answer it gets into its output container, and end. Note that the call will be made on the "approver" partner to perform the approve operation. In order for the process to respond to the customer's request, it uses a Reply activity. Once a reply activity is reached, the partner-portType-operation triplet it contains is used to figure out whom to send the reply to. Therefore, in order to reply to the message that arrived through the $< receive >$ activity, we would need a Reply activity with the same triplet. In this case, we want to tell the customer what the financial institution decided, so the message to be sent will be found in the output container of the invoke: approvalInfo.

```
< sequence >
    < receive name = "receive1" partner = "customer" portType = "apns : loanApprovalPT"
            operation = "approve" container = "request" createInstance = "yes"/ >
    < invoke name = "invokeapprover" partner = "approver"
            portType = "apns : loanApprovalPT" operation = "approve"
            inputContainer = "request" outputContainer = "approvalInfo"/ >
    < reply name = "reply" partner = "customer" portType = "apns : loanApprovalPT"
            operation = "approve" container = "approvalInfo"/ >
< /sequence >
< /process >
```

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

4.2.1 Problem Analysis

If we consider the interaction 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 $< myRole >$ or $< partnerRole >$ and a coordinating agent (substitute for a workflow server) that is defined implicitly in BPEL4WS.

Although a conventional BPEL4WS process model based system can be understood as a multi-agent system, the responsibility given to the coordinating agent (workflow server) as addressed in the previous section is too heavy and, correspondingly, is too light on service providing agent on the contrary. This is understandable because BPEL4WS was initially designed for the coordination of web services which only have very limited computing capabilities. However, with agents that have stronger computing capabilities, the burden on the coordinating agent can be shared, which gives us the possibility of eliminating the coordinating agent. If we can dispatch the tasks that are performed by the workflow server (coordinating agent) based on conventional client-server architecture to service providing agent, the process models that are used in conventional workflow system then can be used in multi-agent based system. Thus, to enable the MAS based distributed workflow system, the first step is to decide what sorts of tasks are performed by the workflow server and how they can be dispatched to agents. Figure 4.3 shows the minimum components that are required by a BPEL4WS based conventional workflow server and those components should have correspondences in new system. As introduced earlier, the main concern of LCC
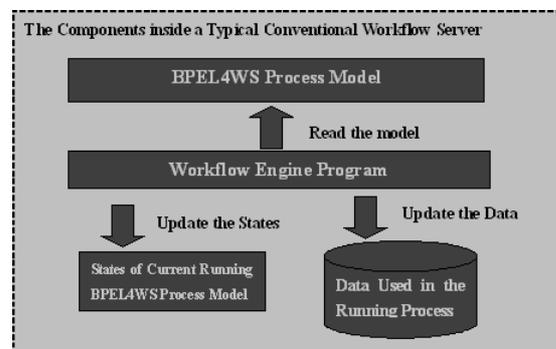
Figure 4.3: The components of a typical conventional workflow server

protocol based MAS is the production of protocol rather than the design of agents (actually, in a LCC based MAS, the agents are usually dummy agents). If we can find a approach that keeps those components in the LCC protocol, BPEL4WS model based workflow systems can then be deployed on a functionally equivalent MAS platform. This mechanism is illustrated in Figure 4.4:
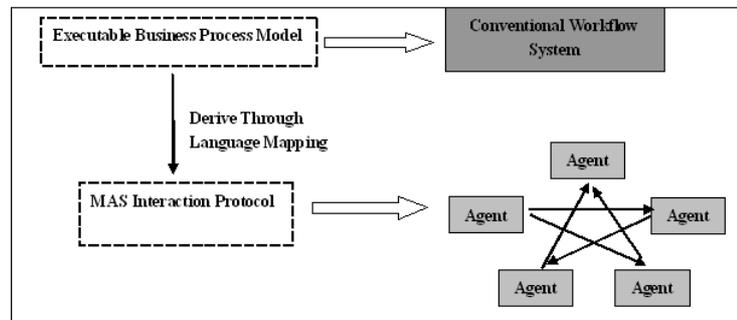
Figure 4.4: Connecting workflow systems and multi-agent systems via language mapping

Figure 4.5 shows the correspondences between conventional workflow server components and the LCC framework, from which we can see that all the components of
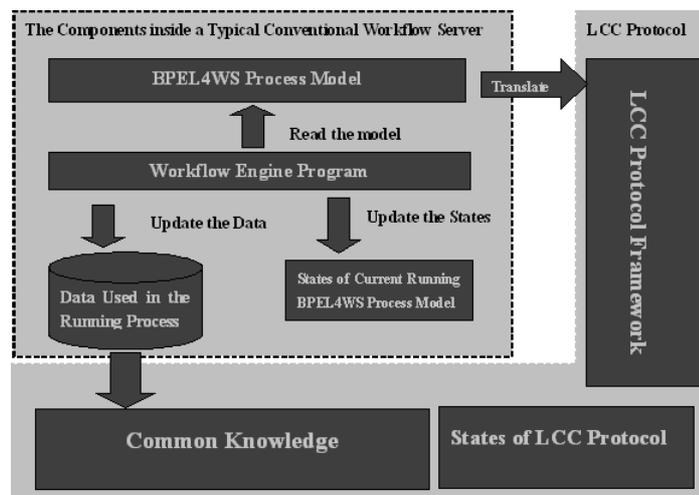
Figure 4.5: Correspondence between LCC protocol and conventional workflow server's components

a conventional workflow server are retained in a LCC based multi-agent system. The correspondence of "workflow engine program" is the program that each agent uses for processing the received LCC protocols. The states of running business process instances is corresponding to the states of LCC protocol instances. BPEL4WS model and the data used for the running workflow instance can be replaced by LCC protocol

framework and common knowledge. From the above analysis, we can see that it is possible to use a LCC based MAS for distributed workflow deployment.. We will then discuss how to connect this two systems that are based on different architecture.

4.2.2   Why choose language mapping?

The most widely used technique for connecting two different specification based systems is through syntax based language mapping. After the two languages, which are used for describing the specifications in different system, are mapped, specifications that are written in one language can be translated into another automatically and can thus be used in another system.

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 all these sorts of activities are all undertaken 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). Actually, if P_1 and P_2 know the information of each other and know the intended order of messages being passed, the workflow server is not required at all for the message forwarding purpose since as agents, they could communicate directly. Structure activities define only the execution order of basic activities and if the IP protocol language (such as LCC) has the syntax to describing such information, the 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 easily be dispatched to distributed agents. The following BPEL4WS partial model shows the problem, for simplicity, it is expressed using plain text:

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

Three roles are defined in the above chunk (P_1, P_2 are defined explicitly and the workflow server is defined 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/P_2 wholly or separately. To decide deterministically which agent should execute what computing activity is impossible without more information. Therefore, in order to translate a BPEL4WS specification to a LCC protocol, the BPEL4WS model must be specified in a more stylised way, 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.

Based on the above analysis, we can see that all the tasks performed in a BPEL4WS based conventional workflow system can be completely or partially realised by a MAS also if the BPEL4WS specification is represented by the MAS interaction protocol especially in LCC. In the following sections, we will discuss in detail how to perform the language mapping from BPEL4WS to LCC.

4.2.3   Performing language mapping from BPEL4WS to SPPC

We observe that syntactically, a BPEL4WS model is close to a SPPC model but is relatively far away from LCC protocol. We have designed algorithms in chapter 3 to generate LCC protocols from SPPC models. Therefore, if we can translate a BPEL4WS model into a SPPC model first, it can be then translated into LCC protocol directly using our existing algorithms. We will discuss in this section how to translate the notations, as classified above, from BPEL4WS to SPPC.

4.2.3.1 Translation from partners defined in BPEL4WS to SPPC roles

All the participants defined in $< partnerLinks >$ in a BPEL4WS model can be directly mapped to the agents (sender and receiver) no matter whether they are $< myRole >$ or $< partnerRole >$. For example, for the loan approval scenario that we used, the definition of all participants involved in process is:

$$< partners >$$
$$< partner \, name = "customer"$$
$$serviceLinkType = "lns : loanApproveLinkType"$$
$$myRole = "approver"/ >$$
$$< /partners >$$

From the above definition, we know there are two roles participating in the interaction and in the derived SPPC protocol, we will have two agents $(a(customer, ID)$, $a(approver, ID_1))$ directly related to them.

4.2.3.2 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. The complete definition for it from BPEL4WS is:

$$< receive \, partnerLink = "ncname" \, portType = "qname" \, operation = "ncname"$$
$$variable = "ncname"? createInstance = "yes|no"?$$
$$< /receive >$$

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:

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

One point that needs to be mentioned here is that 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, proper response will be sent back to the right agent. This constraint is used to represent the 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.

$$< reply\, partnerLink = "ncname"\, portType = "qname"\, operation = "ncname"$$
$$variable = "ncname"?\, faultName = "qname"?$$
$$< /reply >$$

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 defined in BPEL4WS.

- The $< invoke >$ construct allows the business process to invoke a one-way or request-response operation on a portType offered by a partner.

$$< invoke\, partnerLink = "ncname"\, portType = "qname"\, operation = "ncname"$$
$$inputVariable = "ncname"?\, outputVariable = "ncname"?$$
$$$$

The corresponding SPPC might be one message or two messages connected by "then", which depends on wether 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, we can see that all the message passing activities ($< receive >$, $< invoke >$, $< reply >$) in BPEL4WS can be translated into SPPC messages.

4.2.3.3   Translation from computing activities defined in BPEL4WS to SPPC constraints

The computing activities defined in BPEL4WS are $< assign >$, $< wait >$ etc. Since the translating principle for all of them are the same, we only discuss the translation of $< assign >$ in detail. The activity $< assign >$ in BPEL4WS specification defines internal variables assignation in the BPEL4WS workflow engine and it gives BPEL4WS computational ability.

$$< assign >$$
$$< copy > +$$
$$from - spec$$
$$to - spec$$
$$< /copy >$$
$$< /assign >$$

In SPPC, constraints (post-conditions and pre-conditions) are the places where the concrete computation takes place. Therefore, the computation carried by the centralised server, as addressed earlier, should be dispatched to the agents in the multi-agent system as constraints. Eventually, which agent execute what constraints doesn't matter too much. 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 consideration of the translation of structure activities also.

4.2.3.4 Translation from structure activities defined in BPEL4WS to SPPC model

BPEL4WS structure activities control the execution orders between the activities (message passing activities, computing activities and structure activities) that are nested within them explicitly from the point of view of activities. SPPC, however, uses operators to control the temporal orders between message clauses and the temporal order between computing clauses and message clauses is 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 translated into SPPC using operators and when they should be represented by the combinations of messages and pre-conditions/post-conditions in SPPC, are 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.

$$< sequence\ standard - attributes >$$
$$standard - elements$$
$$\textbf{activity}+$$
$$< /sequence >$$

If we only consider message passing activities and structure activities, it is quite simple to derive a SPPC model from it since BPEL4WS and SPPC have similar notation for sequence. However, it becomes much more complex when the computing activities are considered since we have to decide how the computing activities should be used as pre-conditions/post-conditions of SPPC messages during the translation with the initial time order defined in $< sequence >$ kept. 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 $< sequence >$, certain re-write rules have to be applied:

$$(A_1 \, then \, A_2) \Rightarrow (E_1 \to C) \qquad if \quad A_1 \xrightarrow{A_1 \, is \, a \, message \, passing \, activity} E_1, \qquad (rule_1)$$
$$A_2 \xrightarrow{A_2 \, is \, a \, computing \, activity} C$$

$$(A_1 \, then \, A_2) \Rightarrow (C_1 \wedge C_2) \qquad if \quad A_1 \xrightarrow{A_1 \, is \, a \, computing \, activity} C_1 \qquad (rule_2)$$
$$A_2 \xrightarrow{A_2 \, is \, a \, computing \, activity} C_2$$

$$(A_1 \, then \, A_2) \Rightarrow (C_1 \to E_2) \qquad if \quad A_1 \xrightarrow{A_1 \, is \, a \, computing \, activity} C_1 \qquad (rule_3)$$
$$A_2 \xrightarrow{A_2 \, is \, a \, structure \, activity} E_2$$

$$(A_1 \, then \, A_2) \Rightarrow ((E_1 \, then \, E_2) \, or \, E_2) \qquad if \quad A_1 \xrightarrow{A_1 \, is \, <while> \, activity} E_1 \qquad (rule_4)$$
$$A_2 \xrightarrow{A_2 \, is \, not \, a \, computing \, activity} E_2$$

$$(C_1 \to (E_1 \, or/par...or/par \, E_n)) \Rightarrow (E_i \, or/par...or/par \, E_{i+n}) \quad if \quad (C_1 \to E_1) \Rightarrow E_i, ..., (C_1 \to E_n) \Rightarrow E_{i+n} \quad (rule_5)$$
$$((E_1 \, or/par...or/par \, E_n) \to C_1) \Rightarrow (E_i \, or/par...or/par \, E_{i+n}) \quad if \quad (E_1 \to C_1) \Rightarrow E_i, ..., (E_n \to C_1) \Rightarrow E_{i+n} \quad (rule_6)$$

$C_1 \to A_1$ and $A_1 \to C_1$ 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 $< squence >$ can be used as the pre-condition/post-condition of the SPPC message that is derived from the message passing activity. E represents the possible SPPC clauses that are derived from non-computing BPEL4WS activities. 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 4.6.

Using the algorithm, it should be noticed that a computing activity can never be used as the last element of a $< sequence >$ activity as discussed earlier. Otherwise, the translation is not possible. Therefore, not all the existing BPEL4WS models can be directly translated into SPPC models using this language mapping approach.

- The <switch> construct allows you to select exactly one branch of activity from

---

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 of Sequence and CL*

    while (P_1 *is not pointing to the last element of 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*$_1$

Figure 4.6: Algorithm for deriving a SPPC model from a BPEL4WS < *sequence* > activity

a set of choices.

$$< switch\ standard − attributes >$$
$$standard − elements$$
$$< case\ condition = "bool − expr" > +$$
$$activity$$
$$< /case >$$
$$< otherwise >?$$
$$activity$$
$$< /otherwise >$$
$$< /switch >$$

In < *switch* >structure, each < *case* > can possibly has four kinds of direct child elements: basic activities (message passing activities and computing activities), < *sequence* > structure, < *switch* > structure, < *flow* > structure and < *while* > structure. The execution of each branch is controlled by *condition*(*Co*) defined for each < *case* >. A < *switch* > activity can be represented using SPPC "or" notation in the following format,

$$(Co_1 → A_1)\, or\, (Co_2 → A_2)\, or\, ...$$

where C_i means the conditions defined for < *case* > in < *switch* > and A_i rep-

resents 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 $< case >$ is depended on the types of them. The *condition* defined for each $< case >$ can be translated using the following re-write rule together with the rules defined for $< sequence >$:

$$(Co_1 \rightarrow A_1) \Rightarrow (Co_1 \rightarrow E) \quad if \quad A_1 \Rightarrow E \qquad\qquad (rule_7)$$

The algorithm for translating a BPEL4WS $< switch >$ activity to a SPPC model is given in Figure 4.7.

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$_1$

Figure 4.7: Algorithm for deriving a SPPC model from a BPEL4WS $< switch >$ activity
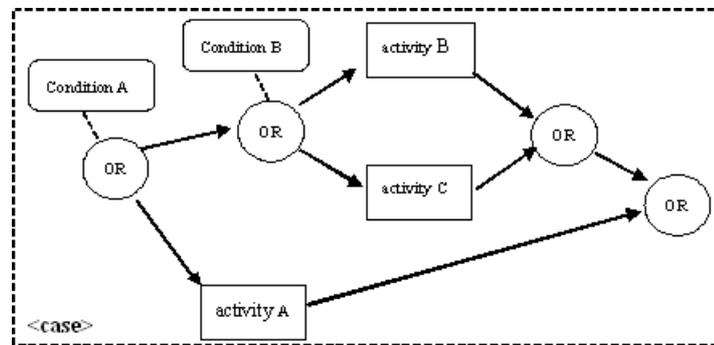
Figure 4.8: Diagrammatical representation of a $< case >$ in $< switch >$

Figure 4.8 shows the diagrammatical representation of a $< case >$ defined in a $< switch >$ activity. After applying our algorithm, the diagrammatical representation of the generated SPPC model is shown in Figure 4.9 in which pre-conditions are translated from the conditions and message A, B and C are translated from the activity A, B and C defined in $< case >$ in Figure 4.8.
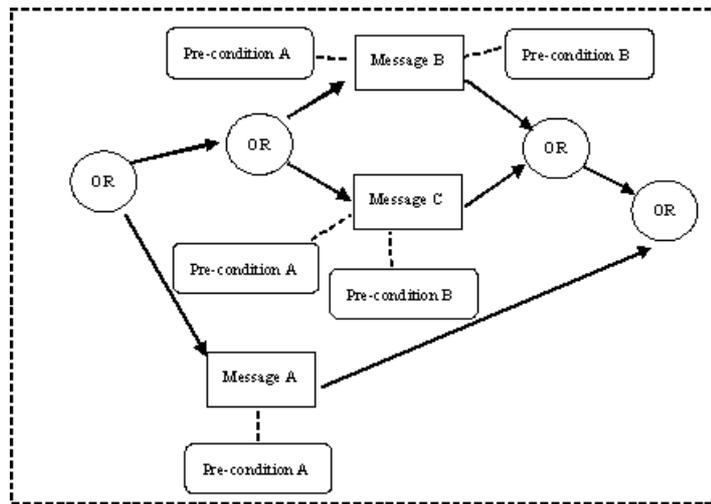
Figure 4.9: Processed diagrammatical representation of the $< case >$

- The $< flow >$ construct allows us to specify one or more activities to be performed concurrently. Links can be used within concurrent activities to define arbitrary control structures.

$$< flow \, standard - attributes >$$
$$< links >?$$
$$< link \, name = "ncname" > +$$
$$< /links >$$
$$activity+$$
$$< /flow >$$

A $< flow >$ activity creates a set of concurrent activities directly nested within it. It further enables expression of synchronisation dependencies between activities that are nested directly or indirectly within it. The link construct is used to express these synchronisation 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 specifies a source element specifying the link's name and the target of the link specifies 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.

$$
\begin{aligned}
&< flow > \\
&\quad < links > \\
&\qquad < link\,name = "CtoD"/ > \\
&\quad < /links > \\
&\quad < sequence\,name = "Y" > \\
&\qquad < receive\,name = "C"... > \\
&\qquad\quad < source\,linkName = "CtoD"/ > \\
&\qquad < /receive > \\
&\qquad < invoke\,name = "E".../ > \\
&\quad < /sequence > \\
&\quad < invoke\,partnerLink = "D"... > \\
&\qquad < target\,linkName = "CtoD"/ > \\
&\quad \\
&< /flow >
\end{aligned}
$$

In a conventional client-server based workflow system, the execution of concurrent activity and control of the synchronisation link are possible because the workflow server can control the state of all the branches 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 4.10 shows the algorithm that we use to turn all the synchronisation links defined in a $< flow >$ activity into SPPC messages. Using this algorithm, a $< flow >$ activity can 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

```
procedure translate_flow(Flow, CL, SPPC)
    input: Flow, the BPEL4WS < flow > 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(L₁) in L)
         scan the whole Flow and find out the activity(A) that defines the < source > of L₁
         scan the whole Flow and find out the activity(A₁) that defines the < target > of L₁
         replace A with "A then A″₂ in which A₂ is a message sended from the receiver of A to sender of A₁
         replace A₁ with "A₃ then A″₁ in which A₃ is a message sended from the receiver of A to sender of A₁
    for (All the branches(B) of FLow)
         translate_structure_activity(B, CL, SPPC₁)
         SPPC = SPPC par SPPC₁
```

Figure 4.10: Algorithm for deriving a SPPC model from a BPEL4WS $< flow >$ activity

until a certain success criteria has been met.

$$< while\ condition = "bool − expr"\ standard − attributes >$$
$$standard − elements$$
$$activity$$
$$< /while >$$

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 as shown in Figure 4.11. Whether the loop is executed is controlled by the precondition defined for the M_1. However, a BPEL4WS $< while >$ activity represented
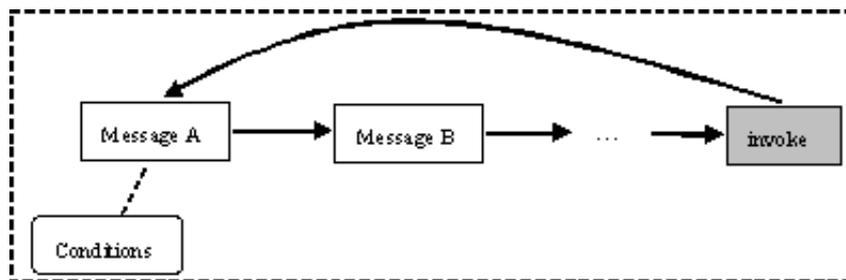
Figure 4.11: Diagrammatical representation of a SPPC loop

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 as illustrated in Figure 4.14.
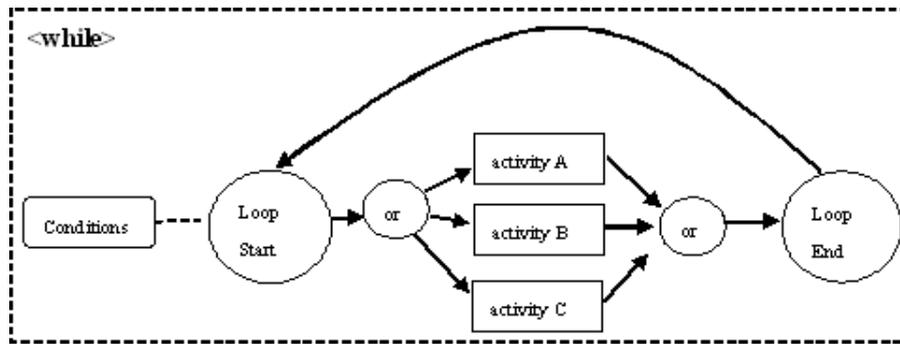
Figure 4.12: Diagrammatical representation of a $< while >$ activity
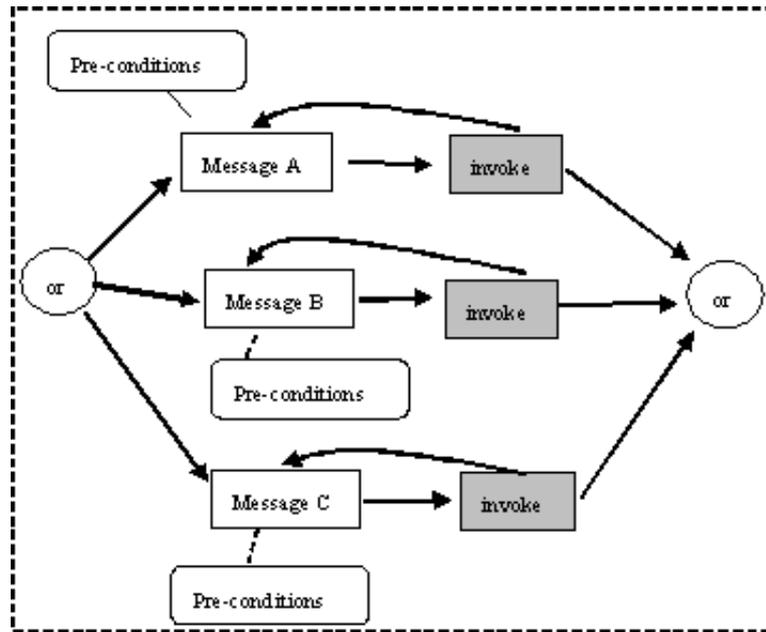
Figure 4.13: Diagrammatical representation of a SPPC model that is equivalent to the $< while >$ activity in Figure 4.12

The SPPC model that is semantically equivalent to the $< while >$ in Figure 4.12 is shown in Figure 4.13 in which the pre-conditions for message A,B and C are derived from the condition define for $< while >$ and message A,B and C correspond to the activity A,B and C in Figure 4.12. In addition, at the end of message A, B and C in the SPPC model, three invokes are added (invoke(A),invoke(B) and invoke(C)) to represent the repeated execution of the three messages. The above example shows the basic idea of translating a $< while >$ into SPPC clauses, which 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 defined for it. Several re-write rules are thus defined for different sorts of content

$$
\begin{array}{lll}
(A_1 \, then...then \, A_i) \Rightarrow & if \quad A_1 \Rightarrow E_1,...,A_i \Rightarrow E_i & (rule_8) \\
(E_1 \, then...then \, E_n) & \quad (E_i \rightarrow generate\_invoke(E_1)) \Rightarrow E_n &
\end{array}
$$

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

$$
\begin{array}{lll}
(E_i \rightarrow generate\_invoke(E_1)) \Rightarrow (E_i \, then \, E_2) & if \quad generate\_invoke(E_1) \Rightarrow E_2 & (rule_{10})
\end{array}
$$

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

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

By applying the above re-write rules, a $<while>$ activity can be represented using SPPC notations. The algorithm for $<while>$ activity translation is given in Figure 4.14.

---

procedure $translate\_while(While, CL, SPPC)$

    input: $While$, the $BPEL4WS$ $<while>$ $activity$

        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 $(each \, branch(B) \, of \, E \, connected \, by \, "or/par")$

            $invoke\_generator(B, Invoke_1)$

            $Invoke = Invoke \, or/par \, Invoke_1$

$$
\begin{aligned}
&\textbf{procedure}\, loop\_generator(SPPC_1, Invoke, SPPC_2)\\
&\quad \textbf{input:} SPPC_1, SPPC\, clauses\\
&\qquad\qquad Invoke, a\, invoke\, or\, sets\, of\, invokes\, connected\, by\, "or/par"\\
&\quad \textbf{output:} SPPC_2, SPPC\, clauses\, that\, represent\, loop\\
&\quad SPPC_2 = SPPC_1\\
&\quad extract\, the\, last\, element(E)\, of\, SPPC_2\\
&\quad \textbf{if}\,(E\, is\, a\, SPPC\, message)\\
&\qquad replace\, it\, with\, "E\, then\, Invoke"\\
&\quad \textbf{else}\\
&\qquad \textbf{for}\,(each\, branch(B)\, of\, E)\\
&\qquad\qquad loop\_generator(SPPC_2, Invoke, SPPC_3)\\
&\qquad\qquad SPPC_2 = SPPC_3
\end{aligned}
$$

Figure 4.14: Algorithm for deriving a SPPC model from a BPEL4WS $<while>$ activity

4.3 A Simple Case Study

In the previous sections, we discussed how the fundamental notations of a business process model (basic activity and temporal order between basic activities(sequence, or, parallel and loop)) can be translated into SPPC clauses using some of the BPEL4WS syntaxes through language mapping. We use a simple example to show how a SPPC can be derived from a BPEL4WS specification, which starts from a example workflow encoded in BPEL4WS. The example workflow described below consumes two parameters, a stock symbol and a country name. The result of the workflow is a quote for the stock localised into the currency of the given country. It has been simplified by removing attributes that do not help clarify the example.[2]

```
< process >
  < partners >
    < partner name = "requestor"/ >
    < partner name = "stockProvider"/ >
    < partner name = "currencyProvider"/ >
    < partner name = "simpleProvider"/ >
  < /partners >
  < variables >
    < variable name = "request"/ >
    < variable name = "response"/ >
    < variable name = "stockRequest"/ >
    < variable name = "stockResponse"/ >
    < variable name = "currencyRequest"/ >
    < variable name = "currencyResponse"/ >
```

---

[2] The original scenario for this example is taken from [BV04].

```
                < variable name = "simpleRequest"/ >
                < variable name = "simpleResponse"/ >
           < /variables >
           < sequence >
           < receive portType = "request" partner = "requestor" operation = "requestLookup" variable = "request" >
           < /receive >
           < assign >
             < copy >< from variable = "request"/ >< to variable = "stockRequest"/ >< /copy >
             < copy >< from variable = "request"/ >< to variable = "currencyRequest"/ >< /copy >
           < /assign >
           < flow >
             < invoke, portType = "getStockQuote" partner = "stockProvider" operation = "getQuote"
                 inputVariable = "stockRequest" outputVariable = "stockResponse" >
             < /invoke >
             < invoke portType = "getExchangeRate" partner = "currencyProvider" operation = "getRate"
                 inputVariable = "currencyRequest" outputVariable = "currencyResponse" >
             < /invoke >
           < /flow >
           < assign >
             < copy >< from variable = "stockResponse"/ >< to variable = "simpleRequest"/ >< /copy >
           < /assign >
           < invoke portType = "multiplyFloat" partner = "simpleProvider" operation = "multiply"
                 inputVariable = "simpleRequest" outputVariable = "simpleResponse" >
           < /invoke >
           < assign >
             < copy >< from variable = "simpleResponse"/ >< to variable = "response"/ >
             < /copy >
           < /assign >
           < reply portType = "request" partner = "requestor"
                 operation = "requestLookup" variable = "response" >
           < /reply >
           < /sequence >
        < /process >
```

Figure 4.15, provides a graphical view of the structure of the workflow. Internally,
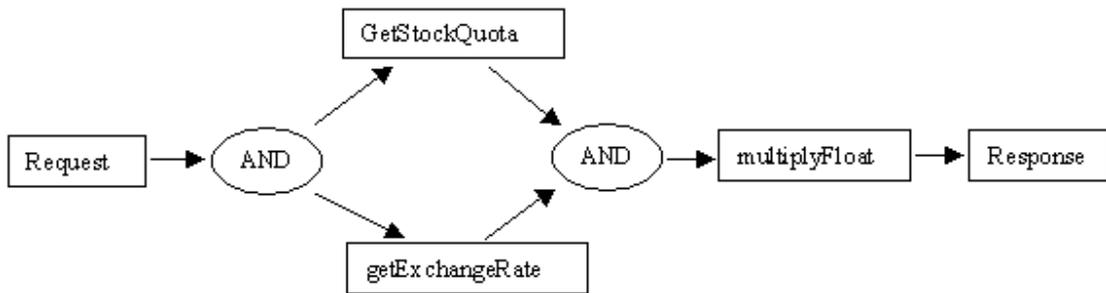

Figure 4.15: Stock lookup process

the workflow definition coordinates the interaction of the five participants named: requestor, stockProvider, currencyProvider, simpleProvider the role of workflow engine itself (called mainServiceProvider for simplicity). The SPPC model derived from it is:

$$msg(mid_1, pre([\,]), mb(request : requestLookup : request), sender(a(requestor, ID)), receiver(a(mainServiceProvider, ID_1))$$

$then$

$$\left(\left(\left(msg \begin{pmatrix} mid_2, pre([stockRequet = request, currencyRequest = request]), \\ mb(getStockQuote : getQuote : stockRequest), \\ sender(a(mainServiceProvider, ID_1)), receiver(a(stockProvider, ID_2)) \end{pmatrix} \right. \right. \right.$$

$then$

$$\left. msg \begin{pmatrix} mid_3, pre([stockRequet = request, currencyRequest = request]), \\ mb(getStockQuote : getQuote : stockRequest : stockReponse), \\ sender(a(stockProvider, ID_2)), receiver(a(mainServiceProvider, ID_1)) \end{pmatrix} \right)$$

par

$$\left(msg \begin{pmatrix} mid_4, pre([stockRequet = request, currencyRequest = request]), \\ mb(getExchangeRate : getRate : currencyRequest) \\ sender(a(mainServiceProvider, ID_1)), receiver(a(currencyProvider, ID_3)) \end{pmatrix} \right.$$

$then$

$$\left. \left(msg \begin{pmatrix} mid_5, pre([stockRequet = request, currencyRequest = request]), \\ mb(getExchangeRate : getRate : currencyRequest : currencyReponse) \\ sender(a(currencyProvider, ID_3)), receiver(a(mainServiceProvider, ID_1)) \end{pmatrix} \right) \right)$$

$then$

$$\left(msg \begin{pmatrix} mid_6, pre([simpleRequest = stockResponse]), mb(multiplyFloat : multiply : simpleRequest), \\ sender(a(mainServiceProvider, ID_1, ID_1)), receiver(a(simpleProvider, ID_4)) \end{pmatrix} \right.$$

$then$

$$\left. msg \begin{pmatrix} mid_7, pre([\,]), mb(multiplyFloat : multiply : simpleRequest : simpleResponse), \\ sender(a(simpleProvider, ID_4)), receiver(a(mainServiceProvider, ID_1,) \end{pmatrix} \right)$$

$then$

$$msg \begin{pmatrix} mid_8, pre([response = simpleResponse]), mb(request : requestLookup : response), \\ sender(a(mainServiceProvider, ID_1)), receiver(a(requestor, ID)) \end{pmatrix}$$

This SPPC model can thus be translated into LCC protocol using the algorithm proposed in chapter 3.

$$a(mainServiceProvider, ID_1) :: request : requestLookup : request \Leftarrow a(requestor, ID)$$

$then$

$$\left(\begin{pmatrix} getStockQuote : getQuote : stockRequest \Rightarrow a(stockProvider, ID_2) \\ \leftarrow (stockRequest = request) \, and \, (currencyRequest = request) \\ then \\ getStockQuote : getQuote : stockRequest : stockResponse \Leftarrow a(stockProvider, ID_2) \end{pmatrix} \right.$$

par

$$\left. \begin{pmatrix} getExchagneRate : getRate : currencyRequest \Rightarrow a(currencyProvider, ID_3) \\ \leftarrow (stockRequest = request) \, and \, (currencyRequest = request) \\ then \\ getExchagneRate : getRate : currencyRequest : currencyResponse \Leftarrow a(currencyProvider, ID_3) \end{pmatrix} \right)$$

$then$

$$\begin{pmatrix} multiplyFloat : multiply(simpleRequest) \Rightarrow a(simpleProvider, ID_4) \leftarrow (simpleRequest = stockResponse) \\ then \\ multiplyFloat : multiply : simpleResponse \Leftarrow a(simpleProvider, ID_4) \end{pmatrix}$$

$then$

$$response \Rightarrow a(\_, \_) \leftarrow (response = simpleResponse)$$

$a(stockProvider, ID_2) :: getStockQuote : getQuote : stockRequest \Leftarrow a(mainServiceProvider, ID_1)$

$\quad\quad\quad then$

$\quad\quad\quad getStockQuote : getQuote : stockRequest : stockResponse \Rightarrow a(mainServiceProvider, ID)$

$a(currencyProvider, ID_3) :: getExchangeRate : getRate : currencyRequest \Leftarrow a(mainServiceProvider, ID_1)$

$\quad\quad\quad then$

$\quad\quad\quad getExchangeRate : getRate : currencyRequest : currencyResponse \Rightarrow a(mainServiceProvider, ID_1)$

$a(simpleProvider, ID_4) :: multiplyFloat : multiply : simpleRequest \Leftarrow a(mainServiceProvider, ID_1)$

$\quad\quad\quad then$

$\quad\quad\quad multiplyFloat : multiply : simpleRequest : simpleResponse \Rightarrow a(mainServiceProvider, ID_1)$

$a(requestor, ID) :: request : requestLookup : request \Rightarrow a(mainServiceProvider, ID_1)$

$\quad\quad\quad then$

$\quad\quad\quad request : requesetLoopkup : request : response \Leftarrow a(mainServiceProvider, ID_1)$

4.4 Summary

In this chapter, we discussed how to develop protocol based multi-agent systems using executable business process models. Language mapping is performed between a business process modelling language (BPEL4WS) and a IP (LCC) to generate the protocol used in a MAS from a business process model. Since the gap between them is large, we use a another modelling language (SPPC) as an intermediary. Thus a SPPC model derived from a BPEL4WS specification can be translated into LCC protocol automatically using the existing algorithm.

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 the business process modelling language cannot be seamlessly represented by a another modelling language which is based on different paradigm. For example, the computing activities nested at the end of a $< sequence >$ activity in BPEL4WS can not be easily translated in to 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 restrictions mean only some BPEL4WS specifications (those conforming to the language mapping rules) can be used for interaction protocol guided MAS development, which makes the approach discussed in this chapter incomplete. In fact, such language mapping based completeness is very hard to achieve (even for particular business process modelling languages) since different business process modelling languages and protocol modelling languages may be based on different computing paradigms.

Chapter 5

A Novel Approach of Using Executable Formal BPMs For MAS Development

In chapter 4, we discussed the automatic generation of LCC protocol from BPEL4WS specification using syntax based language mapping technique and concluded that such an approach can only provide partial solution for the problem (using executable business process model for MAS development) because of missing of a implicit role workflow server in MAS and also the gap between the computing paradigms of two different languages is too great.

Therefore, in this chapter we propose another approach for our work: producing a LCC protocol, which acts as BPEL4WS interpreter. The BPEL4WS specification and the LCC protocol (BPEL4WS interpreter) are passed together between the agents to enable their coordination. BPEL4WS specification defines the tasks that agents need to perform and the LCC protocol tells agents how to interpret BPEL4WS specifications received. Based on this idea, a BPEL4WS specification that is defined in any fashion can be interpreted neatly by LCC protocol when they are passed together in the multi-agent system.

5.1 Agent Coordination Using LCC Protocol and BPEL4WS Specification

From the purely technical point of view, a BPEL4WS model is nothing but a XML document that is composed by certain syntaxes which can be understood by computing software. The BEPL4WS workflow engine is software that is designed and imple-

mented to understand the syntax used in a BPEL4WS specification and is used process them to perform tasks described. In a MAS, if each agent is given the knowledge of how to process the BPEL4WS document and if the states of running process instance are provided, the centralised workflow server is not needed any more at least for executing the business process model. There are two ways to give an agent capability to perform task:

1. embedding the business process model processing capability inside the agent which means the agent knows how to do things when it is initially created. This is the way of how the conventional workflow is implemented and we are not interested in this approach since making each agent additive to particular business process modelling language loses generality.

2. assigning the capability to the agent dynamically, which means the agent can only has the ability of performing certain tasks, for instance, processing BPEL4WS models, when it is given such knowledge at run time. This approach, compared with the first one, is more generic because MAS is simply used as a platform to provide pure distributed architecture and is separated from particular application (workflow management) deployed on it. Therefore, for our work, we concentrate on this approach.

The crucial issue that we need to consider for the second approach then is how we can dynamically assign capability to agent. It is noticed that one of the design principles of LCC is to specify and to tell agents what to do and how to do the tasks specified. Therefore, if we can use LCC protocol to tell agents in MAS how to process BPEL4WS, BPEL4WS specifications can be used directly in MAS. In this way, LCC protocol acts as a language interpreter, which understands all the BPEL4WS syntaxes and their semantics, and this interpreter is given to each agent dynamically during their interaction. In agents' interactions, BPEL4WS is used to tell agents what are the currently requested tasks and LCC protocol tells how to perform the tasks specified by BPEL4WS. In addition, the states of the BPEL4WS model attached in a LCC protocol are also recorded. Figure 5.1 shows the new correspondences between the components of a conventional workflow server and a LCC protocol, from which we can clearly see that all the components that compose a workflow server can be assumed by a LCC protocol. In the other words, with these correspondences, the LCC protocol passed between agents gives agents the ability of acting as a "conventional workflow server" at the moment when they hold the received message packages.
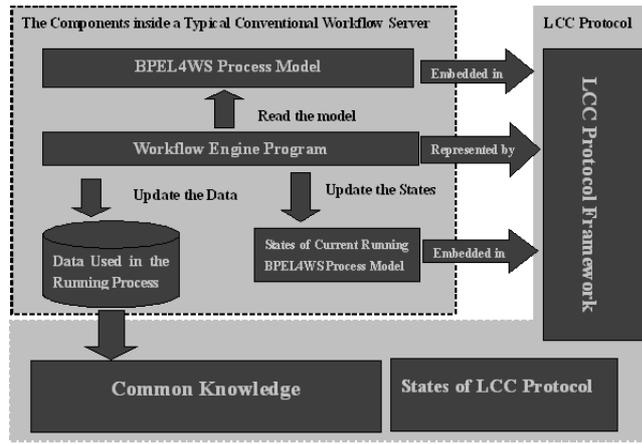
Figure 5.1: The correspondence between the components of the conventional workflow server and LCC

The infrastructure of the system based on the new idea is given in Figure 5.2. Based on this infrastructure, the multi-agent interaction protocol, BPEL4WS specification and interacting messages are packed and passed together between the agents. Once an agent receives the package, it processes: the incoming message (initiating appropriate behaviours), interaction protocol and BPEL4WS (resolving the next action it needs to take), then it sends out a new package to the next agent to continue the coordination. Besides LCC protocol, BPEL4WS model and messages, the package that is passed between agents in MAS also contains all the values of the variables that are used for the attached BPEL4WS model, which means the storage of data is also decentralised.
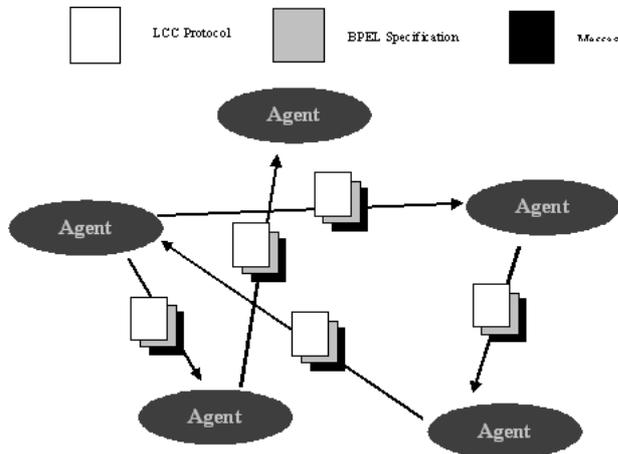

Figure 5.2: The infrastructure of our generic MAS platform

5.2 Interpreting BPEL4WS specification Using LCC protocol

Normally, a LCC protocol framework is written based on the roles ($a(Role, ID)$) involved in the potential interaction. The *Role* defined reflects the semantic of real role in the application domain (customer, seller etc.). When an agent receives a LCC protocol, it checks for the LCC clauses defined for its role (*Role*) and extracts the next action it needs to perform from the LCC definition. However, for a BPEL4WS specification based MAS interaction, the way of writing LCC protocol is quite different. Each role defined in the LCC protocol framework does not correspond to the *application role* anymore but to a BPEL4WS syntax that is named its *BPEL4WS syntax role*. The *application role* of agent is used as one of the arguments of *BPEL4WS syntax role* defined in the form $a(BPELSyntax(Arguments), ID)$, where *BPELSyntax* corresponds to the BPEL4WS syntax and *Arguments* represents five arguments that are used for every *BPELSyntax* role:

- *Model*: is a part of BPEL4WS model and represents the tasks that need to be processed.

- *MList*: stores all the unprocessed parts of a BPEL4WS model. *MList* is used to mark the states of the BPEL4WS model being processed. The BPEL4WS specification is organised in a tree structure with its branches formed by the structure activities and nodes formed by the basic activities (message passing activities and computing activities). The tree structure is processed using a depth first search algorithm from left to right when it is passed between agents. Once a BPEL4WS message passing activity (leaf of a tree) is reached while an agent processes the BPEL4WS model (treated as a tree), the agent starts a new dialogue based on the activity and all of the unprocessed BPEL4WS model stored in *MList* has to be passed to the next agent.

- *VList*: is the place where all the concrete values of the variables that are used in workflow enactment are stored. In the centralised environment, all the information about the variables are controlled by the central server, whereas in the distributed environment, all such information has to be passed around.

- *IDList*: is used to connect a receive activity and its corresponding reply activity. This parameter is designed to fit the BPEL4WS in particular to keep to semantic

of the pair of $< receive >$ and $< reply >$ activities.

- *Role*: represents the participants (application roles) in the interaction defined by $< partnerLink >$ from BPEL4WS.

From the coordination point of view, each agent in our system is a generic agent. When an agent receives a BPEL4WS model from the others, it doesn't know the type of the BPEL4WS model; neither can it choose the right *BPELSyntax* role to process the received model. Therefore, we must provide a mechanism to help agents recognise the type of received BPEL4WS model that is requested to be processed. To serve this purpose, two general roles $a(receiver(Role), ID)$ and $a(interpreter(Arguments), ID)$ are defined in LCC protocol besides *BPELSyntax* roles. Role $a(receiver(Role), ID)$ is taken by an agent whenever it receives a package from the others. When an agent is in the role $a(receiver(Role), ID)$ the incoming messages that it can recognise can only be of two forms:

$$message(run\_this, Model, MList, VList, IDlist, Role) \qquad (form\,1)$$
$$message(web\,service\,invocation\,message, Model, MList, VList, IDList, Role) \quad (form\,2)$$

The only difference between these two forms is the first element defined. For form one, the first element "*run_this*" means the receiver of this message should process the model that is defined by the second element (*Model*) of this message while for form two, the first element of it contains all the information for a web service's invocation. Agent should perform a web services invocation before starting process the *Model*. According to the different incoming messages, the agent may perform two type of operations in role $a(receiver(Role), ID)$:

- It provides a service to the requestor if the incoming message is a service request message (contains information for web services' invocation) and may also process the received BPEL4WS model. The only case for this operation is the incoming message is from an agent that is in the role of $a(invoke(Arguments), ID)$. Since according to the BPEL4WS definition, $< invoke >$ activity is the one and the only one that carries real web service computation and may generate response on the service provider side.

- It processes the received BPEL4WS model (*Model*) only.

The LCC definition for $a(receiver(Role), ID)$ is given as follows:

$$
\begin{aligned}
&a(receiver(Role), ID) :: \\
&\quad message(M, Model, MList, VList, IDList, Role) \Leftarrow a(AnyRole, ID_1) \\
&\quad then \\
&\quad \left(\begin{array}{l}
a(interpreter(Model, MList, VList, IDList, Role), ID) \leftarrow M =" run\_this" \\
or \\
\left(\begin{array}{l}
message(M_1, Model, MList, VList_1, IDList, AnyRole) \Rightarrow a(AnyRole, ID_1) \\
\quad \leftarrow \left(\begin{array}{l} AnyRole =" invoke" \ and \ has\_output(Model) \\ and \ perform(M_1, M_2) \ and \ update\_variable(M_2, VList, VList_1) \end{array} \right) \\
or \\
a(interpreter(Model, MList, VList_1, IDList, Role), ID) \leftarrow \left(\begin{array}{l} perform(M_1, M_2) \\ and \ update\_variable(M_2, VList, VList_1) \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{aligned}
$$

The above LCC clauses indicate that when an agent receives a package in the role of *receiver*, it first processes the message (M) in the package. If M is "*run_this*", the agent then changes its role to *interpreter* to process the attached BPEL4WS model. If M is a web service invocation message from the others, the agent first performs the required service and then forwards the result of service invocation to the service requestor if there is a returned result from the web services that was just invoked. Otherwise, the agent changes its role to *interpreter* and starts processing the BPEL4WS model it currently holds.

Several constraints are defined in the above clauses also.

- *Perform*(M_1, M_2): performs the real service invocation on the requested services according to the incoming message M_1 by agent and returns the result M_2 from the service.

- *has_output*(*Model*): checks if the BPEL4WS activity (indicated by *Model*) has a *outputVariable*. It is usually used with $< invoke >$ activity only.

- *update_variable*($M, VList, VList_1$): is used to update the value of the variable involved in M in $VList$.

The complete definitions for all the constraints written in Prolog that we defined for our LCC interpreter can be found in appendix C and in later discussion, we will ignore the low level technical details of them.

Role *interpreter* is used to help agent recognise the type of BEPL4WS model it received and accordingly make it change its role properly to process the current BPEL4WS model.

The LCC definition for $a(interpreter(Model, MList, VList, IDList, Role), ID)$ is

$$a(interpreter(Model, MList, VList, IDList, Role), A_1) ::$$

$$\begin{pmatrix} a(invoke(Model, MList, VList, IDList, Role), A_1) \leftarrow is\_invoke(Model, Role) \\ or \\ message(run\_this, Model, MList, VList, IDList, Roel_1) \Rightarrow a(receiver(Role_1), A_2) \\ \quad \leftarrow is\_invoke(Model, Role_1) \end{pmatrix}$$

$$or$$

$$\begin{pmatrix} a(receive(Model, MList, VList, IDList, Role), A_1) \leftarrow is\_receive(Model, Role) \\ or \\ message(run\_this, Model, MList, VList, IDList, Roel_1) \Rightarrow a(receiver(Role_1), A_2) \\ \quad \leftarrow is\_receive(Model, Role_1) \end{pmatrix}$$

$$or$$

$$\begin{pmatrix} a(reply(Model, MList, VList, IDList, Role), A_1) \leftarrow is\_reply(Model, Role) \\ or \\ message(run\_this, Model, MList, VList, IDList, Roel_1) \Rightarrow a(receiver(Role_1), A_2) \\ \quad \leftarrow is\_reply(Model, Role_1) \end{pmatrix}$$

$$or$$

$$a(sequence(Model, MList, VList, IDList, Role), A_1) \leftarrow is\_sequence(Model)$$

$$or$$

$$a(flow(Model, MList, VList, IDList, Role), A_1) \leftarrow is\_flow(Model)$$

$$or$$

$$...$$

The constraints *is_receive/reply/...* play important roles in *interpreter*'s definition. They are the real functions that perform BPEL4WS model recognition. If the BPEL4WS model being processed is a message passing activity, the agent also needs to check if its own application role matches the role required by the activity. If so, the agent starts processing the activity or, if not, the agent forwards the BPEL4WS model to another agent whose application role matches the required role. The mechanism by which the agents search/locate each other is not a research issue of this thesis and is assumed to be available. In the following sub-sections, we will explain in detail how to use LCC protocol for interpreting the main BPEL4WS syntax.

5.2.1 Interpreting BPEL4WS Message Passing Activities Using LCC Protocol

The only way for the agents to coordinate with each other in a multi-agent system is through message passing. Therefore, when adopting a BPEL4WS specification in a multi-agent system, the first thing we need to do is to relate the BPEL4WS syntax to

message passing. The relations between BPEL4WS message passing activities and LCC messages are shown in table 5.1. The rationale for the the translations in table

BPEL4WS Message Passing Activities	LCC Messages
$< receive\ partner =''\ R''\ portType =''\ P''\ operation =''\ O''\ variable =''\ V''/>$	$message(P:O:V,...) \Leftarrow a(R,ID)$
$< invoke\ partner =''\ R''\ portType =''\ P''\ operation =''\ O''$ $inputVariable =''\ IV''\ outputVariable =''\ OV''/>$	$message(P:O:IV,...) \Rightarrow a(R,ID)$ then $message(P:O:IV:OV,...) \Leftarrow a(R,ID)$
$< reply\ partner =''\ R''\ portType =''\ P''\ operation =''\ O''\ variable =''\ V''/>$	$message(P:O:V,...) \Leftarrow a(R,ID)$

Table 5.1: Translations from BPEL4WS activities to LCC messages

5.1 is clear. All the agents in our system act as the proxies for web services. Thus all the information that relates web services' invocation needs to be contained in the messages that are passed between agents also. The way that agents process the incoming/outgoing message is different according to different BPEL4WS message passing activities. There are three sort of message passing activities ($< receive >, < inovke >$ and $< reply >$ in BPEL4WS as classified earlier in chapter 4. The LCC clauses for interpreting them are given below respectively.

5.2.1.1 Interpreting $< receive >$ activity Using LCC

$$
\begin{aligned}
&a(receive(Model,MList,VList,IDList,MyRole),A_1) :: \\
&\quad process\_receive\_message \left(\begin{array}{l} PartnerRole,PortType,Operation,Variable, \\ ID,VList,IDList,VList_1,IDList_1 \end{array} \right) \\
&\qquad \leftarrow PortType : Operation : Variable \Leftarrow a(PartnerRole,ID) \\
&\quad then \\
&\quad \left(\begin{array}{l} \left(\begin{array}{l} a(receive(Model,MList,VList,IDList,MyRole),A_1) \\ \quad \leftarrow \neg check\_receive(Model,PortType,Operation,Variable,PartnerRole) \end{array} \right) \\ or \left(\begin{array}{l} a(interpreter(Head,Rest,VList_1,IDList_1,\_),A_1) \\ \leftarrow check\_receive(Model,PortType,Operation,Variable,PartnerRole) \\ and\,MList = [Head|Rest] \end{array} \right) \\ or \\ null \leftarrow MList = [] \end{array} \right)
\end{aligned}
$$

If the BPEL4WS model that an agent needs to process is a $< receive >$ activity, the agent waits for an incoming message and checks if this message is the right one (A message is a right one only if it is sent from the right *partner* of current agent and it is defined with the right message type). If the message is not what the agent is waiting for, the agent keeps waiting until it receives the proper one. After an agent receives a right message, it changes its role to *interpreter* to process the unprocessed BPEL4WS

model in *MList* (the checking for the right incoming message is performed by the constraint *check_receive*(...)).

The update of *IDList* is used to record the information about the service requestor and the service they invoke, so that later on, the result of service invocation will be sent to the right agent.

5.2.1.2 Interpreting $<invoke>$ activity Using LCC

$$
a(invoke(Model, MList, VList, IDList, Role_1), A_1) ::
$$
$$
message(PortType : Operation : InputVariable, Model, MList, VList, IDList, Role_2) \Rightarrow a(receiver(Role_2), A_2)
$$
$$
\leftarrow process\_invoke(Model, PortType, Operation, InputVariable, Role_2)
$$
$$
then
$$
$$
\left(\begin{array}{l} null \leftarrow Model = ..[\_, partnerLink(\_), portType(\_), operation(\_), inputVariable(\_), \\ \qquad outputVariable(null), sourceLink(\_), targetLink(\_)] \\ or \\ \left(\begin{array}{l} message(PortType : Operation : InputVariable : OutputVariable, Model, MList, VList, IDList, Role_1) \\ \Leftarrow a(receiver(Role_2), A_2) \\ then \\ \left(\begin{array}{l} null \leftarrow MList = [] \\ or \\ a(interpreter(Head, Rest, VList_3, IDList, Role), A_1) \\ \leftarrow MList = [Head|Rest] \, and \, VList_1 = [OutputVariable, InputVariable|VList] \end{array} \right) \end{array} \right) \end{array} \right)
$$

When an agent is of the role *invoke*, it extracts the necessary information: *PortType*, *Operation* and *InputVariable* from the current BPEL4WS $<invoke>$ activity (*Model*) and sends it out to the next agent that is in the role of $a(receiver(...), ID)$ for web service's invocation. If the *outputVairable* is defined in the current $<invoke>$ activity, it will be a response from the message receiver later on. After the sender receives the response, it will changes its role to *interpreter* to continuously process the unprocessed BPEL4WS model. The constraint *process_invoke*(...) is used to extract the necessary information from $<invoke>$ activity that needs to be processed.

5.2.1.3 Interpreting $<reply>$ activity Using LCC

$$
a(reply(Model, MList, VList, IDList, myRole), A_1) ::
$$
$$
\left(\begin{array}{l} \left(Variable_1 => a(Partner, ID) \leftarrow \left(\begin{array}{l} process\_reply(Model, Partner, PortType, Operation, Variable, \_) \\ and \\ get\_ID(Partner, PortType, Operation, IDList, Variable, ID) \\ and \\ look\_up(VList, Variable, Variable_1) \end{array} \right) \right) \\ or \\ \left(Fault => a(Partner, ID) \leftarrow \left(\begin{array}{l} process\_reply(Model, Partner, PortType, Operation, Variable, Fault) \\ and \\ get\_ID(Partner, PortType, Operation, IDList, Variable, ID) \end{array} \right) \right) \end{array} \right)
$$

An agent sends a message in reply to a message that was received from $a(Partner, ID)$. The *Partner* and *ID* is stored in *IDList* to make sure that the message is sent to the right partner. Constraint *process_reply* is used to extract the necessary information from $< reply >$ activity and $get\_ID(...)$ is used to find out the corresponding service requestor's information that is previously stored in the LCC common knowledge to make sure that the result ($Variable_1$) will be sent to the right receiver. Constraint $loop\_up(VList, Variable, Variable_1)$ fetches the value ($Variable_1$) of the outgoing variable (*Variable*) defined in $< reply >$.

5.2.1.4 Interpreting $< sequence >$ activity Using LCC

$$a(sequence(Model, MList, VList, IDList), A_1) ::$$
$$a(interpreter(Model_1, [Model_2|MList], VList, IDList, Role), A_1)$$
$$\leftarrow process\_sequence(Model, Model_1, Model_2)$$

$a(sequence(Model, MList, VList, IDList), A_1)$ corresponds to the BPEL4WS $< sequence >$ activity. When an agent is in this role, it first gets the first child element $Model_1$ of *Model*, stores the left children elements $Model_2$ in *MList* and then changes its role to *interpreter* to process $Model_1$ recursively. In this way, the elements of a $< sequence >$ activity can be processed one by one in a sequential order thus keeps the semantic of $< sequence >$ from BPEL4WS.

5.2.1.5 Interpreting $< switch >$ activity Using LCC

$$a(switch(Model, MList, VList, IDList), ID) ::$$
$$a(interpreter(Model\_1, MList, VList, IDList, Role), ID) \leftarrow process\_switch(Model, Model_1)$$

The underlying principle for processing a $< switch >$ activity using a LCC protocol is all the branches ($< case >$) defined are processed one by one from the most left one to the right by the constraint $process\_switch(Model, Model_1)$. The branch whose *condition* is true (depending on current process instance) is extracted as $Model_1$ for further processing.

5.2.1.6 Interpreting $< flow >$ activity Using LCC

The $< flow >$ activity represents the concurrent execution of several activities that are nested in it. In a system that has a centralised server, it is not difficult to implement

this since all the states of the activities that are enclosed in a $< flow >$ activity can be recorded such that the time order between them can be controlled properly. But in a multi-agent system, all the activities have to be executed in a sequential order because there is no way to collect the states of all the activities without a centralised controller. So if we want to use BPEL4WS in a pure decentralised manner, we first need to represent concurrent structure activities (like $< flow >$) in sequential order without affecting the result of the process.

The algorithm for converting a flow structure to a sequence structure is based on the breath-first search. We implement this search using lists: **open** and **closed**, to keep track of progress. **open** lists states that have been generated but whose children have not been examined. The order in which states are removed from **open** determines the order of the search. **closed** records states that have already been examined. The complete algorithm is given in Figure 5.3.

---

procedure *convert_flow2sequence(Flow, FS)*

    inputs: *Flow, a $< flow >$ activity that needs to be converted*

    outputs: *FS, a $< sequence >$ activity generated from the input $< flow >$*

    initiates two lists : open = [Start], closed = []

    while *(open $\neq$ [])*

        if *(the current node is a structure activity($\mathcal{S}$))*

            expands the current node and puts all its nested activities in list open

        else if *(the current node is a basic activity($\mathcal{B}$))*

            if *($\mathcal{B}$ has neither $< sourceLink >$ or $< targetLink >$)*

                removes it from the list open and puts $\mathcal{B}$ in the list closed by FIFO principle

            else if *($\mathcal{B}$ has $< targetLink >$)*

                update the common knowledge to make the $< targetLink >$ public

                and put $\mathcal{B}$ in the list closed

            else if *($\mathcal{B}$ has $< sourceLink >$)*

                check for the corresponding $< targetLink >$ in common knowledge

              if *(the corresponding $< targetLink >$ exists)*

                  removes it from the list open and puts $\mathcal{B}$ in the list closed by FIFO principle

              else if *(the corresponding $< targetLink >$ doesn't exist)*

                ignores the current node, goes back to its parent level and

                tries the sibling of its parent in open

Figure 5.3: Algorithm for converting a $< flow >$ activity to $< sequence >$

Figure 5.4 shows the structure of a simple $< flow >$ activity, in which solid box represents the basic activity, dashed box represents the sequential structure activity and solid arrow indicate the synchronisation link defined between activities. Applying our
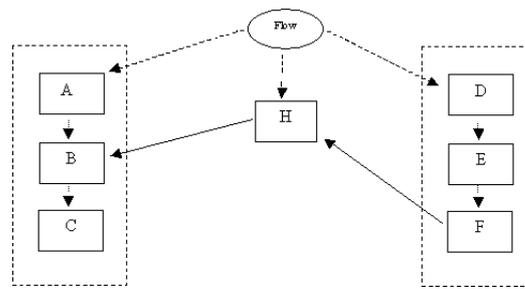
Figure 5.4: Diagrammatic representation of $< flow >$ activity

algorithm to the flow example:

1. The flow activity is expended and all its nested activities are putted into **open** ([A then B then C,H,D then E then F]).

2. The leftmost element is taken out from **open** and because it is a sequence activity and first element is then extracted and all the remaining elements are used to form a new sequence activity (B then C). The new sequence activity is then appended at the end of **open** (H,D then E then F, B then C). Because the extracted element A is a basic activity and has neither source link nor target link, it is put into **closed** ([A]).

3. H is taken out from **open**. However, it has a incoming source link. It has to be appended at the end of **open** (D then E then F, B then C, H).

4. Repeating the step 2, the contents of **open** are [B then C, H, E then F]. The content of **closed** is [A,D].

5. Repeating the step 2, because B has a incoming source link from H. There is no new element of **closed** at this stage.

6. Repeating step 3.

7. **open**:[B then C, H, F], **closed**:[A,D,E]

8. Repeating the step 2.

9. Repeating the step 3.

10. **open**:[B then C,H], **closed**:[A,D,E,F]. 11 > Repeating the step 2.

11. **open**:[B then C], **closed**:[A,D,E,F,H].

12. **open**:[C], **closed**:[A,D,E,F,H,B]

13. **open**:[], **closed**:[A,D,E,F,H,B,C]

The LCC protocol for interpreting a $< flow >$ activity is thus:

$$a(flow(Model, MList, VList, IDList, Role), ID) ::$$
$$a(interpreter(Model_1, MList, VList, IDList, Role), ID) \leftarrow process\_flow(Model, Model_1)$$

where constraint $process\_flow(Model, Model_1)$ performs the above converting algorithm and generate a $< sequence >$ activity $Model_1$.

5.2.1.7 Interpreting $< while >$ activity Using LCC

$$a(while(Model, MList, VList, IDList, \_), ID) ::$$
$$a(interpreter(Model_1, MList_1, VList, IDList, \_), ID)$$
$$\leftarrow \left(\begin{array}{l} extract\_activity(Model, Activity) \, and \\ Activity = ..[\_, Condition, Model_1] \, and \, Condition \, is \, true \\ and \, MList_1 = [Model|MList] \end{array} \right)$$
$$or$$
$$a(interpreter(Head, Rest, VList, IDList, \_), ID) \leftarrow MList = [Head|Rest]$$

When an agent processes a $< while >$ activity using the above LCC protocol, it first checks if the conditions associated with $< while >$ are satisfied and if so it extracts the direct nested activity defined in the $< while >$, changes its role to *interpreter* and starts processing it. The *MList* also has to be updated using current $< while >$ activity as its first element. The reason for this is because the child element of $< while >$ has to be processed repeatedly until the conditions no longer hold. Thus, the next agent that receives the package also has to check the conditions to decide if $< while >$ activity has to be performed again. If the conditions don't hold, the agent starts processing the first element stored in the un-processed BPEL4WS model list (*MList*).

5.3   A Simple Example

We use a simple example to illustrate how our approach works. The definition for the input BPEL4WS specification is given as follows with all the irrelevant parts ignored:

```
< process name = "loanApprovalProcess" >
  < /variables >
    < variable name = "request" messageType = "CreditInfoMessage"/ >
    < variable name = "approvalInfo" messageType = "approvalMessage"/ >
  < /variables >
  < partnerLinks >
    < partnerLink name = "customer" partnerLinkType = "LinkType" myRole = "approver"/ >
    < partnerLink name = "approver" partnerLinkType = "LinkType" partnerRole = "approver"/ >
  < /partnerLinks >
  < sequence >
    < receive name = "receive" partner = "customer" portType = "approvalPT"
              operation = "approve" variable = "request" >
    < /receive >

    < invoke name = "invokeapprover" partner = "approver" portType = "approvalPT"
          operation = "approve" inputVariable = "request" outputVariable = "approvalInfo" >
    < /invoke >
    < reply name = "reply" partner = "customer" portType = "loanApprovalPT"
              operation = "approve" variable = "approvalInfo" >
    < /reply >
  < /sequence >
< /process >
```

The basic steps for the agents in our system to coordinate using the above BPEL4WS model and LCC protocol are illustrated in Figure 5.5 and are explained below:

- An agent, $\mathcal{A}_1$, receives the BPEL4WS specification, $\mathcal{B}$ together with the LCC protocol, $\mathcal{P}$. It takes the role of $a(interpreter(\mathcal{B}, [], [], [], \_), \mathcal{A}_1)$. It then tries the clauses that are defined in $\mathcal{P}$ to find the type of the $\mathcal{B}$ by using the constraints $is\_sequence/is\_invoke/...$) to determine the next BPEL4WS operator. For our example, the dominant operator in $\mathcal{B}$ is a *sequence* activity. $\mathcal{A}_1$ changes its role to $a(sequence(\mathcal{B}, [], [], [], \_), \mathcal{A}_1)$.

- $\mathcal{A}_1$ processes $\mathcal{B}$ in the role of $a(sequence(\mathcal{B}, [], [], [], \_), \mathcal{A}_1)$ by using the constraint $process\_sequence(\mathcal{B}, \mathcal{B}_1, \mathcal{B}_2)$ and gets the first element, $\mathcal{B}_1$, of $\mathcal{B}$ and the left elements $\mathcal{B}_2$ and then changes its role to $a(interpreter(\mathcal{B}_1, [\mathcal{B}_2], [], \_), \mathcal{A}_1)$ to repeat the first step.
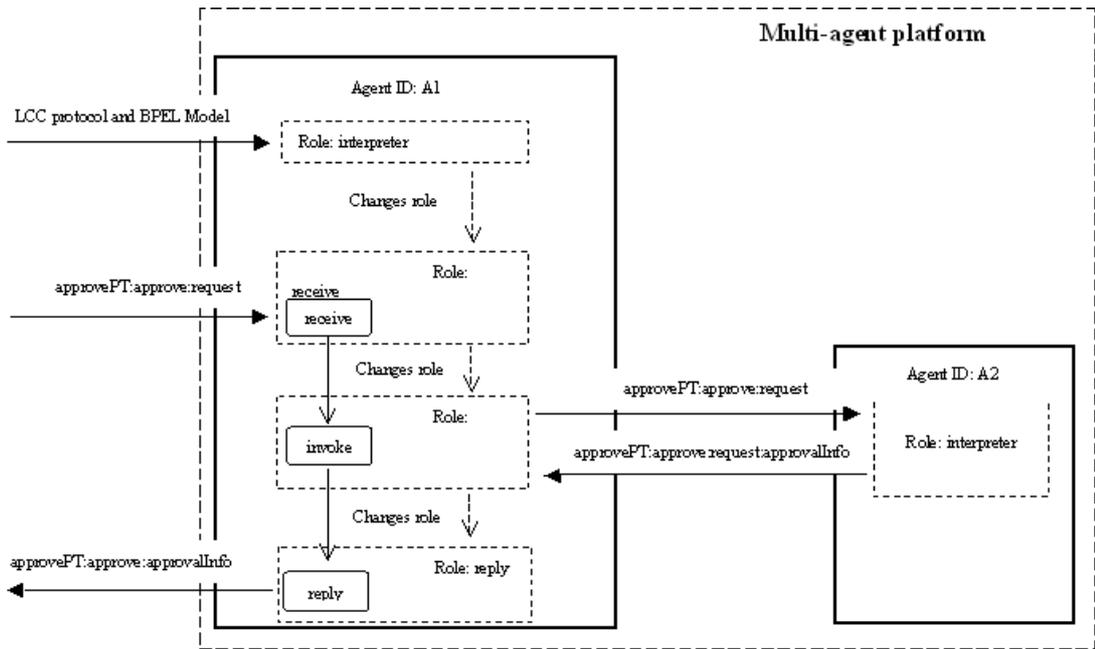
Figure 5.5: Agent's coordination for performing the illustrate example.

- By repeating the first step, $\mathcal{A}_1$ changes its role to $a(receive(\mathcal{B}_1,[\mathcal{B}_2],[],approver),\mathcal{A}_1)$ and waits for the message $PortType : Operation : request$. Once $\mathcal{A}_1$ receives the message, following the instructions in $\mathcal{P}$, it changes its role to

  $a(interpreter(\mathcal{B}_3,[\mathcal{B}_4],[request],[PortType : Operation : Customer : CustomerID],\_),\mathcal{A}_1)$ in which $\mathcal{B}_3$ is the first child element of $\mathcal{B}_2$ and $\mathcal{B}_4$ contains the remaining child elements of $\mathcal{B}_2$.

- By repeating the previous steps, $\mathcal{A}_1$ changes its role to $a(invoke(...),\mathcal{A}_1)$ and sends a appropriate message $\mathcal{M}$ to an agent $\mathcal{A}_2$ together with $\mathcal{P}_1$. $\mathcal{A}_2$ starts processing the $\mathcal{B}_4$ after it receives the $\mathcal{P}_1$ and $\mathcal{M}$. The coordination continues, until the processing of $\mathcal{B}$ is finished.

5.4 Agent Design

The agents that participate the interaction on the BPEL4WS model based MAS platform are proxy agents, which means the agents themselves don't need to make complex decision making processes but simply follow what the LCC protocol asks them to do and perform some of the computational functions. Therefore, the design issues of such agents are mainly about how to enable the agent conform to the protocol received and to perform proper actions. The contents of the package passed between agents have to be discussed before we get into the agent's design since the rationale of agent design

relies on this. Figure 5.6 shows the inside structure of the message package that is used between agents based on our approach. For simplicity, the diagram only shows the essential components of the message package. The components located at the com-
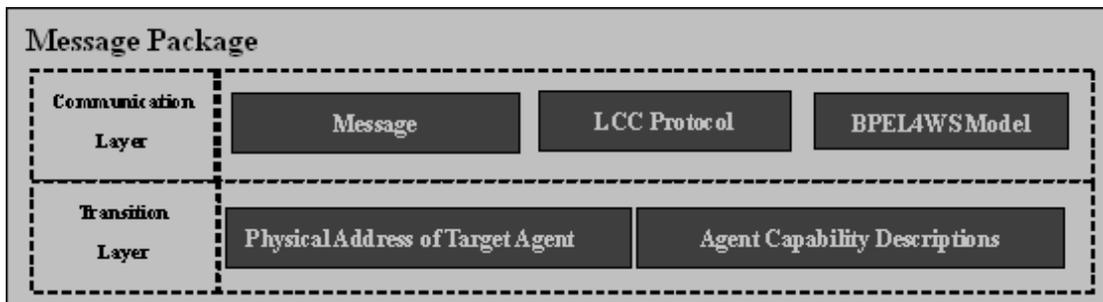
Figure 5.6: The essential components of our message package

munication layer have been discussed earlier. The transition layer contains two forms of agent verification information. "Physical agent address" defines the real location of the agents in the system, which might be a URL etc. "Agent capability description" describes the intend message receiver's capability. Thus when an agent receives a message package, it is able to decide if it can process this package before further expanding it.

According to the message package contents, the internal structure of the agents based on our approach is shown in Figure 5.7:
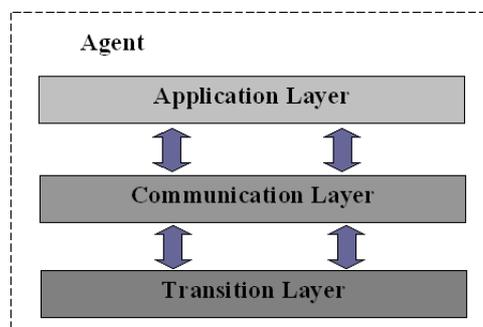
Figure 5.7: The internal structure of an agent

- **Transition layer:** is responsible for the underlying message passing between different agents. It controls the message passing at the lowest level of our system. It receives the processed outgoing messages from communication layer and forwards the received messages from other agents to communication layer. The basic components of transition layer is shown in Figure 5.8 The "incoming mes-
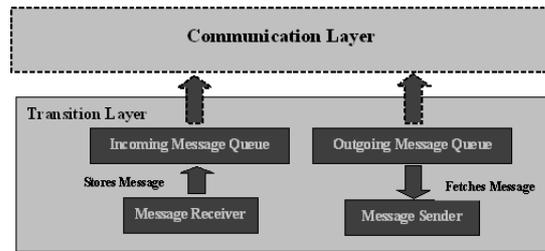
Figure 5.8: The components of agent's Transition layer

sage queue" and "outgoing message queue" are used to store the the message re-
ceived and the messages that are going to be sent out. These two message queues
are operated by "message receiver" and "message sender" in a first in first out
manner and are used as a channel for the communication between the transition
layer and communication layer. Once a "message receiver" receives 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 filtering
transition level information of the received package such as the if this message is
intended for it or if the agent it represents for matches the agent's capability de-
scription 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 the transition layer and producing the outgoing messages according to the
  protocol attached with the received messages. Figure 5.9 gives the inside look
  of the communication layer. "Incoming message processor" is used to judge
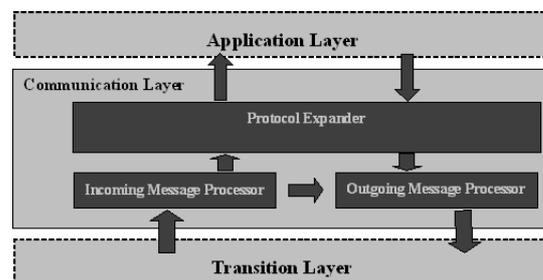
Figure 5.9: The components of agent's communication layer

whether the message that is fetched from "incoming message queue" is the one

that is required by the "protocol expander". If so, it passes it to "protocol expander". Otherwise, "incoming message processor" put this message and everything that is attached with it 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 a 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]:

$$A :: B \xrightarrow{M_i,M_o,\mathcal{P},O} A :: E \qquad if \quad B \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad (rule_1)$$

$$A_1 \, or \, A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad if \quad \neg closed(A_2) \wedge A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad (rule_2)$$

$$A_1 \, or \, A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad if \quad \neg closed(A_1) \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad (rule_3)$$

$$A_1 \, then \, A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E \, then \, A_2 \qquad if \quad A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad (rule_4)$$

$$A_1 \, then \, A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} A_1 \, then \, A_2 \qquad if \quad closed(A_1) \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E \qquad (rule_5)$$

$$A_1 \, par \, A_2 \xrightarrow{M_i,M_o,\mathcal{P},O_1 \cup O_2} E_1 \, par \, E_2 \qquad if \quad A_1 \xrightarrow{M_i,M_o,\mathcal{P},O_1} E_1 \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O_2} E_2 \quad (rule_6)$$

$$C \leftarrow A \Leftarrow M \xrightarrow{M_i,M_i-M \Leftarrow A,\mathcal{P},\phi} c(M \Leftarrow A) \qquad if \quad (M \Leftarrow A) \in M_i \wedge satisfy(C) \qquad (rule_7)$$

$$M \Rightarrow A \leftarrow C \xrightarrow{M_i,M_o,P,M \Rightarrow A} c(M \Rightarrow A) \qquad if \quad satisfied(C) \qquad (rule_8)$$

$$a(R,I) \leftarrow C \xrightarrow{M_i,M_o,P,\phi} a(R,I) :: B \qquad if \quad clause(\mathcal{P},a(R,I) :: B) \wedge satisfied(C) \quad (rule_9)$$

Rule$_1$ means the definition for a given agent may be re-written by re-writing the components of that definition. *Rule*$_2$ and *rule*$_3$ means that if any branch of a $''or''$ operator is properly expanded, processed and closed, the processing of $''or''$ operator is then accomplished. In order to expand a $''then''$ operator according to its sequential semantic, *rule*$_4$ and *rule*$_5$ together indicate that the clauses defined before a $''then''$ operator must be expanded before the expansion of the clauses defined after the $''then''$ operator. Parallel execution in LCC in controlled by operator "par" for which the re-write rule is defined by *rule*$_6$. *Rule*$_7$ and *rule*$_8$ are used to tell agent how to behave when it receives a message and sends out a message. When dealing with message passing, each agent has to process the constraints associated with the messages according to *rule*$_7$ (checks if the received message is the message that it waits for and then processes the constraints) and *rule*$_8$ (checks if the constraints are satisfied before it sends out the message and close the clause). *Rule*$_9$ defines the re-write procedure for agent role's

changing. According to it, if the constraints defined for agent role's chang-
ing is satisfied, agent then fetches the clauses defined for new role and
starts executing them according to the other re-write rules. A protocol term
is decided to be closed as follows:

$$closed(c(X))$$
$$closed(A \, or \, B) \leftarrow closed(A) \vee closed(B)$$
$$closed(A \, then \, B) \leftarrow closed(A) \wedge closed(B)$$
$$closed(A \, par \, B) \leftarrow closed(A) \wedge closed(B)$$
$$closed(X :: D) \leftarrow closed(B)$$

satisfied(*C*) is true if *C* can be solved from the agent's current state of
knowledge. *satisfy*(*C*) is true if the agent's state of knowledge can be
made such that *C* is satisfied. *clause*($\mathcal{P}$, *X*) is true if clause *X* is the dialogue
framework of protocol $\mathcal{P}$, as defined earlier.

  – If it holds a protocol and is waiting for a message, it asks "incoming mes-
    sage processor" for the message and blocks itself until it receives the re-
    quired message.

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

"Outgoing message processor" simply forwards the message package that it re-
ceives from "protocol expander" to "outgoing message queue" currently. It is a
place holder for outgoing message processing. For example, the message pack-
age 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. It contains at least two components, web services invoker and con-
  straints solver, as shown in Figure 5.10 "Web services invoker" takes care of all
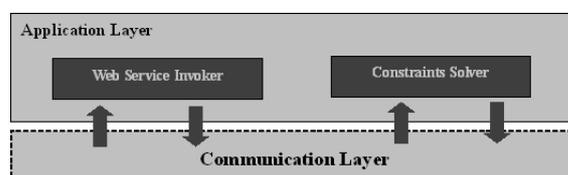
Figure 5.10: The components of agent's application layer

the issues of web services invocation including: invoking a web service accord-
ing 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 to the LCC protocol or purely solved by the local methods.

5.5   Prototype Implementation

5.5.1   JXTA P2P framework

The JXTA project (http://www.jxta.org) is a project proposed by Sun Microsystems, which is used to tackle current problems existing in the p2p world and provides a basic P2P platform. JXTA provides sets of open, generalised p2p protocols and services that help devices on the network to communicate and coordinate with each other. For the purpose of inter-operability, the project does not limit itself to any particular company, programming language, system or network infrastructure and tries to provide platform-independent solutions for p2p applications.

For developers, it provides a set of constructing components that support fundamental infrastructure for distributed applications. JXTA promises to support common functions that are required by all the p2p applications such as discovery, message routing i.e.. Therefore, users can concentrate on the high level application itself rather than low level system infrastructure. On the JXTA platform, a peer may be any networked device that implements one or more of the JXTA protocols. Peers decide to join peer groups on their own initiatives. A peer group is a collection of peers that have agreed on a common set of services and want to collaborate with each other to chase some common goals. To enable peers to advertise themselves and discover each other, to communicate and route messages to proper target, six JXTA protocols are supported by the current JXTA standard, which are:

- Peer Discovery Protocol (PDP) is the protocol that is used by peers to advertise their own resources and discover resources from other peers within a peer group.

- Peer Information Protocol (PIP) is the protocol provides a set of messages for peers to use to obtain the other status.

- Peer Resolver Protocol (PRP) is the protocol that enables peers to send a generic query to one or more peers and receive a response.

- Pipe Binding Protocol (PBP) is the protocol that is used by applications and services in order to communicate with other peers. It helps peers to build up a virtual communication channel with others for message exchanging.

- Endpoint Routing Protocol (ERP) is the protocol by which a peer can discover a route (sequence of hops) to send a message to another peer potentially traversing firewalls and NATs

- Rendezvous Protocol (RVP) is the protocol that is used for propagation of messages within a peer group. The Rendezvous Protocol provides mechanisms which enable propagation of messages to be performed in a controlled way

JXTA is a popular, open-source, royalty- and license-free p2p framework which has a large number of registered members of the development community. Actively supported by a growing community of p2p developers, JXTA technology has seen strong growth in its adoption, and some commercial applications are now emeraging. For these reasons, our prototype system is built up adopting the JXTA framework.

5.5.2 Overall prototype framework

The current prototype is produced in the Java using J2SE version 1.4 API. This prototype uses the JXTA grouping feature for virtual community management. Communication between agents relies on JXTA messaging protocols such as advertisement and pipe. The messages that are passed between peers are in XML format. This prototype consists of two main components which perform business workflow functions and p2p functions, respectively as illustrated in Figure 5.11.

The core services of JXTA include the group service, the peer service, the pipe service, the discovery service and the advertisement service. A summary of the core services is as follows:

- Group service: this service deploys the grouping concepts of p2p applications. Each group can have policies of membership. For example, all the participants involved in a BPEL4WS interaction model is a group and actually, in our prototype, each BPEL4WS process model is used to organise a JXTA group, which means a JXTA group is built up according to given BPEL4WS model. The group service provides peers with the capability to discover groups, fetch information about all the participants in a group, create a pipe to communicate with others in a group, join a group and leave a group, etc.
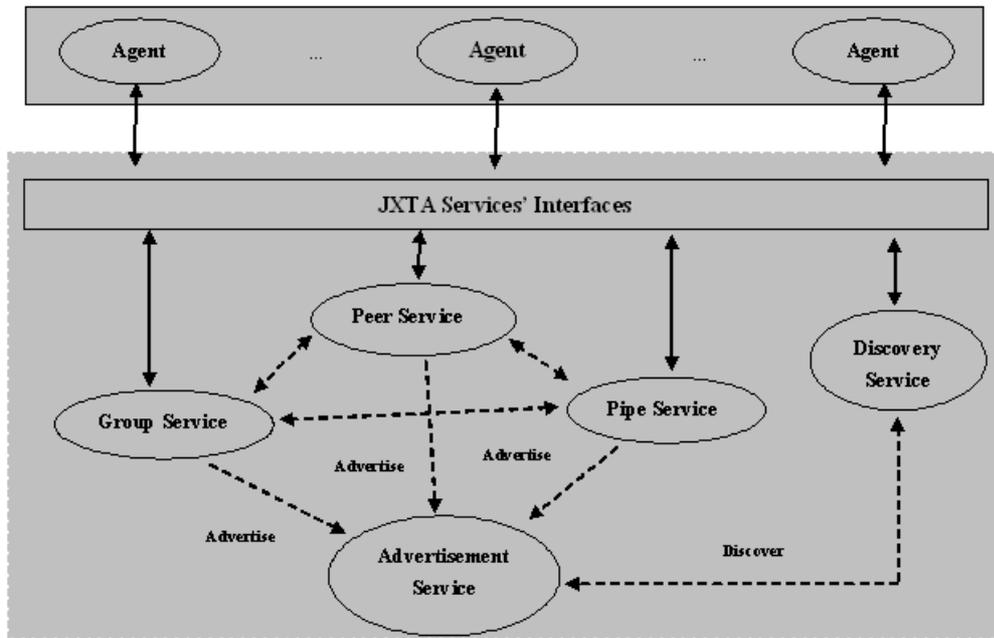
Figure 5.11: Overview framework of prototype

- Pipe service: this service has capabilities of managing the communication between peers. The service can help peers search for a pipe advertisement using the discovery service. This service consists of two sub-services:

  - the InputPipe service that is used to enable peers to send messages to others and

  - the OutputPipe service that is used to help peers to receive messages from others.

  Also, pipes offer two sorts of communication styles, point-to-point and broadcast, which can be used for different circumstances. For our prototype, each agent has a binding input pipe for receiving messages. The connection between an agent and its input pipe is through the connection of the agent's advertisement and the input pipe's advertisement.

- Advertisement service: this service is used to publish resources in the JXTA virtual network. There are several different advisement types defined in JXTA such as:

  - peer advertisement: describes the basic information of a peer/agent, including its physical JXTA ID, its application role, its capabilities and the

information of its incoming pipe that is used receive message from others.

- – peer group advertisement: describes the basic information of existing peer group, including its JXTA ID, its name and the information of its associated BPEL4WS model.

- – pipe advertisement: describes the basic information of the pipes created within a peer group for sending and receiving messages.

- Discovery service: this service is used to help peer search for advertisements in the so that resources associated with advertisements can be used.

5.5.3 Implementation of Key System Components

The implementation of the key components of the LCC based decentralised workflow management system are described in the following sections.

5.5.3.1 Implementation of agents group

Our prototype system uses the concept of an agent group to enable agents' interaction in an organised manner. Each agent group, as explained, is organised according to its associated process model. Any agent that is willing to participant in the interaction specified by a process model can join or quit its group. Once an agent joins a group, it must take one of the application roles defined in the process model. In order to carry a valid interaction, each group should be composed of at least one agent for each of the application roles that are required by the process model. More agents that act for an individual application role are allowed in our system, since any of them can be selected as interaction partner by others for a particular process instance during the interaction. The agent group in our prototype is realised by JXTA group service. When an agent joins a group, it needs to publish its peer advertisement and its input pipe advertisement so that other agents in the group can discover and communicate with it.

Figure 5.12, 5.13 shows the web interface for browsing,joining and quitting existing interaction groups.
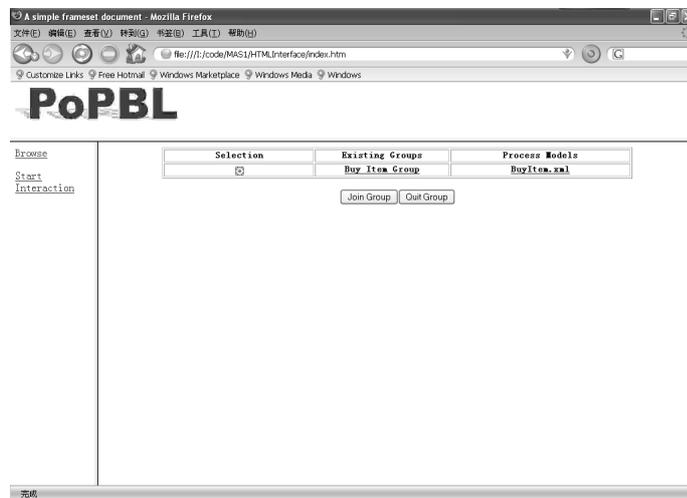
Figure 5.12: Interface for browsing, joining and quitting existing interaction groups

Figure 5.13: Interface for selecting application role

After joining in an agent group, an agent is allocated with an application role as shown in Figure 5.14
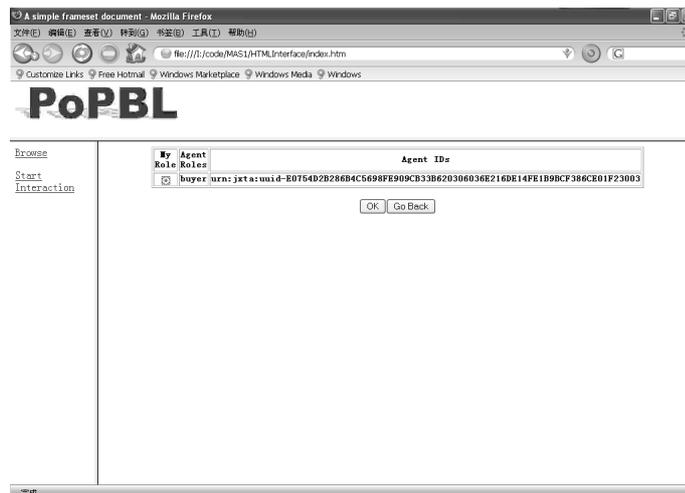
Figure 5.14: Interface for browsing existing agents in a Group

5.5.3.2 Implementation of agent kernel

At the heart of our prototype, the agent kernel of each of the distributed agents collaborates with others to achieve their common goal (automation of workflow process). In general, each agent works independently in the system, according to the workflow definition, and contributes to the operation of the whole workflow system. In the prototype, the interaction between two agents is realised through message exchange and the implementation of each layer inside an agent is discuss as follows:

- **Transition layer**: provides the message passing mechanism at the lowest as explained earlier. When an agent is first created, it has two fixed properties:

    - a JXTA Peer ID. Each agent/peer in JXTA can only have one unique physical ID. However, this unique ID can be mapped to multi IDs that are advertised for the JXTA peer in different peer advertisement in a JXTA group. Thus, it gives us the flexibility of using one physical agent to realise many agents that are defined in a LCC protocols since when the agents in our system communicate with each other, they locate each other solely using the IDs that are published in the peers' advertisement rather than their physical one.

    - a JXTA input pipe. Each agent/peer in our prototype has a unique JXTA input pipe which is used to receive messages from others. In JXTA, if a peer tries to send a message to another, it must know how to connect with its partner's input pipe using its output pipe. This is the common way in JXTA for agents/peers to communicate with each other. However, in LCC, agents don't care how the underlying message passing is done. All they need to know for the message passing is the recipient's ID. Therefore, when using JXTA to realise LCC protocol based agents' communication, the agents' ID must be associated with the concrete message transferring mechanism (pipe service). The simplest way to realise such association in JXTA is using its advertisement service. In our prototype, each agent's ID is published also in its input pipe advertisement. Thus, when an agent tries to send a message to its partner, it fetches its partner's ID in the message package and according to the ID, it uses JXTA discovery service to discover the proper input pipe that is associated with the ID and then it creates its own output pipe to connect with the input pipe discovered. After the connection is built, messages are then passed through the channel.

    Figure 5.15 shows the construction of message channel between two agent's transition layers on JXTA platform.

- **Communication layer**: handles all the LCC protocol related operations. The LCC protocol interpreter is based on the logic programming language Prolog. Although any LCC protocol can be mapped directly to Prolog syntax and thus can be easily processed by Prolog engine. Prolog is little used by industry at
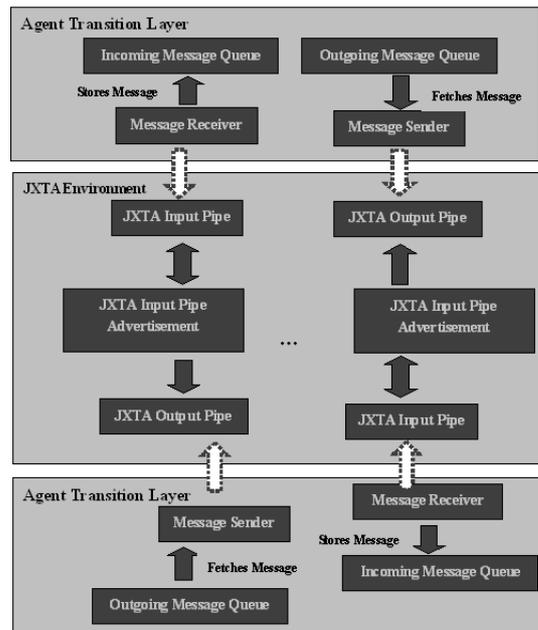
Figure 5.15: Implementation of the components at agent transition Layer

present. Therefore, we develop a JAVA based engine which only understands
the concepts that are from LCC. This engine is the core component, namely a
protocol expander in communication layer. It processes the LCC protocol in the
same way as the Prolog engine. It is able to process the predicates with multiple
arguments that are designed for LCC and returns multiple results accordingly.
Thus, the existing mechanisms and algorithms that are used for Prolog based
LCC processor, can be adopted in directly for the JAVA based version.

- **Application layer**: is responsible for handling the execution of computational
  functions. In our system prototype, the computational functions are implemented
  in three ways:

    – web services;

    – locally stored functions;

    – and functions that are passed from the others.

  Web services provide the business application functions that are required by the
  BPEL4WS process model. Locally stored functions; the functions passed from
  the others are used for LCC protocol constraint solving. The reason for us to
  design both local functions and communicated functions is because some of
  the functions, especially for those that are independent on any of the particu-

lar agents, can be re-used if they can be passed between agents and thus reduce the complexity of individual agents.

Figure 5.16 and Figure 5.17 show the interface for the variables' instantiation and for tracking the messages' passed between agents.
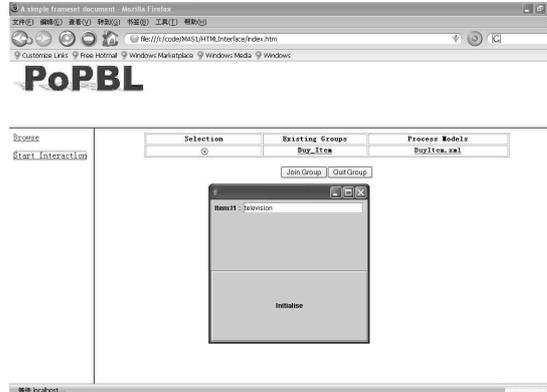
Figure 5.16: Interface for initialising variables
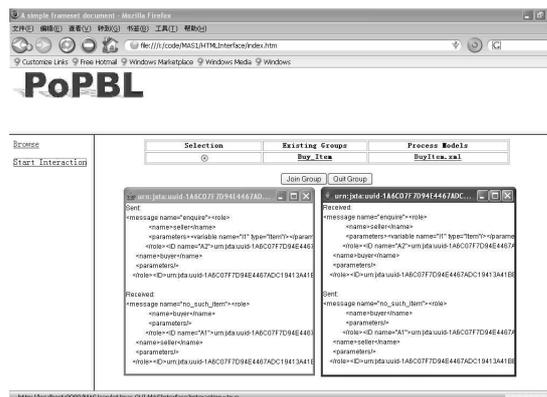
Figure 5.17: Interface for tracking agent's messages' passing

5.6 Discussion

Our approach provides an opportunity to build a multi-agent based distributed workflow system starting from a business process model rather than from a interaction protocol, which narrows the gap between the high level requirement and system specification in the development of multi-agent system and connects the business workflow community and multi-agent community. Thus, business users can produce their own

business process models that can be used directly in the multi-agent system. Furthermore, since there are many available techniques and tools for business process modeling, these can be adopted directly for building multi-agent systems based on our approach.

The LCC protocol used to interpret BPEL4WS models is independent of any specific message passing infrastructure, although we have described it with respect to a distributed and multi-agent based system infrastructure, it could equally well be deployed in a more traditional server based style. Different styles of deployment are described in detail in [Rob04]. Furthermore, the protocol can be used prior to deployment in order to predict behaviours and possible errors in interaction[Wal04b]. Another advantage is that the workflow engine built using our approach is a real generic server. The only specific knowledge it contains is how to process the LCC protocol and how to invoke the web services but not how to process the particular business process modelling language, which gives us a very efficient and lightweight way for system re-design and re-implementation.

5.7   Summary

In this chapter, we proposed a novel approach for using BPEL4WS specification to guide multi-agent interactions. In this approach the LCC protocol is used as a language interpreter to enable agents to understand the BPEL4WS syntaxes.

A system prototype based on the proposed approach is shown. It uses JXTA as the underlying infrastructure and uses our proposed coordination mechanism for agents' interaction. Web services are also conscripted as computing units to enable real applications to be accessed our system.

Chapter 6

Extending Our System For Incomplete Process Support

6.1   Causes of Incomplete Processes

Workflow management systems that support traditional application domains (office work, banking industry, etc.) are usually mature and fixed. Processes' goals, the activities that lead to each goal and the details of each activity are normally pre-defined. Hence, process modellers often can completely see and formally specify the boundary of a process in advance. As a result, traditional workflow management systems conform to the principle of defining first and executing thereafter.

The execution mechanism adopted conventionally is known as lazy execution where the workflow system starts the execution of the instances of a workflow process only after the process is modelled and specified completely. The build-time functions that act on process modelling, representation and storage issues and the runtime functions that perform the execution of process instances are conducted respectively. However, in some application domains such as scientific computing, health care, the process specification obtained before-hand only describes the workflow process in a rough and incomplete manner. Main activities, products, roles and the structure of the process model can normally be articulated. However further elaboration and eliciting of the process are required to be accomplished during the execution of process based on process instance data [SJT97, Sie99]. These sorts of processes are known as incomplete processes. A simple but typical incomplete process scenario is illustrated in Figure 6.1.

The process model in Figure 6.1 shows a incomplete process representing a typical
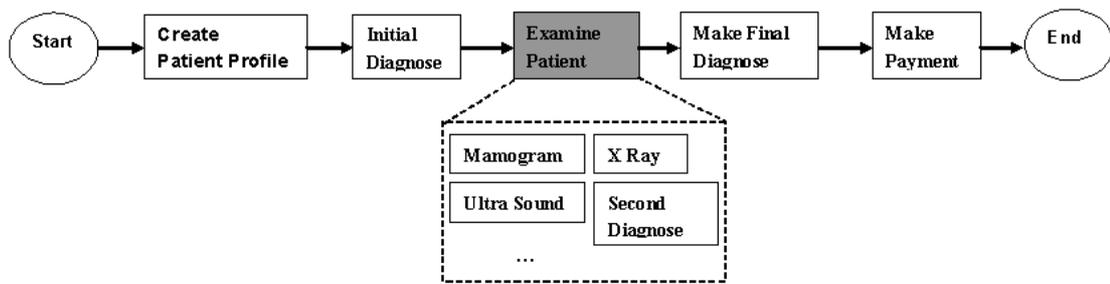
Figure 6.1: The healthy care process

diagnostic process for investigation of patients in hospital. A new patient coming into the center will first be created a profile according to his/her current status by using registration service. All patients then consult with an attending physician for the first diagnose, who will determine what tests need to be performed, on a case-by-case basis. This brings the flexibility for the process. After the tests, the patient is called again by the attending physician who explains the results of the tests and makes a diagnosis. The patient is then required to report to accounts to make the required payments before leaving the center. In this process, the activity that is depicted in the gray box is an incomplete activity. It can not be pre-defined until the physician sees the real situation of the patient. For such flexible workflows, conventional client-server workflow architecture is not perfectly suitable. Because the execution of the workflow relies highly on the individual decision making process of each participant.

Normally, incomplete processes exists because of the following:

1. Processes in some new domains are much more complicated than those in normal ones. The increased complexity makes a process very difficult to be completely designed before its execution. At design-time, only the main process structure and few activities can be determined, while others remain unsolvable and need to be accomplished during process execution. For example, in the domain of health care where the entire task is modelled as a complex process, some instance data based activities may not be finalised until that information is fetched. Normally, this part of the process is modelled generally as composite activities (with some of is sub-activities missing) during the process modelling period. These composite activities need to be further articulated to concrete activities at run time when process execution reaches a certain level, considering the result of some of the completed activities.

2. Also in some new domains, processes are more flexible than those in conven-

tional ones. Different instances may not follow pre-defined process rules precisely but have small deviations. In particular, the possibilities to complete a task in various cases may be vary different. To foresee and to model of all these possibilities is either impossible or at least not necessary. Therefore, it is hard to define a complete process model in advance which describes all the situations that the instances may fall in. A typical example of this case is still the health care process, where inpatient treatments are prescribed uniquely for each case[SSO01].

3. In some cases, it is hard to get essential information to model processes completely before-hand, especially in application domains like scientific research, invention and the laboratory environment. Modelling such exploratory work is a difficult task because that very limited information is available and can be used for reference[JWB96, SV96]. In particular, some modelling information is completely unknown until the process instances are executed to a certain stage. Therefore, the complete process definition of tasks at a later stage cannot be pre-defined because the outcomes of tasks at early stages are unclear. As a result, a complete workflow process cannot be obtained at the design time.

6.2 Problem Analysis

Incomplete process support is the capability of a workflow system to execute a process model before it is complete specified, where the full specification of the process can made at runtime, and may vary for different instances. Some requirements have been described for workflow management systems that support incomplete models as follows, which are not fulfilled in conventional workflow research[JYR04].

1. Incomplete parts of workflow processes have to be specified explicitly as incomplete at build-time.

2. A workflow management system uses a different execution mechanism, where execution of the process instances is able to be performed even if the process is not specified completely.

3. An automated run-time facility should be provided, which enables further articulation of the workflow processes at run time without affecting the current running instances.

In a decentralised workflow management environment, for instance, a MAS based workflow system, some extra requirements are needed:

4. An incomplete process definition can be divided into task partitions and task partitions can be distributed to relevant agents appropriately.

5. Real time incomplete process support can be carried out in a decentralised environment so that process elaboration can be performed at the right time and the right place by the right participant.

For the above sorts of requirements, a conventional centralised workflow architecture (client-server) cannot easily be adopted since all the client end participants can only be invoked passively and have no authority to revise the workflow model. Of course, to fulfill the first three requirements, the process modelling language that is used to describe the process models used in conventional workflow management system can be revised. However, the fourth and fifth requirements can not be satisfied easily anyway because the fulfillment of incomplete activities has to be carried in a distributed manner since the private knowledge of each participant is needed.

On the contrary, our approach proposed in chapter 5 could satisfy all the above requirements perfectly with minor extension. Based on our proposed architecture (a decentralised multi-agent platform), all the participants (agents) are equal and have their own initiatives. Each activity in the pre-defined workflow model is executed by a corresponding agent and the whole process model is passed between agents in a sequential order, which means that the agents can instantiate the incomplete activities defined in the process model appropriately since all the states of the running process instance are clear to the agent when it holds the process instance. In the following sections, we will discuss different sorts of incomplete activities in incomplete processes and how they can be instantiated by our agents.

6.3   Categories Of Incomplete Activities

As discussed earlier, the reason processes are incomplete is because some of the activities/infomation inside/of the processes are missing. Those missing activities or activities with some of their properties missing are known as incomplete activities. Incomplete activities in incomplete process can be of two sorts:

- A property missing activity: is an activity that has some of their properties undefined at build time and those missing properties have to be decided at run time.

A properties missing activity can be either a basic activity in process model or a composite activity as shown in the following example:

$$basic\_activity(\{role, role\}, activity\_name, \{unknown\}, \{postconditions\})$$

In the definition of the above activity, only the *role* and the *postconditions* are pre-defined and the information of all the other properties are missing (input, pre-conditions and outputs). These attributes need to be articulated before the execution of activity instances starts.

- A component missing activity or an incomplete composite activity: is a composite activity with some/whole of its sub-activities missing. For example, a sequence activity might be defined with some unknown activities as its elements shown as follows:

$$process(sequence(name, unknown\_activity, basic\_activity(...), unknown\_activity, ...)$$

For an incomplete composite activity, some/all of its sub-activities are missing at the design time and have to be fulfilled at run time. An incomplete composite activity is a "black-box" defined in the process model. Each composite activity represents a piece of work which is filled by executing a set of sub-activities. These sub-activities, each of which can be either atomic or composite, can also be partially specified, forming a sub-process. A composite activity has a predictable contribution to the whole process. In other words, the objective of a composite activity, its position within the process, and its input and output should be all pre-defined. But the details about how to fulfil composite activities, i.e., how to convert input parameters into output parameters remain uncertain beforehand. The full specification of the composite activity needs to be made in real time. The construction of a composite activity will not affect those activities which feed it with inputs or use its outputs. According to the ways in which an incomplete composite activity is instantiated, incomplete composite activities can be further classified into two categories:

  - An open composite activity: intergrates pre-determined and open activities within a single workflow [NH94]. In this case, a pre-determined workflow is used as the main process structure but some of the composite activities of it are unknown completely. What we mean by "unknown" here is most of its information is not available and needs to be built up at run-time.

At a particular step of the execution of the process, several participants will coordinate with each other for the completion of unknown composite activities.

– A controlled composite activity: "is characterised by integrating some types of activities into predefined workflows that are somewhat more pre-determined than completely open process elements"[NH94]. In particular, a controlled composite activity has a set of components, where each component may consist of either an atomic activity or is a composite activity. Normally, the fulfilment of a controlled composite task requires the execution of some of the components in a certain sequence. However, this sequence remains uncertain beforehand and needs to be determined in real time by the activities' performer.

Other sorts of incomplete activities that are more complicated and are more difficult to address can exist. For instance, an activity that may have both missing properties and unclear relationships with other activities. However normally, a process model that contains such incomplete activities are not considered as incomplete. It is almost impossible to start executing such incomplete processes with expectation to elicit them properly later on. Therefore, we believe that support for the above incomplete activities is enough for real life applications.

6.4 Incomplete Activity Instantiation

The instantiation of incomplete processes in general, can be performed in one of the following ways:

- Semi-automated or manual support: The instantiation task requires an agent to define new activities or adapt some information and then build the sub-activities from the new/existing activities.

- Fully automated support: The instantiation task automatically makes a complete specification of an incomplete composite task by composing the existing activities based on the instance data and given constraints.

Several questions have to be answered in order to instantiate incomplete activities at the run-time regardless of the underlying system architecture:

1. When will an incomplete activity be instantiated?

2. Where will the instantiation of an incomplete activity occur?

3. How will an incomplete activity be instantiated?

For different incomplete activities that have different information missing at build time, the answers for the above questions are different. For those incomplete activities that have some of its properties missing (inputs, performers, outputs, etc..), the information of the missing properties must have to be fulfilled before the processing of themselves. However, for those incomplete composite activities which have all their properties defined properly but say nothing about how to achieve the transition between the properties, they can be instantiated during the execution time.

To fulfill the incomplete activities that are with some of the properties missing, a special managerial activity, known as a instantiation activity, is designed. What the instantiation activities do in an incomplete workflow process model is that they are used to instantiate particular incomplete activities that are associated with it as depicted in Figure 6.2:
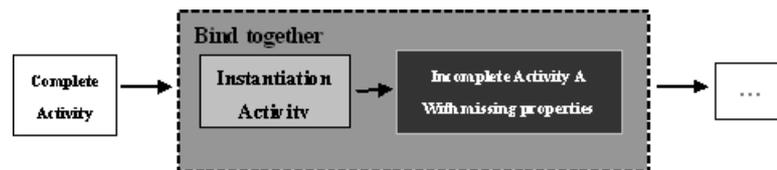
Figure 6.2: The bind of instantiation activity and its associated activity

An instantiation activity is only used to complete the missing properties of the incomplete activities but doesn't care about how the incomplete activity is made complete. The description of an instantiation activity is as follows:

- Responsibility: An instantiation activity is carried out by a certain participant who offers special services to model processes, such as a process engineer or a project manager.

- Inputs: The inputs of an instantiation activity are the existing information of the incomplete activities that it needs to instantiate as well as the information about the current states of the whole process. Given this information, the instantiation activity is able to decide how to complete missing properties of incomplete activities.

- Output: The single output of an instantiation activity is a complete specification of the properties of its associated incomplete activities. If output activities are

incomplete basic activities, they can then be executed directly while being invoked. If output activities are composite activities, their sub-activities have to be completed before execution.

By using instantiation activities, the instantiation questions of $''where \, and \, when''$ for properties missing activities can be answered. An incomplete basic activity is instantiated by the execution of an instantiation activity that is defined before it in the process model. Such instantiation takes place in the space of the instantiation activity's performer's space. For incomplete composite activities, instantiation is achieved in two phases. In phase one, the missing properties are instantiated by the instantiation activity with which they are associated and in phase two, the components of them and the orders between them are decided by all the participants involved.

Technically, there are many ways of indicating the notations of both incomplete activity and instantiation activity in a process model; the simplest way might be the revision of the process modelling languages. New syntax is adopted to distinguish complete and incomplete activities so that when the workflow participants are executing activities, they know what actions need to be performed accordingly. We use the following notation:

$$activity(Name, ID, \textbf{Instantiation Activity}, \textbf{Associated Incomplete Activity}, Outputs)$$
$$activity(Name, ID, \textbf{Incomplete Activity}, Inputs, Outputs)$$

In the following sub-sections, we will explain for different sorts of incomplete activities *how* they are instantiated.

6.4.1 Completing activity properties

Completion of missing properties of activities , as we have discussed earlier, is performed by instantiation activities. We assume that if an instantiation activity is reached during process execution, the missing information at build time for the completion of its associated activity is all available. Thus, the agent that executes the instantiation activity can use such information to complete the missing properties of the activity. An instantiation activity can be understood as a place holder to tell when the information for completing incomplete activities' properties is available.

How the agent that processes the instantiation activity completes the missing properties is domain specific and varies for different applications so is not the issue that we address in this thesis. The more general point that we address is that how missing properties articulating a process are undertaken based on our existing decentralised

platform. In general, the instantiation activity performing agent might use its own knowledge and the available information to instantiate incomplete activities or it can communicate with other agents to make the decision together.

As explained in the previous chapter, the whole process model is passed between agents and is processed in a linearised manner. Thus, after an instantiation activity is executed, the result of the execution (an activity with all its missing properties completed) is fully available to the following agents who are going to process the incomplete activity. Whenever an agent executes a property missing activity, it looks up in its received package for the completed replacement of this activity and executes it. Figure 6.3 shows the basic process
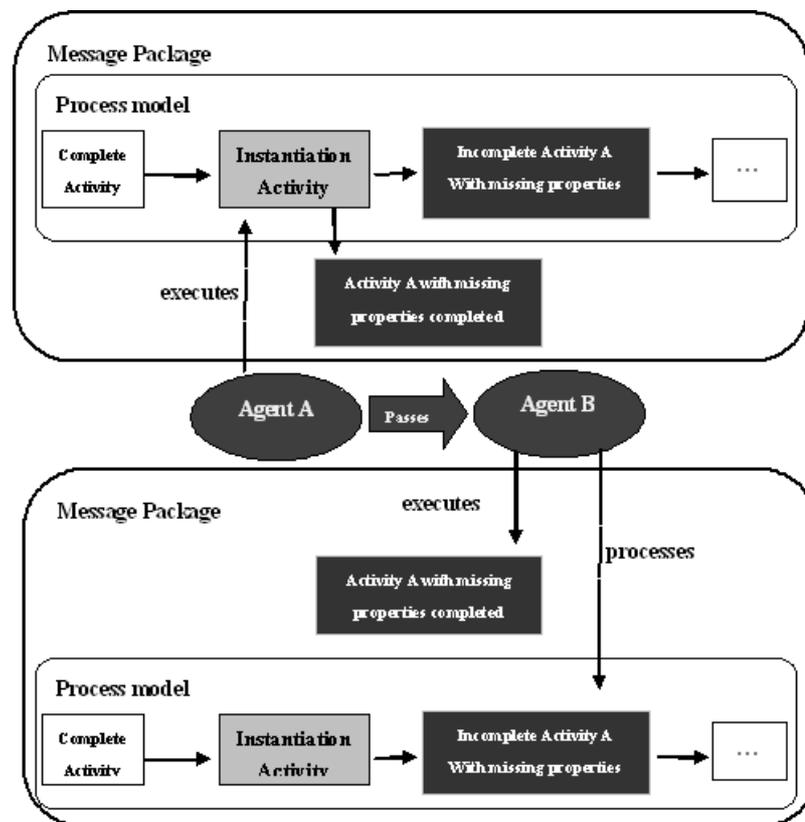
Figure 6.3: The healthy care process

6.4.2   Instantiation of Controlled Incomplete Composite Activities

A controlled incomplete activity, as explained earlier has already got all of its sub-components available at design phase. Normally, the performer of this sort of incomplete composite activity instantiation is pre-defined and all we need to do during

the instantiation process is to decide the proper sequence between those components. When a designated agent receives such an activity, it starts processing it using its own knowledge according to the existing evidence. A critical question however is, how we can ensure that the composition of the selected sub-activities for the incomplete composite activity is a valid one, where validity relates to the semantic correctness of the composition in relation to the process under consideration. Valid composition must be ensured through the build rules captured that are associated with the instantiating activities.

In order to tackle the problem addressed above, a framework is proposed as shown in Figure 6.4. In this framework, after the agent receives a message and an attached
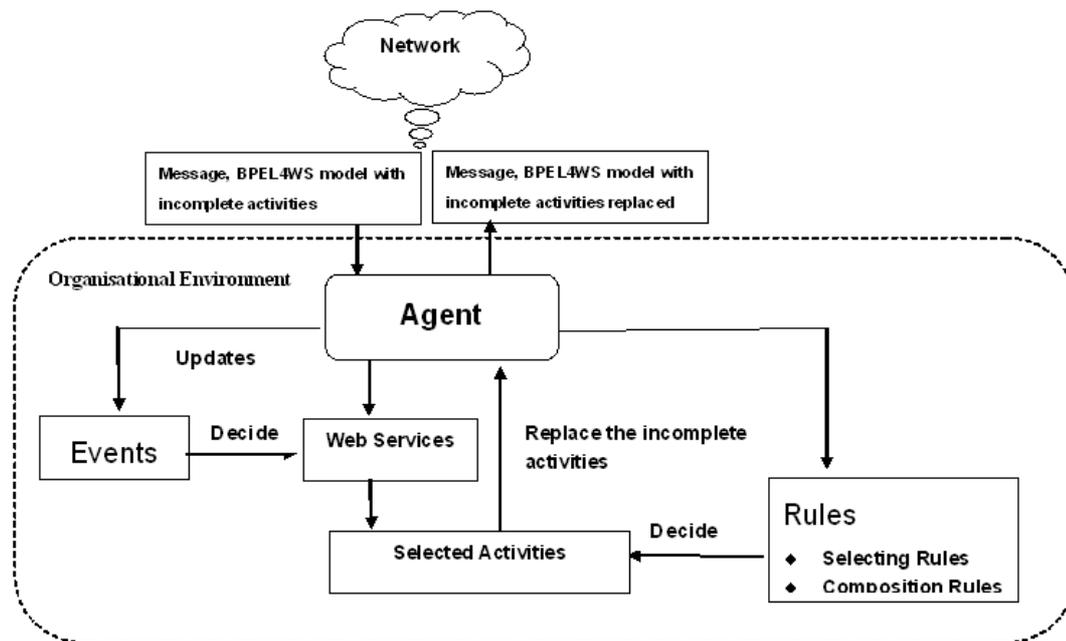
Figure 6.4: A framework for incomplete activity instantiation

instantiating activity,

1. It first requires the end users to select activities from the available basic activities (*web services*) according to the run-time instance information.

2. Then it checks whether the *selected activities* conform to the *selecting rules* defined associated with the instantiating activity.

3. If the selection is valid, the activities are composed into a sub-process manually by end users or automatically by algorithm.

4. After the agent checks whether the composition complies with the *composition rules*, it sends the instantiated activities to next agent.

The basic elements that are used in the framework are:

- **Web Services($\mathcal{W}$)**: is a set of web services that are available to the agent inside an organisation, which is expressed in the form of $\mathcal{W} = \{w_1, w_2, ..., w_n\}$.

- **Selected Activities($\mathcal{A}$)**: is a set of activities that are made up by the agent using the existing $\mathcal{W}$, which is expressed in the form of $\mathcal{A} = \{a_1, a_2, ..., a_n\}$. The activities in $\mathcal{A}$ can be a basic activity, composite activity or even incomplete activity.

- **Events(E)**: is the information about the data that is created by the instantiating activity (inputs/outputs). It may come from the messages captured and interpreted by the agent. For example, two possible events of the invocation on a BPEL4WS invoke activity can be interpreted by an agent as the facts below:

$$event(activity\_type, PortType : Operation : InputVariable)$$
$$event \left(\begin{array}{l} activity\_type, PotyType : Operation : \\ InputVariable : OutputVariable \end{array} \right)$$

Each event enacts at least an action that determines/helps the users to determine the sub-activities of the incomplete activity:

$$event(A, E) \Rightarrow \mathcal{A} = \{selected(a_1), selected(a_2), ..., selected(a_n)\}$$

- **Selecting rules:** defines the basic principles for agents to pick appropriate activities for fulfilling the incomplete activities. The selecting rules are constructed from the following basic operations (OP):

  - $selected(a)$: activity a is selected for the instantiation of the incomplete activity.

  - $\neg selected(a)$: activity a cannot be selected for the instantiation of the incomplete activity.

  - $selected(a) \lor selected(b)$: only one of the activities a and b can be selected for the incomplete activity's instantiation.

  - $selected(a) \land selected(b)$: both of the activities a and be can be selected for the incomplete activity's instantiation.

A Selecting rule is defined as:

$$OP[\wedge OP[, \vee OP[, ...]]] \Rightarrow OP[\wedge OP[, \wedge OP[, ...]]]$$

For example, the following selecting rules

$$selected(a) \vee selected(e) \Rightarrow (\neg selected(b)) \wedge (selected(c) \wedge selected(d))$$

means if activity a or e is selected, activity b cannot be selected, and there should only one activity between c and d be selected.

- **Composition rules:** indicates how the selected activities are composed for the incomplete activities (sequences among selected activities).

    - $or(a,b)$: If both activity a and b are selected, a can be executed before or after b no matter whether they are adjacent or not.

    - $before(a,b)$: If both activity a and b are selected, a must be executed before b no matter whether they are adjacent or not.

    - $sequence(a,b)$: If both activity a and b are selected, a must be executed before b and the two activities must be adjacent.

We can't explicitly define parallel structure in the composition rule because the architecture of our system. Both of the selecting rules and the composition rules can be defined manually or generated automatically based on the information of activities, for example, data dependence between two activities. The automatic composition of a given set of selected activities $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ using a set of composition rules $\mathcal{CR} = \{cr_1, cr_2, ..., cr_n\}$ that are relative to $\mathcal{A}$ is possible.

6.4.3 Instantiation of Open Incomplete Composite Activities

As termed earlier, an open incomplete composite activity is the activity that involves several different participants as its instantiators. In a multi-agent based environment, like ours, the process for instantiating open incomplete activities can be understood as a negotiation process among different participants. A negotiation process is viewed as a distributed search through potential compromises where each agent brings into the negotiation specific constraints on what it considers an acceptable resolution. For our system, the input to the negotiation process is an incomplete composite activity (with some of its properties or sub-activities missing) and the final output is a completed composite activity (with all its properties and sub-activities fulfilled). In order to enable

such a negotiation process, the definition of the open incomplete composite activity at least needs to have the following three properties:

- Inputs/pre-conditions that are used for staring the activity.

- Outputs/post-conditions that this activity should produce.

- Agents that should be involved in the instantiation process for the incomplete activity. Also it has to be ensured that these agents have the ability to contribute to the activities' instantiations.

After the open incomplete activities are equipped with the above properties, the instantiation problem then becomes a distributed planning problem. The final goal of the system is to produce a complete plan for executing the required task. Much research has been done for distributed planning problem but most of them reply on a centralised planner, which is what we try to take off for an open system. Therefore, a decentralised distributed planning mechanism has to be used. To build such a cooperative distributed planning system in an open manner, some of the key questions we must address include:

- How is the overall planning problem decomposed and allocated to the agents?

- How are the sub-plans of individual agents concatenated to produce the overall plan that can be executed coherently and effectively?

- How do agents communicate with one another during planning?

With the approach proposed in chapter 5, all the above questions can be answered. For the problem/goal decomposition and allocation question, the simplest solution might be using goal transformation, where a given goal is transformed into the another that is similar to the first or that is a sub-goal of the given goal. For example, if an agent in our system cannot achieve a goal by itself, it can transfer the goal into one that is achievable through coordination with others. In the other words, to solve a goal G, solve instead a goal G' that generates a sub-solution, and then pass the remainder of the goal (i.e., G minus G') to another agents. To achieve a particular goal state G, an agent must have an appropriate operator (internal functions or valid web services in our case). Thus, all the goals/sub-goals that are passed between agents are split into two sets. Set one contains the states for which the agent has an operator, and set two contains those states for which it does not. The agent solves all the states in set

one, and then it requests other agents to solve each state of set two. In the following sub-sections, we will explain in detail how our distributed planning mechanism works.

6.4.3.1   Round Table Coordination For Distributed Planning

Basically, for almost all the planning system, there are three necessary elements, which are:

- Initial State: describes what we have to start a plan. For our work (instantiating incomplete activities), the initial state of the planning system is the inputs of the activities.

- Goal State: describes what the final plan needs to achieve. Similarly with the initial state for our system, the outputs of the incomplete activities are the goal state.

- Operators: is a set of activations that make up the plan that leads us from initial state to goal state.

However, despite of the above common features of planning system, there is a clear difference between conventional planning systems and distributed planning systems, which is that the operators in distributed planning systems although are assumed to be there but only are available to those agents who own it. For our work, such operators are the complete activities that each agent knows and are represented as follows for later planning purposes:

$$Op(Action : activity\_name, Precondition : activities\_inputs, Effect : activities\_outputs)$$

The selection of proper operators (activities that each agent owns) for particular states are not possible since the operators are distributed and located in different agents. Although a centralised planner works as discussed in others' work [Geo88][NRdW05] for solving distributed planning problem, this violates our initial idea of building an open system. Therefore, for our system, to undertake the planning task without adopting a coordinator, it must select appropriate operators among all the participating agents. As we know, the planning process can be viewed as a search process for finding out the paths that connects the initial state and goal state in a tree structure. States are the nodes of the tree and the operators are the links that connect the states. A complete plan can be achieved as long as we can build up the complete tree structure using all

the operators. According to this, we propose a mechanism called "round table coordination". The general idea of this approach is that a plan package that contains all the un-solved states for a plan is passed between all the agents in a cyclical manner. During the process, all the agents look up each of the un-solved states and try to contribute their operators to make them evolve to new states. The basic representation of a plan package is given below:

$$
Plan \begin{cases}
Current\,State : \{state(AgentID, S_1), ..., state(AgentID, S_n)\} \\
Final\,State : \{S_n\} \\
Operators : \{O_1 : Op(...), O_2 : Op(...), ..., O_n : Op(...)\} \\
Links : \{S_1 \xrightarrow{O_1} S_2, ...\}
\end{cases}
$$

In the above representation, a plan package is composed of four child elements:

- Current state: defines a set of states that currently need to be solved. Each state is of the form: $state(AgentID, S_1)$ in which *agentID* indicates that which agent generates this state and can not make further evolvement on it. This concept is used to record the evolving process of the un-solved states. For our work, the un-solved states here indicate those inputs/outputs of certain activities that have no matches from other activities' outputs/inputs.

- Final state: defines the last state that indicates the completion of the plan.

- Operators: define a group of operators that can be used for the solution of the un-solved states.

- Links: defines a set of causal links. A causal link is written as $S_i \xrightarrow{O_n} S_j$. Causal links serve to record the purposes of operators in the plan: here a purpose of O_n is to achieve the states changing from S_i to S_j.

By passing around the plan package, a complete plan can be generated by the agents if it is obtainable. Whenever an agent receives the plan package from the others, it first checks the current state list and deletes all the states that are marked by itself. Then it checks all the remaining states that are generated by other agents and tries to use its own operators to make them evolve. All the new states after the evolvement should be marked by itself and added into the current state list. Also the operators and links' list are updated. One the agent has nothing more to do, it passes the updated plan package to the next agent for further processing. This process continues until the goal state is solved. An example distributed planning process is illustrated in the diagram below:
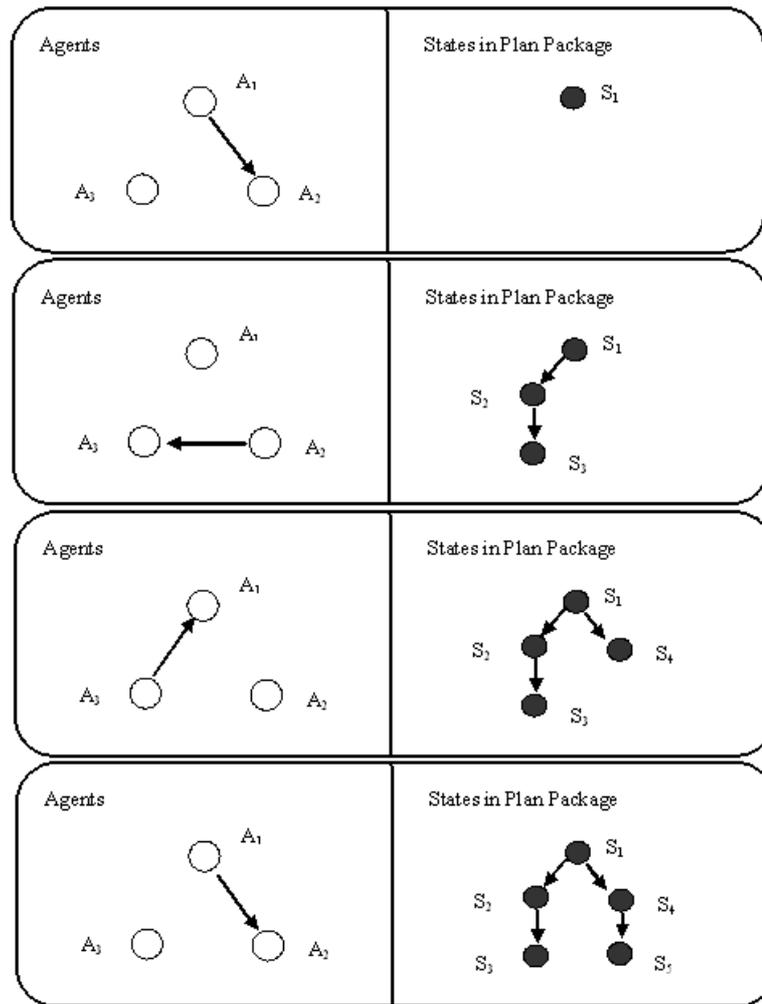
Figure 6.5: A framework for incomplete activity instantiation

In the example shown in Figure 6.5, there are threes agents A_1, A_2 and A_3 that are involved in the planning process. The initiate state of the whole planning process is the existence of a variable I_1 and the goal state of it is the existence of variables $O_m \land O_n$. The "round table coordination" process starts from A_1. Before A_1 contributes anything to the plan, the plan package's content is as follows:

$$Plan \begin{cases} Current\,State : \{state(Start, I_1)\}, \\ Final\,State : \{O_m, O_n\}, \\ Operators : \{\}, \\ Links : \{\} \end{cases}$$

and once A_1 receives the package, it tries to evolve the states in current state list using its own operator,

$$Op : \{Action : action_1, Precondition : I_1, Effect : O_1\}$$

and update the current state list, operators list and links list in the plan package. The content of the new plan package after A_1's processing is:

$$Plan \begin{cases} Current\ State : \{state(A_1,O_1)\}, \\ Final\ State : \{O_m,O_n\}, \\ Operators : \{action_1\}, \\ Links : \{I_1 \xrightarrow{action_1} O_1\} \end{cases}$$

A_1 then passes the plan package to A_2 to solve the remaining states. A_2 then updates the plan package using its operators and after its processing, the plan package becomes:

$$Plan \begin{cases} Current\ State : \{state(A_1,O_1),state(A_2,O_2),state(A_2,O_3)\}, \\ Final\ State : \{O_m,O_n\}, \\ Operators : \{action_1,action_2,action_3\}, \\ Links : \{I_1 \xrightarrow{action_1} O_1,O_1 \xrightarrow{action_2} O_2,O_2 \xrightarrow{action_3} O_3\} \end{cases}$$

Once A_3 grabs the above plan package, it finds that it can only contribute its operators to $state(A_1,O_1)$ to make it evolve and can not nothing to for those two states that are generated by A_2. The content of plan package is thus updated to: after its processing, the plan package becomes:

$$Plan \begin{cases} Current\ State : \{state(A_1,O_1),state(A_2,O_2),state(A_2,O_3),state(A_3,O_4)\}, \\ Final\ State : \{O_m,O_n\}, \\ Operators : \{action_1,action_2,action_3,action_4\}, \\ Links : \{I_1 \xrightarrow{action_1} O_1,O_1 \xrightarrow{action_2} O_2,O_2 \xrightarrow{action_3} O_3,O_1 \xrightarrow{action_4} O_4\} \end{cases}$$

The round that is lead by A_1 of the coordination terminates after A_1 receives the plan package again. A_1 deletes all the states that are marked by itself in the current state list to make sure that these states will not be making evolved again since all the other agents in the planning process have processed them already. It then starts adding operators to make those states that are generated by others evolve. In this way, we can see that a search tree for a complete plan is grown during its passage between those agents and we can ensure that the final plan that we get after the coordination is a complete plan since with the "round table coordination" mechanism, all the possible states during the planning process are checked and are evolved if possible by all the planning participants. The simple LCC chunk given below is used to ensure that the plan package is passed between agents in a cyclical manner.

$$a(planner(Plan\_package,[Head,role_1|Rest],role),ID) ::$$
$$solve\_it(Plan\_package_1,roleList_1) \Rightarrow a(planner(\_,roleList_1,role_1),ID_1)$$
$$\leftarrow update(Plan\_package,Plan\_package_1)\ and\ roleList_1 = [role_1,Rest|Head]$$

How the agents choose to contribute their actions/opeartors to make a complete plan relies completely on their internal design and in this thesis, we are only interested

in the architectural and communication issues. We don't discuss the agents internal intelligent decision making issues here.

6.5 Summary

In this chapter, our decentralised multi-agent platform has been extended to support incomplete processes. The causes of incomplete processes have been identified and conventional workflow system's inability to support incomplete processes has been analysed.

By introducing the instantiation activities, run-time instantiation of properties missing activities is modelled as an essential step in the process and integrated into the decentralised architecture. The missing components can then be filled up using agent's internal intelligence or the cooperation of a group of agents. From a system coordination viewpoint, the instantiation tasks are distributed, instantiated and scheduled to be executed as an ordinary task. Thus, process modelling at run-time can be performed with the support of the mechanisms for completing processes, at either instance or process level.

Chapter 7

Experimental Evaluations

Based on the system design and the corresponding mechanisms discussed in Chapters 4, 5 and 6, we use several real-world workflow applications in this chapter to illustrate how our approaches and system support workflow processes in a decentralised manner for evaluation purposes. The first case, discussed in Section 7.1, is a process for handling university student registration service, which can be considered as a conventional, complete workflow process and is used to test our interpretation based approach. The second case describes a shipping service process, which contains almost all the important BPEL4WS syntax and thus can be used to test our language mapping based approach. The third case discusses how our approach supports a heath care process that is first given in Chapter 6 , which is normally viewed as a non-traditional, incomplete workflow process.

7.1   Case Study 1: Student Registration Process

The student registration service processes the registration of students, which may include the activities of courses' registration, changing of schedule, quitting a course, paying tuition fees, and so on. This process is normally well defined and can be seen as a fixed activity process. Thus, workflow solutions are well suitable for this scenario. A typical scenario of the student registration process would be of the following:

- A student submits a completed registration form to the student's course advisor for approval. The course advisor views the information in the registration form and starts approving it. If the request is approved, the registration information will be sent to an enrolment officer for recording and if the request is rejected, the registration form will be sent to an enrolment officer to close this request.

- The enrolment officer updates the student's course information if the request is approved and sends a payment form to the financial section for billing. The registration information is also sent to university technical staff to setup up a computer account for the student.

- The officer in the financial section and the technical staff deal with the payment and computer account and inform an enrolment officer who doesn't have to be the same person that sent them the student's registration form.

- The enrolment officer collects notifications from both the financial section and the technical staff to complete the present registration service request.

- Finally, an enrolment officer advises the student the outcome of the request and closes this request.

Three characteristics of the student registration service which need to be addressed properly are illustrated in this thesis:

- First, as we can see that the student registration service is physically distributed. To carry out the whole registration process, staff from different departments are involved. For example, course advisors approve the registration requests of courses, technical staff sets up the computer accounts of students, financial section handles payment, and enrolment officers carry out all paper and some of the coordination work. These distributed staff, in terms of physical location and administration, should be able to collaborate with each other efficiently to provide the registration service to the student without the need of a centralised coordination mechanism.

- Second, due to the large number of the students, the student registration service would experience a heavy load. Students, may send their registration requests at anytime during the time period (just before the deadline for example). Thus, performance has to be considered as a main issue when the system is designed. The system is expected to handle a large amount of requests in a relatively short period of time. A pure decentralised coordination mechanism is clearly helpful for this purpose.

- Finally, although this scenario can be modelled as a workflow process and represented as a process model easily as shown in Figure 7.1 [1] (the formal BPEL4WS

---

[1] All the diagrams depicted in this chapter use FBPML[CBR98] notations

process model specified for this example can be found in appendix D.1), to represent it using a multi-agent interaction protocol is hard or almost impossible for non-technical users.
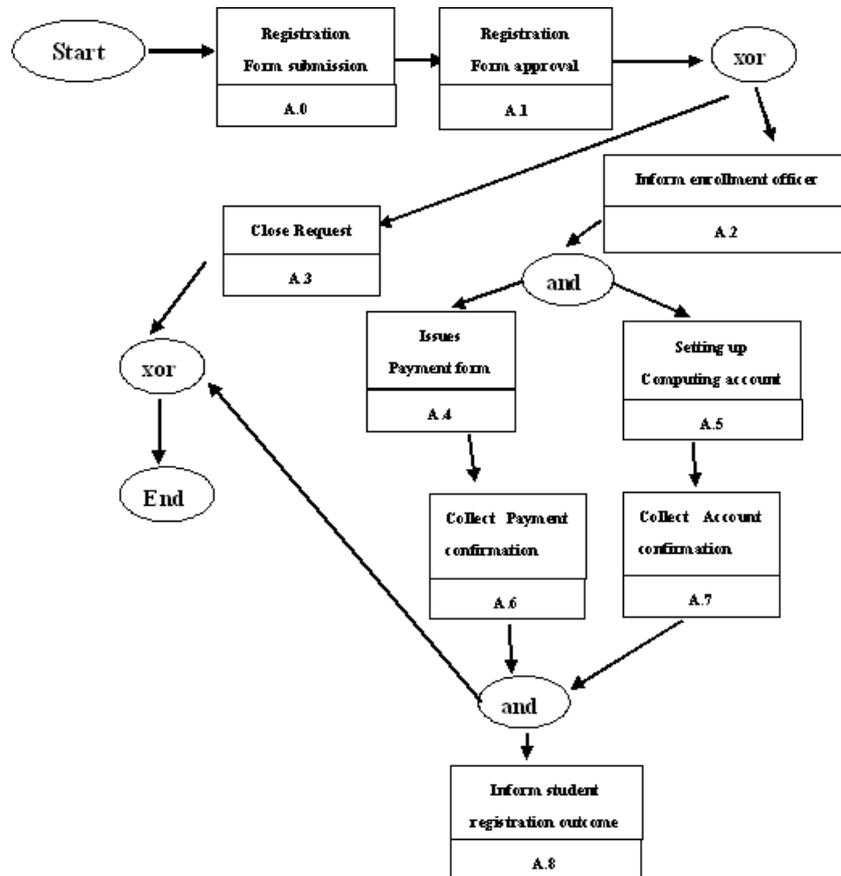
Figure 7.1: Student registration process

The above characteristics of student registration service, make the approaches described in this thesis attractive. This process consists of a set of tasks which need to be executed in a certain order. Also, this process involves participants such as enrolment officers, course advisors, technical staff and treasurers. The virtual organisational structure based on a multi-agent point of view is given in Figure 7.2.
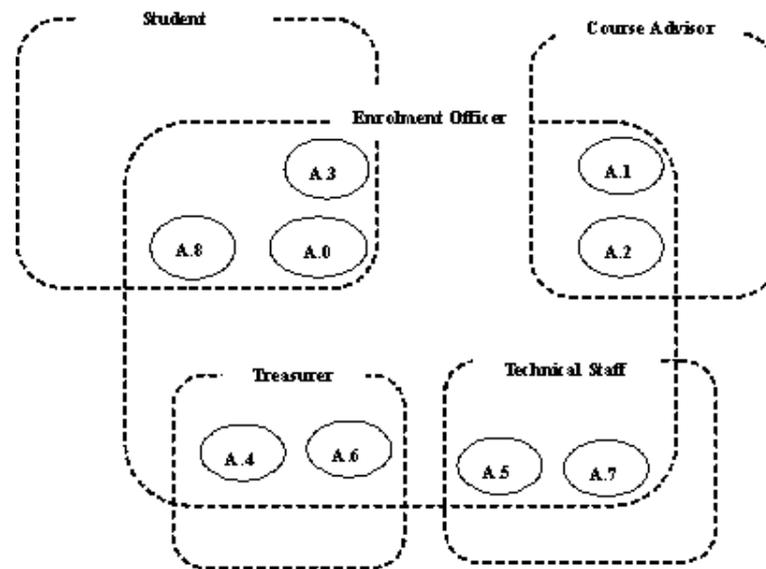
Figure 7.2: Virtual organisational structure of student registration process

Once a new registration request is received, a process instance following the process model depicted in Figure 7.3 is created to handle this request. Various agents collaborate with one another to create a process instance, using the mechanisms described in Chapter 5. Each agent in this virtual organisation has no overall knowledge of how the coordination process is organised and only performs the tasks when requested.

Five agents, namely *student, enrolment officer, course advisor, treasurer and technical staff*, are created to deploy the process for evaluation purpose.

7.1.1 Experimental evaluation of interpretation based approach

When this example process is deployed on our system using interpretation based approach proposed in Chapter 5, it is first re-written into a substitute (as shown in Figure 7.3 that has no concurrent computation structure defined (see Chapter 5 for detail). With support of the system developed in chapter 5, this re-write process is generated automatically (formal representation is listed in appendix D.2):

With the substitute process, when

- triggered by an approvable student registration form, the scenario is enacted by our decentralised and LCC interpretation based system as follows:

  – After *enrolment officer* receives a registration form from *student* (executing activity $A.0$), it first: performs the activity defined in the process model
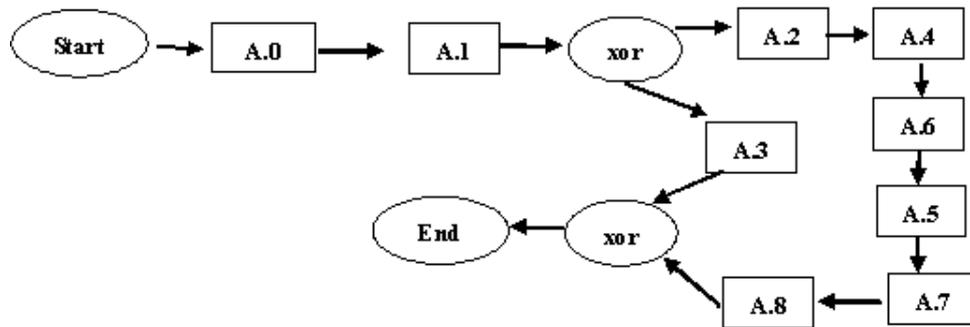
Figure 7.3: Substitute of original student registration process deployed on our system

internally; re-forms the registration form; and then passes the revised registration form and un-processed model to *course advisor* for further processing (executing activity *A*.1).

– *Course advisor* keeps processing the received document and process model. After it finishes its processing, it returns the result and process model to *enrolment officer*.

– Since the given data at the beginning is an approvable registration form, *enrolment officer* will execute task sequences for *Account − management* and *Payment handling* (*A*.2,*A*.4,*A*.6,*A*.5,*A*.7,*A*.8). The un-processed activities defined in the process model are passed between *enrolment officer*,*treasurer* and *technical staff* accordingly in a sequential order and at the last of the coordination process, a message (*registrationSuceed*) is sent back to the student.

• it is triggered by an un-approvable student registration form, the scenario is enacted by our decentralised and LCC interpretation based system as follows:

– After *enrolment officer* receives a registration form from *student* (executing activity *A*.0), it first: performs the activity defined in the process model internally; re-forms the registration form; and then passes the revised registration form and un-processed model to *course advisor* for further processing (executing activity *A*.1).

– *Course advisor* keeps processing the received document and process model. After it finishes its processing, it returns the result and process model to *enrolment officer*.

– Since the given data at the beginning is an un-approvable registration form,

enrolment officer will execute task *A*.3 and a message (*registrationFailed*) is then sent back to the student.

From this experiment, we can see that based on the two different sort of inputs (approvable student registration form and un-approvable stduent registration form), our system the intended task (defined by the original process model) well. Comparing with the execution performed by conventional workflow system, the only difference is that our system has to execute the parallel structure in the process model in a fixed manner, which is not as flexible as a conventional workflow system although the results of execution are the same.

In this case study, an unavailable agent (staff) exception can be detected and handled automatically. For example, if the delegated *officer of financial section* becomes unavailable before executing the payment function for billing, this task instance can be re-allocated to another *financial officer* quickly if there is one available, using the mechanism discussed in chapter 5.

Some benefits of our interpretation based approach and decentralised system are reflected using this case study.

- First of all, direct interaction between different agents would decrease communication delay, reduce the traffic of network and thus may achieve good performance. With conventional workflow system, the messages between all the participants have to be forwarded to each other through the centralised workflow server. As addressed previously, when the number of registration students increased during busy period, the server will be overloaded and thus the performance of the whole system is affected.

- Secondly, system robustness is likely to be enhanced because failure of any agent would not cause the failure of the whole system. For example, when an agent that represents an enrolment officer is broken, the work assigned to this agent can be quickly reassigned to another enrolment officer for execution using certain fault discovery mechanisms. With conventional workflow, once the workflow server is down, the whole system is dead and maybe not recoverable.

- Thirdly, the system is much more open as new staff members can join the system more easier to offer better processing capacity. For a centralised workflow system, this feature is not quite easy to achieve because once the workflow server

is designed, the capabilities of the system is fixed and the dynamic extension of the system's capacity during run time is hard.

- Finally, our system may satisfy staff members better. For example, enrolment officers can be involved in different process instances. They are not required to stick to any particular process and what they are requested to do completely relies on the messages they receive from others and their own initiative. In a conventional workflow system, all the staff members are only allowed to be allocated tasks and invoked for providing their services passively. In addition, they have very limited capability to take part in the management of the whole process during run time once the execution of some process instances have started.

7.2   Case Study 2: Shipping Service Process

7.2.1   Experimental evaluation of language mapping based approach

This case study uses a rudimentary shipping service described by BPEL4WS (formal model is given in appendix D.3) as a test bench to prove the soundness of our language mapping based approach. This service handles the shipment of orders offers two types of shipment as shown in Figure 7.4: shipments where the items are held and shipped together and shipment where the items are shipped piecemeal until all of the order is accounted for. Two participants (*shipping service customer* and *shipping service*
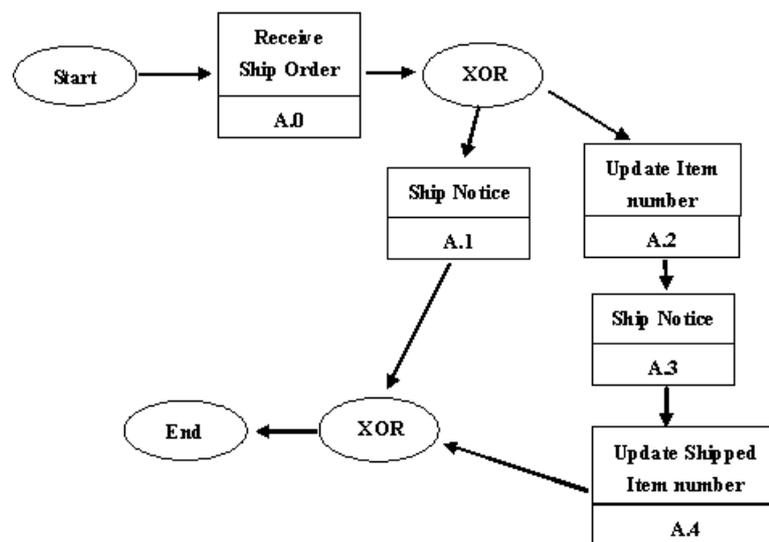
Figure 7.4: The Shipping service process

provider are involved in the process and interact in following way:

- After a *shipping service provider* receives a ship order from a *shipping service customer*, it starts processing the shipping request.

- The *shipping service provider* ships the items away and keeps sending ship notices to *shipping service customers* until all the items are done.

This process is chosen for the test of our language mapping based approach is because that it covers most of the important BPEL4WS syntax ($< receive >, < invoke >, < assign >, < sequence >, < switch >, < while >$) although it is not complicated and it is well written (it is translatable according to the principles give in Chapter 4. The automatically generated LCC protocol using our system is given in appendix D.4. The generated LCC protocol is tested on Linda server[Rob04] that is a Prolog based multi-agent simulation platform (message passing is performed locally). Two types of testing are undertaken:

- One-to-One based interaction: Only two agents, namely *shipping service provider* and *shipping service customer*, are created and different data instances are used to prove the correctness of the LCC protocol generated when it is used to guide the interaction. Branches and iterations as defined in the original BPEL4WS process model are all well performed by the two agents. Desired outputs are through the agents' interaction based on different inputs.

- Many-to-one based interaction: One *shipping service provider* agent and many *shipping service customer* agents are created. Different *shipping service customer* agents send shipping requests to *shipping service provider* agent randomly (*shipping service provider* agent may receive requests from different customer simultaneously). Results of the interactions that takes place between *shipping service provider* and each *shipping service customer* are also proved to be correct.

From this case study we can conclude that for those translatable BPEL4WS process models, using our language mapping based approach, the LCC protocols derived can be finely used to guide multi-agent interactions.

7.3 Case Study 3: Health Care Process

In Section 7.1, we discusses how our system supports complete workflow processes. In this section, support for incomplete workflow processes is demonstrated using a case of health care process, which is also used as an example to explain the concept of incomplete process in Chapter 6.
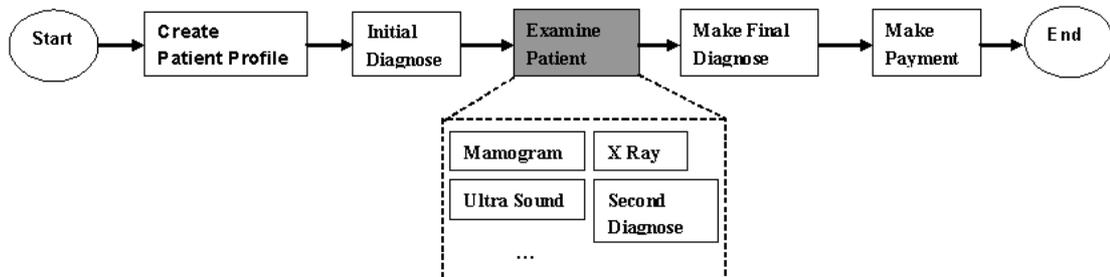
Figure 7.5: The healthy care process

Figure 7.5 shows an incomplete process representing a typical diagnostic process for investigation of patients in hospital (the initial BPEL4WS model with the syntax that supports incomplete process of scenario can be found in appendix D.5. A new patient coming into the centre will first be created a profile according to his/her current status via a registration service. All patients then consult with an attending physician for the first diagnosis, which will determine what tests need to be performed, on a case-by-case basis. This introduces the flexibility into the process. After the tests, the patient is called again by the attending physician who explains the results of the tests and makes a diagnosis. The patient is then required to report to accounts to make the required payment before leaving the centre. In this process, the activity that is depicted in the gray box is an incomplete activity. It can not be pre-defined until the physician sees the real situation of the patient. Our system can serve as an effective platform for the health care process for the following reasons:

- First, this health process aims are normally achieved through the accomplishment of individual services, which have inherently logical relationships and should be performed in a certain order. Thus, the conduct of healthy care can be easily modelled as a process and our system can provide automated support for deployment of such a process. As this process can be modelled, it is supported by conventional workflow management system as well once the process model is formalised.

- Second, most of the non-trivial health care process are based on collaborative work. A number of health care departments, who focus on various health care tasks, are involved. These departments, sometimes geographically distributed, should be coordinated properly so that health care process can be passed from one department to another, according to a set of defined rules. Obviously, such coordination can be well supported by our system to improve efficiency and productivity as addressed in previous case study.

- Third, there is a vast amount of communication amongst patients and medical departments for coordination purposes. Our system also reflects this feature well with automatic coordination.

- Finally, there are uncertain activities that are not clear at the process design time (*examine patient*) and their executions fully reply on instance data of previous activity (*initial diagnose*). Our approach and system can support this feature well since they fully exploit individual agent's knowledge and capabilities.

Obviously, at the early stage of health care process, although the goals and expected outcomes of *examine patient* are expressed, the activities of achieving it through a set of steps remains uncertain. The particular patient examine tasks can be gained only after *first diagnose* has been completed. In other words, the decomposition of *examine patient* into sub-processes should be performed on-the-fly at run-time. In the view of this situation, the execution of health care process falls into the category of controlled incomplete composite activity's instantiation (see Chapter 6 for detail) since the sub-activities of *examine patient* and their composition depend on activity *initial diagnose* and the instantiating process is performed by one agent (attending physician). For example, one of the possible complete process instance might be as depicted in Figure 7.6 (its formal representation can be found in appendix D.5.2) if *examine patient* is instantiated by two concrete activities (*X Ray checking, ultra sound checking*) that are executed in sequential order after the initial diagnosis is made.
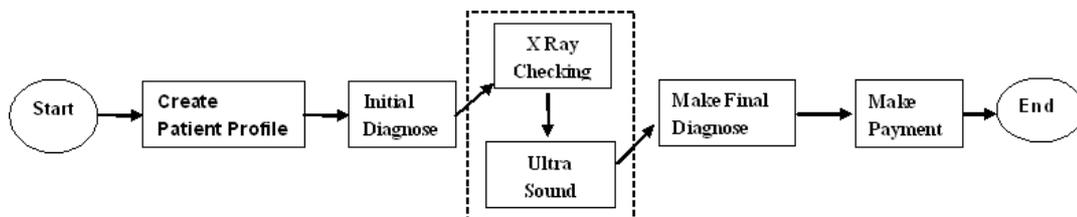
Figure 7.6: A possible complete healthy care process instance

In this example, when *patient* interacts with *attending physician* for initial diagnose, it knows nothing about what examination it needs to take or technically speaking, it knows nothing about the un-process process model. Therefore, at the time point when the *patient* agent processes the *initial diagnose* activity, whether the whole process is complete or incomplete doesn't matter. After the *attending physician* receives the un-processed model from the *patient*, it instantiates the incomplete activity in the unprocessed model and sends it back according to the definition in the completed process model. Agent *patient* then processes the received process model and makes the whole process continue. We can see in this process, the *patient* agent is totally unaware of the incompleteness of initial process model. In addition, since in our system, the process model is distributed amongst agents, different instantiations of the incomplete process instance will not affect the consistency.

7.4 Summary

Three case studies are used in this chapter to demonstrate the applicability of the key ideas presented in this thesis. The studies have shown that the design rationale of our multi-agent based workflow clearly. The analysis also demonstrates that the multi-agent based workflow approach proposed in this thesis is applicable for supporting both conventional (complete), and non-conventional (incomplete processes).

Chapter 8

Discussion

This chapter discusses the both of the positive and negative sides of our work, multi-agent based decentralised workflow enactment mechanisms proposed in this thesis. The advantages of our approach are suggested in Section 8.1 and the tradeoffs of using the multi-agent computing paradigm in comparison to the conventional system architecture (client-server architecture) are discussed in Section 8.2. A discussion of application domains in which the multi-agent based workflow management system might be adopted and might perform better than others is given in Section 8.3.

8.1   Discussion of the Advantages of This Research

It has been well recognised that business processes are important and crucial in all organisations. Workflow management for business processes is becoming a more and more important part of organisational information systems. However, the current situation is that most workflow systems suffer from problems of poor performance, single point failures, limited openness and lack of sufficient incomplete process support. Furthermore, as inter-enterprise e-commerce becomes more and more complex, requirements emerge for open systems, in which each participants can join, contribute to and leave the interaction on their own will. With conventional workflow systems, such requirements can not achieved easily.

The approaches presented in this thesis treats all the above problems from a novel point of view. We believe that most of those problems, if not all, are caused by the mismatch between system requirements and system realisation. Therefore, a centralised management architecture that is not well suitable for decentralised workflow applications needs to be replaced by an open, collaborative, and decentralised one while the

existing business rationales for building workflow management systems should not be affected by the change of the system architecture.

Based on this observation, the target of this research is to address these unsolved problems by exploiting features of multi-agent technology that provide a decentralised architecture to support workflow. As a result, a new framework and corresponding process coordination technologies are presented. The advantages of our proposed approach are summarised as follows:

- Two different computing paradigms, namely workflow management systems and multi-agent systems are linked. With our approaches, multi-agent based workflow management systems can be constructed faster than before since the existing work (modeling tools, existing business process models etc) can continue to be used in the larger design process and the possibility of acceptance of new system is largely improved.

- Our system enables better system performance because within our framework, both data and process control are cleanly distributed and the computing and storage capabilities of each agent are well utilised. The decentralised coordination mechanism proposed is very light weight and cost effective. In addition, the dynamic work assignment based on message passing, which is another distinguishing feature, also contributes to achieving better system performance and increasing agility as addressed in chapter 5.

- Multi-agent based workflow system avoids some of the risks of client-server approaches. It is more stable in the circumstances where the whole system fails simply because some individual points (bottlenecks) fail. The possibility of one point failure of individual agent is reduced in our system since the computation and communication are relatively better balanced between all the participants; Dead agents in the system can be discovered by others and the system can then be made alive by replacing the dead point.

- System openness is also improved as agents represent a loosely coupled computing paradigm. Virtual communities should be open and dynamic so that workflow participants can join and leave on their own initiative. Our approach loosens restrictions on workflow participants. Workflow participants, represented by agents, are autonomous in the system. With essential data, agents are able to participate in workflow systems more actively and the behaviours of workflow

agents do not require updating the centralised workflow server as with a conventional workflow system architecture. New agents can join the system at any time and through any existing agent without affecting the current status of whole system.

- Our approach utilises novel techniques involving multi-agent execution of workflow processes. This research shows useful results for research topics such as Web service based workflow and Grid workflow. Certain requirements such as agent based service interaction were found lacking in the existing Web services and Grid services technologies [SKL02]. With our approach, computing units like web services can be used to provide external behaviours for agents, thus can be adopted into our existing system. This further increases the openness of system and support service-oriented workflow well. In addition, the composition and execution of Web services can also be facilitated properly by agents.[BBN02, GPW03].

- Our approach provides possible support for incomplete processes. By using decentralised task decomposition, support for incomplete processes is embedded within the system framework. This extended feature makes our system capable of supporting processes in some of the non-conventional workflow domains in which processes cannot be completely designed in advance.

8.2 Discussion on the Tradeoffs of the Proposed Approach

Change from conventional workflow architecture to multi-agent based system architecture brings some tradeoffs which show potential limitations. Some of the tradeoffs of the proposed approaches in this thesis are summarised as follows:

- The execution of the workflow process is decentralised while the ability of concurrent computing that is supported by conventional workflow management system is no longer supported by our multi-agent based platform. Business process models have to passed between agents and are executed in sequential order. As explained in chapter 5, all the parallel structures defined in a process model need to be converted into sequence structures which execution are the sub-set of the parallel structures' execution set.

- Management and monitoring of workflow execution may become more difficult in a multi-agent based workflow system. Extra agents (say administrating agents) need to be developed for administrating purpose in order to collect the related information (current states of agents.) by communicating with workflow agents. If the administrating agent has to be designed as a special agent, it would become a new bottleneck of the whole system, which is what we try to avoid. However, if any common workflow agent in the system has the functions of managing and monitoring, the complexity of individual agent design will be largely increased.

- The ability to handle exceptions and erroneous situations may be difficult with multi-agent based workflow system. Unlike conventional workflow systems with which errors and exceptions can be detected and solved by centralised servers, more complicated mechanisms such as how to detect and handle unexpected exceptions are required. These are future work.

- Multi-agent based open systems enable networked access to resources. This can bring security problems [VAM01]. In particular, with our approaches, every agent has the right to access the activities defined in the process model being passed no matter whether they are designated for those activities. Therefore, issues of authentication and security are a major concern for particular applications.

8.3   Discussion on Suitable Application Domains of MAS Based Workflow Management System

With the discussion of the advantages and some of the possible disadvantages of our approaches presented in Sections 8.1 and 8.2, we can see that multi-agent based workflow systems may perform better for some application domains but not for all. Therefore, an analysis of the application domains where our approaches and system are more suitable can provide a better understanding of multi-agent based workflow systems.

In general, our approach is capable of providing better support for standard workflows providing the advantages outlined in Sections 8.1. Also, it is capable of providing certain support for incomplete processes based application. In particular, with the widespread deployment of wireless technologies, the next phase of electronic business

growth will be in the area of wireless and mobile e-commerce. The existing mobile network infrastructure provides an open environment for running large scale applications. Such applications can be viewed as multi-agent systems in which each mobile device is viewed as an agent from technical point of view.

Multi-agent technology is thus recommended as a well-suited software paradigm for such mobile devices based system. Mobile agents are able to travel between platforms to fulfil their tasks at different locations. Decentralisation helps to cope with the complexity problem of service infrastructures. However, when adopting the research result from existing multi-agent world to the mobile agents based system, new problems emerge due to the features of mobile devices (limited computing ability, high mobility, huge numbers, etc). With our work, the mobile devices can be used to deploy workflow system. The mobile agents can be completely dummy agents because the issues of how they are about to communicate with each other and perform business functions at the right time are pre-defined in the message package. Consequently, no complex functions for controlling the coordination between mobile agents need to be designed inside each mobile agent. The mobile agent thus can be used to deploy workflow management system with more mobility. Another possible application direction of our approaches is the extension of web services. Currently, web services can only be invoked passively as service provider without any initiative. Possible extension can be made using our approaches to add very lightweight layers on top of web services to enable them work initiatively.

For those application domains that require heavy parallel computation, our system doesn't fit so well since agents in our system can only perform the designated tasks in sequential order, which might reduce the overall system performance.

Chapter 9

Conclusions and Future Work

9.1 Summary of This Thesis

The objective of this thesis was to develop an innovative system architecture and process coordination mechanism for multi-agent based, decentralised, workflow management systems with high level business rationale kept. The thesis was organised as follows:

- Chapter 1 introduced workflow concepts as well as the state-of-the-art of workflow. It also described the purposes of this work, the key issues addressed in this thesis and the structure of this thesis.

- Chapter 2 analysed the some of the existing research problems in conventional workflow system in detail. Based on the problems analysis, we observed that most, if not all, of these problems are caused by the conventional system architecture (client-server based architecture) in open environment like the internet. After reviewing the some of major related work, we believe that workflow's increasingly distributed nature in open environment can be reflected better by a multi-agent based, decentralised and collaborative system architecture. Also it is argued that underlying system architecture change should not affect the upper level business rationale. This is the philosophy of this research.

- Chapter 3 proposed a framework for modelling multi-agent system protocols starting from a high level process model. With this framework, a process model can be used as a basis for protocol property verification. A simple language SPPC is defined for property checking purposes and any protocol model defined by SPPC can be translated into an existing protocol language(in this case LCC).

Using this framework, much effort can be saved in the process of MAS protocol modelling since some requirements level errors can be discovered using automatic verification, which is different from the typical protocol modelling engineering method. Furthermore, using this approach, any revision to an existing protocol can also be checked quickly to make sure all the business logic level changes are correct.

- Chapter 4 discussed how to develop protocol based multi-agent systems using executable business process models. Language mapping is performed between a business process modelling language (BPEL4WS) and an interaction protocol language (LCC) to generate the protocol used in MAS from business process model. Since the gap between them is quite huge, we use another modelling language (SPPC) as an intermediary. First we perform a language mapping between BPEL4WS and SPPC, then the derived SPPC model can be translated into a LCC protocol automatically. 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 another modelling language which is based on different paradigm. For example, the computing activities defined at the end of a $< sequence >$ activity in BPEL4WS can not be easily translated in to 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 restrictions mean that only some BPEL4WS specifications (those conforming to the language mapping rules) can be used for protocol based MAS development, which makes the approach discussed in this chapter incomplete. In fact, language mapping based completeness is very hard to achieve (even for particular business process modelling languages) since different business process modelling languages and protocol modelling languages may be based on different computing paradigms.

- Chapter 5 provided an approach to build a multi-agent based distributed workflow system starting from a business process model rather than from a interaction protocol, which narrows the gap between the high level requirement and system specification in the development of a multi-agent system and connects the business workflow community and multi-agent community. The LCC pro-

tocol used to interpret BPEL4WS models is independent of any specific message passing infrastructure, although it has been described with respect to a distributed and multi-agent system infrastructure, it could equally well be deployed in a more traditional server based style. Furthermore, the protocol can be used prior to deployment in order to predict behaviours and possible errors in interaction[Wal04b]. Another advantage is that the workflow engine built using our approach is a generic server. The only specific knowledge it needs is about how to process the LCC protocol and how to invoke the web services, but not about how to process the particular business process modelling language. This gives us a very efficient and direct way for system re-design and re-implementation. Even more generally, this approach can be in particular used to adopt any functional requirement, as long as the requirement is operational and can be represented by message passing, on a multi-agent platform.

- Chapter 6 extends our decentralised multi-agent platform to support incomplete processes. The causes of incomplete processes have been identified and conventional workflow system's inability to support incomplete processes has been analysed. By introducing the instantiation activities, run-time instantiation of properties missing activities is modelled as an essential step in the process and integrated into the decentralised architecture. The missing components can then be completed using agents' internal intelligence or the cooperation of a group of agents. From an engineering point of view, this approach is justifiable because the ordinary workflow participants may and should not be given an interface to specify a composite task using a complex workflow modelling language. From a system coordination viewpoint, the instantiation task is distributed. Thus, process modelling at run-time can be performed with the support of mechanisms for completing processes, at either instance- or process-level.

- Chapter 7 presents three case studies to demonstrate the applicability of the key ideas presented in this thesis. The studies shows the design rationale of our multi-agent based workflow. The analysis also illustrates how the multi-agent based workflow approach proposed in this thesis is applicable for supporting both conventional (complete), and non-conventional (incomplete processes).

- Chapter 8 discusses the both of the positive and negative sides of our work, multi-agent based decentralised workflow enactment mechanisms proposed in this thesis.

9.2   Contributions of This Thesis

The significance of this research is that it tackles some of the unsolved problems in the workflow area from the system architecture point of view. Based on existing work from multi-agent world, this research combines a new system architecture (multi-agent based) and process coordination technologies for deploying workflow systems comprehensively, which can be considered as a paradigm change. This new framework and corresponding technologies exploits the features provided by multi-agent computing technology in order to better reflect the distributed nature of current workflow. This research contributes to the challenging research area of multi-agent based workflow, which opens new ground in workflow research, and the process support area in general. The main outcome of this research is using a decentralised, open and multi-agent architecture to deploy distributed workflow systems without affecting the existing workflow constructing rationale (starting from a business process model). Therefore, this research shows that emerging technologies such as multi-agent system for workflow system support provide critical features. Moreover, the new system architecture proposed in this research changes the way that all the participants are involved in coordination in conventional workflow management (from contributing their services to the coordination passively to providing their own services to the workflow initiatively). The major contributions of this thesis are:

- **Identifying the causes of the existing problems in conventional workflow management systems**. We have analysed that most of the existing problems in conventional workflow management systems are caused by the mismatch between application nature and underlying system architecture. Based on this argument, it has been found that multi-agent computing technology can be used as a underlying infrastructure to support workflow applications better.

- **Approaches for adopting existing work for conventional workflow system development on multi-agent platform** Although multi-agent system has shown more valuable and natural features for distributed workflow system's enactment, it can be hardly accepted by end users if everything existed has to be re-designed/re-implemented to fit the new computing features. The approaches proposed by us in this thesis finely bridge the gap between conventional workflow developing rationale and multi-agent system. Therefore, almost all the existing work can be adopted for the development of multi-agent based workflow management system.

- **A multi-agent based system architecture and design for decentralised workflow management systems.** Using multi-agent based computing technology to support distributed workflow is considered as a paradigm shift. However, the system designs of the few existing so called multi-agent/p2p based workflow approaches are normally incomplete, or even not complete multi-agent based. The proposed approaches, have contributed a relative complete, concrete, and fully decentralised system design methods for deploying multi-agent based workflow applications.

- **A prototype implementation of a multi-agent based distributed workflow management system.** A multi-agent based pure decentralised workflow management system prototype is developed for the proof of concept purpose. It can read two types of system specifications namely LCC protocol and BPEL4WS model to direct the interaction of the agents (participants). The system is also able to adopt external web services thus can help to realise service-oriented architecture. This prototype we believe serves as a fine basis for future extension for real world workflow management systems.

9.3 Future Work

In future, further investigation into multi-agent based decentralised workflow will be carried out. Future research includes adopting agents' intelligence into the whole workflow system. At present, all the agents in our system as explained are dummy agents which can only perform required tasked following the instruction in the process model/interaction protocols. More intelligent agents that are adaptive and are aware of the changes of state of environment/context can be very helpful for performing workflow management and monitoring tasks.

As indicated in Chapter 6, our system prototype is for the purpose of demonstration and proof of concept only. After certain extension and improvement, more real-world applications should be developed based on this prototype in order to collect more concrete results. Thus, a more sophisticated comparison of different workflow systems, either centralised or decentralised can be performed.

Some other crucial factors such as organisational management, run time verification of workflow instances in a decentralised manner, security of multi-agent based workflow are currently ignored to make the problem simpler. In order to have a practi-

cal workflow solution, more research should be carried out on all of the above issues.

Appendix A

Algorithm Description Language

The syntax of the language that we use to describe the algorithm in the thesis is explained here. All the algorithms are defined as procedures.

$$
\begin{aligned}
procedure &::= \textit{Name}, \textit{Arguments}, \textit{Body} \\
Name &::= \textit{String} \\
Arguments &::= \textit{Inputs}, \textit{Outputs} \\
Inputs &::= \textit{Any legal Term} \\
Outputs &::= \textit{Any legal Term} \\
Body &::= \textit{Declaration\_Sequence} | \textit{Statements\_Sequence} \\
Declaration\_Sequence &::= \textit{Declaration of the arguments used} \\
Statements\_Sequence &::= \textit{if\_statement} | \textit{while\_statement} | \\
& \quad \textit{for\_statement} | \textit{any computing clause} \\
if\_statement &::= \textit{if}(\textit{boolean\_expression}) \\
& \quad \textit{Statements\_Sequence} \\
& \quad \left[\begin{aligned} &\textit{else if}(\textit{boolean\_expression}) \\ &\quad \textit{Statements\_Sequence} \end{aligned} \right] \\
while\_statement &::= \textit{while}(\textit{boolean\_expression}) \\
& \quad \textit{Statements\_Sequence} \\
for\_statement &::= \textit{for}(\textit{list of element}) \\
& \quad \textit{Statements\_Sequence}
\end{aligned}
$$

Appendix B

Representing BPEL4WS Model in Plain Text

$$
\begin{aligned}
Model \quad &:= \quad \{Scope\} \\
scope \quad &::= \quad \{description([Description,...]),Structure\} \\
Description \quad &::= \quad partnerLink\left(\begin{array}{l} name(Constant), parnterLinkType(Constant), \\ myRole(Constant), partnerRole(Constant) \end{array}\right) \\
&\qquad |variable(name(Constant),messageType(Constant)) \\
&\qquad |faultHandlers\left(\left[\begin{array}{l} catch(faultName, faultVariable, Activity), \\ ...,catchAll(Activity) \end{array}\right]\right) \\
&\qquad |compensationHandler(Activity) \\
Structure \quad &::= \quad scope([Description,...],Structure/Activity)| \\
&\qquad flow(Activity/Structure,Activity/Structure,...)| \\
&\qquad switch(condition(Condition,Activity/Structure),...)| \\
&\qquad while(condition(Condition,Activity/Structure)| \\
&\qquad Structure/Activity then Structure/Activity| \\
&\qquad pick\left(\begin{array}{l} onMesssage(partnerLink(Constant), portType(Constant), \\ operation(Constant),variable(Constant),Activity), \\ onAlarm(for(duration-expr), \\ until(deadline-expr),Activity) \end{array}\right)
\end{aligned}
$$

$$
\begin{aligned}
\textit{Activity} \quad ::= \quad &\textit{invoke} \left(\begin{array}{l}
\textit{partnerLink(Constant)}, \\
\textit{portType(Constant)}, \\
\textit{operation(Constant)}, \textit{inputVariable(Constant)}, \\
\textit{outputVariable(Constant)}, \textit{sourceLink(Constant)}, \\
\textit{targetLink(Constant))}
\end{array}\right) \\[1em]
&|\textit{receive} \left(\begin{array}{l}
\textit{partnerLink(Constant)}, \textit{portType(Constant)}, \\
\textit{operation(Constant)}, \textit{variable(Constant)}, \\
\textit{sourceLink(Constant)}, \textit{targetLink(Constant)}
\end{array}\right) \\[1em]
&|\textit{reply} \left(\begin{array}{l}
\textit{partnerLink(Constant)}, \textit{portType(Constant)}, \\
\textit{operation(Constant)}, \textit{variable(Constant)}, \\
\textit{faultName(Constant)}, \\
\textit{sourceLink(Constant)}, \textit{targetLink(Constant)}
\end{array}\right) \\[1em]
&|\textit{assign} \left(\begin{array}{l}
\textit{from} \left(\begin{array}{l}
\textit{expression/opaque/variable(Constant)}, \\
\textit{property(Constant)}
\end{array}\right), \\
\textit{to(variable(Constant)}, \textit{property(Constant))}, \\
\textit{sourceLink(Constant)}, \textit{targetLink(Constant)}
\end{array}\right) \\[1em]
&|\textit{throw} \left(\begin{array}{l}
\textit{faultName(Constant)}, \textit{faultVariable(Constant)}, \\
\textit{sourceLink(Constant)}, \textit{targetLink(Constant)}
\end{array}\right) \\[1em]
&|\textit{wait} \left(\begin{array}{l}
\textit{for(Constant)}, \textit{until(Constant)}, \\
\textit{sourceLink(Constant)}, \textit{targetLink(Constant)}
\end{array}\right) \\[1em]
&|\textit{terminate(sourceLink(Constant)}, \textit{targetLink(Constant)))}| \\
&\textit{empty(sourceLink(Constant)}, \textit{targetLink(Constant))} \\[0.5em]
\textit{Condition} \quad ::= \quad &\textit{Term}|\textit{Condition} \wedge \textit{Condition}|\textit{Condition} \vee \textit{Condition} \\
\textit{Constant} \quad ::= \quad &\textit{Term}
\end{aligned}
$$

Appendix C

Prolog Definitions For All the Constraints Used in LCC Interpreter

C.1   Constraints Used For Role $a(receiver(Role), ID)$

$$extract\_activity(Model, Activity) :-$$
$$Model = ..[scope, \_, Model_1],$$
$$extract\_activity(Model_1, Activity).$$

$$extract\_activity(Model, Model).$$

$$update\_variable(M, [Head|Rest], VList_1) :-$$
$$compare\_variable(M, Head),$$
$$VList_1 = [M|Rest].$$

$$update\_variable(M, [Head|Rest], VList_1) :-$$
$$update\_variable(M, Rest, VList_1).$$

$$update\_variable(M, [], VList_1).$$

C.2 Constraints Used For Role $a(interpreter(...), ID)$

$$is\_receive(Model, Role) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[receive, myRole(Role)|\_]$$

$$is\_reply(Model, Role) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[reply, myRole(Role)|\_]$$

$$is\_invoke(Model, Role) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[invoke, myRole(Role)|\_]$$

$$is\_assign(Model) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[assign|\_]$$

$$is\_throw(Model) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[throw|\_]$$

$$is\_sequence(Model) : -$$
$$\quad extract\_activity(Model, Activity),$$
$$\quad Activity = ..[then|\_].$$

$$is\_switch(Model) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[switch|\_]$$

$$is\_while(Model) : -$$
$$\quad extract\_activity(Model, Activity)$$
$$\quad Activity = ..[while|\_]$$

C.3   Constraints Used For Role $a(receive(...), ID)$

$$process\_receive\_message \left(\begin{array}{l} PartnerRole, PortType, Operation, null, \\ ID, VList, IDList, VList, IDList_1 \end{array} \right) :-$$

$$append(PartnerRole : PortType : Operation : null : ID, IDList, IDList_1).$$

$$process\_receive\_message \left(\begin{array}{l} PartnerRole : PortType, Operation, Variable, \\ ID, VList, IDList, VList_1, IDList_1 \end{array} \right) :-$$

$$append(Variable, VList, VList_1),$$

$$append(PartnerRole : PortType : Operation : Variable : ID, IDList, IDList_1).$$

$$check\_receive(Model, PortType, Operation, Variable, PartnerRole) :-$$

$$extract\_activity(Model, Activity),$$

$$Activity = .. \left[\begin{array}{l} receive, \_, partnerLink(PartnerLink), portType(PortType), \\ operation(Operation), variable(Variable), \\ sourceLink(\_), targetLink(\_) \end{array} \right],$$

$$partnerLink(name(PartnerLink), , myRole(), partnerRole(PartnerRole)).$$

C.4 Constraints Used For Role $a(reply(...),ID)$

$process\_reply(Model, Partner, PortType, Operation, null, Fault) : -$
$\quad extract\_activity(Model, Activity)$
$\quad Activity = ..[\_, partnerLink(PartnerLink), portType(PortType),$
$\quad\quad\quad operation(Operation), variable(null), faultName(Fault), \_, \_],$
$\quad partnerLink(name(PartnerLink), \_, myRole(\_), partnerRole(Partner)).$

$process\_reply(Model, Partner, PortType, Operation, Variable, \_) : -$
$\quad extract\_activity(Model, Activity)$
$\quad Activity = ..[\_, partnerLink(PartnerLink), portType(PortType),$
$\quad\quad\quad operation(Operation), variable(Variable), \_, \_, \_],$
$\quad partnerLink(name(PartnerLink), \_, myRole(\_), partnerRole(Partner)).$

$get\_ID([], \_, \_, \_, \_,) .$

$get\_ID([Head|Rest], Partner, PortType, Operation, Variable, ID) : -$
$\quad Head = Partner : PortType : Operation : Variable : ID.$

$get\_ID([Head|Rest], Partner, PortType, Operation, Variable, ID) : -$
$\quad get\_ID(Rest, Partner, PortType, Operation, Variable, ID).$

$loop\_up([], \_, \_).$

$look\_up([Head|Rest], Variable, Head) : -$
$\quad Head\,is\,of\,the\,same\,type\,and\,same\,variable\,name\,with\,Variable.$

$look\_up([Head|Rest], Variable, Variable_1) : -look\_up(Rest, Variable, Variable_1).$

C.5   Constraints Used For Role $a(invoke(...), ID)$

$process\_invoke(Model, PortType, Operation, null, VList, Role) : -$

$$Model = .. \begin{bmatrix} invoke, partnerLink(PartnerLink), portType(PortType), \\ operation(Operation), inputVariable(null), \_, \_, \_ \end{bmatrix},$$

$partnerLink(PartnerLink, \_, \_, parterRole(Role)).$

$process\_invoke(Model, PortType, Operation, InputVariable, VList, Role) : -$

$$Model = .. \begin{bmatrix} invoke, partnerLink(PartnerLink), portType(PortType), \\ operation(Operation), inputVariable(Variable), \_, \_, \_ \end{bmatrix},$$

$look\_up(Variable, VList, InputVariable),$

$partnerLink(PartnerLink, \_, \_, parterRole(Role)).$

C.6   Constraints Used For Role $a(assign(...), ID)$

$process\_assign(Model, VList, VList_1) :-$
  $extract\_activity(Model, Activity),$
  $Activity = ..[copy, List],$
  $process\_assign\_copy(List, VList, VList_1).$

$process\_assign\_copy([], \_, \_).$

$process\_assign\_copy([Head|Rest], VList, VList_1) :-$
  $Head = ..[Type, From, To],$
  $process\_from(From, Value),$
  $update\_dataSet(VList, Variable_1, Part, To,, VList_2),$
  $process\_assign\_copy(Rest, VList_2, VList_1).$

$process\_assign\_copy([Head|Rest], VList, VList_1) :-$
  $Head = ..[Type, From, To],$
  $process\_from(From, Value),$
  $update\_dataSet(VList, To, Value, VList_2),$
  $process\_assign\_copy(Rest, To, Value, VList_1).$

$process\_assign\_copy([Head|Rest], VList, VList_1) :-$
  $Head = ..[Type, From, To],$
  $process\_from(From, Value),$
  $To = ..[to, partnerLink(PartnerLink), \_],$
  $partnerLink(PartnerLink, \_, \_, partnerLink(Value)),$
  $process\_assign\_copy(Rest, To, Value, VList_1).$

$process\_from(From, Value) :- From = ..[\_, variable(Variable), part(Part)],$
  $lookup\_dataSet(VList, Variable, Part, Value).$

$process\_from(From, Value) :- From = ..[\_, variable(Variable), property(Property)],$
  $lookup\_dataSet(VList, Variable, Property, Value).$

$process\_from(From,Value) : -From = ..[\_,expression(Value)].$

$process\_from(From,Value) : -From = ..[\_,opaque(Value)].$

$process\_from(From,Value) : -$
$From = ..[\_,partnerLink(PartnerLink),endpointReference(myRole)],$
$partnerLink(PartnerLink,\_,myRole(Value),\_).$

$process\_from(From,Value) : -$
$From = ..[\_,partnerLink(PartnerLink),endpointReference(partnerRole)],$
$partnerLink(PartnerLink,\_,\_,partnerRole(Value)).$

$lookup\_dataSet([Head|Rest],Variable,Part,Value) : -$
$check\,if\,Head\,and\,Variable\,are\,of\,the\,same\,message\,type,$
$get\,the\,Value\,of\,Head's\,Part.$

$lookup\_dataSet([Head|Rest],Variable,Part,Value) : -$
$lookup\_dataSet(Rest,Variable,Part,Value).$

$lookup\_dataSet([Head|Rest],Property,Value) : -$
$check\,if\,Head\,and\,Variable\,are\,of\,the\,same\,message\,type,$
$get\,the\,Value\,of\,Head's\,Property.$

$lookup\_dataSet([Head|Rest],Property,Value) : -$
$lookup\_dataSet(Rest,Property,Value).$

$update\_dataSet(VList,To,Value,VList_1) : -$
$To = ..[variable(Variable,null)],$
$update(VList,Variable,null,VList_1).$

$update\_dataSet(VList,To,Value,VList_1) : -$
$To = ..[variable(Variable,part(Part))],$
$update(VList,Variable,Part,VList_1).$

$$update\_dataSet(VList, To, Value, VList_1) : -$$
$$To = ..[variable(Variable, property(Property))],$$
$$update(VList, Variable, Property, VList_1).$$

$$update([Head|Rest], Variable, null, \_, VList_1) : -$$
$$If\,Head\,and\,Variable\,are\,of\,the\,same\,message\,type,$$
$$VList_1 = [Variable|Rest].$$

$$update([Head|Rest], Variable, null, \_, VList_1) : -$$
$$update(Rest, Variable, null, \_, VList_2),$$
$$VList_1 = [Head|VList_2].$$

$$update([Head|Rest], Variable, Part, Value, VList_1) : -$$
$$If\,Head\,and\,Variable\,are\,of\,the\,same\,message\,type,$$
$$update\,the\,Value\,of\,Head's\,Part,$$
$$VList_1 = [Variable|Rest].$$

$$update([Head|Rest], Variable, Part, Value, VList_1) : -$$
$$update(Rest, Variable, Part, VList_2),$$
$$VList_1 = [Head|VList_2].$$

$$update([Head|Rest], Variable, Property, Value, VList_1) : -$$
$$Head = ..[variable(Variable, property(Property, \_))],$$
$$VList_1 = [Variable|Rest].$$

$$update([Head|Rest], Variable, Property, Value, VList_1) : -$$
$$update(Rest, Variable, Property, VList_2),$$
$$VList_1 = [Head|VList_2].$$

C.7   Constraints Used For Role $a(throw(...), ID)$

$process\_throw(Model, FaultHandlingActivity) : -$
  $extract\_activity(Model, Activity)$
  $Activity = ..[throw, faultHandler(FaultHandler), faultVariable(FaultVariable)],$
  $find\_fault\_handler(Model, FaultHandler, FaultVariable, FaultHandlingActivity).$

$find\_fault\_handler(Model, FaultHandler, FaultVariable, FaultHandlingActivity) : -$
    $extract\_description(Model, List),$
    $process\_description\_list(List, FaultHandler, FaultVariable, FaultHandlingActivity).$

$extract\_description(Model, List) : -$
        $Model = ..[scope, DList, Model\_1],$
        $extract\_description(Model\_1, List\_1),$
        $List = [DList | List\_1].$

$extract\_description(\_, []).$

$process\_scope\_list([], \_, \_, null).$

$process\_scope\_list \left(\begin{array}{l} [Head | Rest], FaultHandler, \\ FaultVariable, FaultHandlingActivity \end{array} \right) : -$
    $process\_description\_list(Head, FaultHandler, FaultVariable, null),$
    $process\_scope\_list(Rest, FaultHandler, FaultVariable, FaultHandlingActivity).$

$process\_description\_list \left(\begin{array}{l} [Head | Rest], FaultHandler, \\ FaultVariable, FaultHandlingActivity \end{array} \right) : -$
    $Head = ..[faultHandlers, catchList],$
    $process\_catch\_list(catchList, FaultVariable, FaultHandlingActivity).$

$process\_catch\_list([], \_, \_, null).$

$process\_catch\_list([catchAll(FaultHandlingActivity)], \_, \_, FaultHandlingActivity).$

$process\_catch\_list([Head|Rest], FaultHandler, FaultVariable, FaultHandlingActivity) : -$
$\quad\quad Head = ..[catch, faultHandler(FaultHandler), faultVariable(FaultVariable),$
$\quad\quad\quad\quad FaultHandlingActivity].$

$process\_catch\_list([Head|Rest]) : -process\_catch\_list(Rest).$

C.8   Constraints Used For Role $a(sequence(...), ID)$

$process\_sequence(Model, Model_1, Model_2) : -$
$\quad Model = ..[scope, DList, Activity],$
$\quad compose\_activity(Activity, Activity_1, Activity_2),$
$\quad Model_1 = ..[scope, DList, Activity_1],$
$\quad Model_2 = ..[scope, DList, Activity_2].$

$process\_sequence(Activity, Activity_1, Activity_2) : -$
$\quad Activity = ..[\_, Activity_1, Activity_2].$

C.9   Constraints Used For Role $a(switch(...), ID)$

$process\_switch(Model, Model_1) : -$
 $Model = ..[scope, DList, Activity],$
 $process\_switch(Activity, Model_2),$
 $Model\_1 = ..[scope, DList, Model_2].$

$process\_switch(Model, Model_1) : -$
 $Model = ..[switch, [Head|Rest]],$
 $process\_condition([Head|Rest], Model_1).$

$process\_condition([Head|Rest], Model_1) : -$
 $Head = ..[condition(Condition), Activity],$
 $Condition\ is\ true,$
 $Model\_1 = Activity.$

$process\_condition([Head|Rest], Model_1) : -$
 $Head = ..[condition(Condition), Activity],$
 $Condition\ is\ not\ true,$
 $process\_condition(Rest, Model_1).$

Appendix D

Formal Representations Used For Evaluation

D.1 Student Registration Process Described by BPEL4WS

$< process\, name = "studentRegistrationProcess" >$
$\quad < variables >$
$\quad\quad < variable\, name = "regisForm"/ >$
$\quad\quad < variable\, name = "approvalResult"/ >$
$\quad\quad < variable\, name = "paymentForm"/ >$
$\quad\quad < variable\, name = "paymentConfirmation"/ >$
$\quad\quad < variable\, name = "accountConfirmation"/ >$
$\quad < /variables >$
$\quad < sequence >$
$\quad\quad < receive\, myRole = "enrolmentOfficer"\ parnterRole = "student"$
$\quad\quad\quad\quad portType = "registrationPT"\ operation = "receiveRegistration"$
$\quad\quad\quad\quad variable = "regisForm"/ >$
$\quad\quad < invoke\, myRole = "enrolmentOfficer"\ parnterRole = "courseAdvisor"$
$\quad\quad\quad\quad portType = "approvalPT"\ operation = "approval"$
$\quad\quad\quad\quad inputVariable = "regisFom"\ outputVariable = "approvalResult"/ >$
$\quad\quad < switch >$

```
< case condition = "approvalResult = TRUE" >
  < sequence >
    < flow >
      < sequence >
        < invoke myRole = "enrolmentOfficer" partnerRole = "treasurer"
                portType = "paymentPT" opeartion = "pay"
                inputVariable = "paymentForm"/ >
        < receive myRole = "enrolementOfficer" partnerRole = "treasurer"
                portType = "paymentConfirmPT" opeartion = "confirmPayment"
                variable = "paymentConfirmation"/ >
      < /sequence >
      < sequence >
        < invoke myRole = "enrolmentOfficer" partnerRole = "technicalStaff"
                portType = "computingAccountPT" opeartion = "setupAccount"
                inputVariable = "regisForm"/ >
        < receive myRole = "enrolementOfficer" partnerRole = "technicalStaff"
                portType = "accountConfirmPT" opeartion = "confirmAccount"
                variable = "accountConfirmation"/ >
      < /sequence >
      < receive myRole = "enrolemetnOfficer" portType = "recordRegisPT"
                operation = "recordingRegis" variable = "regisForm" >
    < /flow >
    < reply myRole = "enrolmentOfficer" portType = "registrationPT"
          operation = "receiveRegistration" variable = "registrationSucced"/ >
  < /sequence >
< /case >
< case condition = "approvalResult = FALSE" >
  < reply myRole = "enrolementOfficer" portType = "registrationPT"
        operation = "receiveRegistration" variable = "registrationFailed"/ >
< /case >
< /switch >
< /sequence >
< /process >
```

D.2 Re-written Student Registration Process Described by BPEL4WS

< process name = "studentRegistrationProcess" >
  < variables >
    < variable name = "regisForm"/ >
    < variable name = "approvalResult"/ >
    < variable name = "paymentForm"/ >
    < variable name = "paymentConfirmation"/ >
    < variable name = "accountConfirmation"/ >
  < /variables >
  < sequence >
    < receive myRole = "enrolmentOfficer" parnterRole = "student"
        portType = "registrationPT" operation = "receiveRegistration"
        variable = "regisForm"/ >
    *
        portType = "approvalPT" operation = "approval"
        inputVariable = "regisFom" outputVariable = "approvalResult"/ >
    < switch >
      < case condition = "approvalResult = TRUE" >
        < sequence >
          *
              portType = "paymentPT" opeartion = "pay"
              inputVariable = "paymentForm"/ >
          < receive myRole = "enrolementOfficer" partnerRole = "treasurer"
              portType = "paymentConfirmPT" opeartion = "confirmPayment"
              variable = "paymentConfirmation"/ >
          *
              portType = "computingAccountPT" opeartion = "setupAccount"
              inputVariable = "regisForm"/ >

$< receive\, myRole = "enrolementOfficer"\, partnerRole = "technicalStaff"$

$portType = "accountConfirmPT"\, opeartion = "confirmAccount"$

$variable = "accountConfirmation"/ >$

$< receive\, myRole = "enrolemetnOfficer"\, portType = "recordRegisPT"$

$operation = "recordingRegis"\, variable = "regisForm" >$

$< reply\, myRole = "enrolmentOfficer"\, portType = "registrationPT"$

$operation = "receiveRegistration"\, variable = "registrationSucced"/ >$

$< /sequence >$

$< /case >$

$< case\, condition = "approvalResult = FALSE" >$

$< reply\, myRole = "enrolementOfficer"\, portType = "registrationPT"$

$operation = "receiveRegistration"\, variable = "registrationFailed"/ >$

$< /case >$

$< /switch >$

$< /sequence >$

$< /process >$

D.3   Shipping Service Process Described by BPEL4WS

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

```
< 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 >
```

D.4   LCC Protocol Generated for Shipping Service Process

$a(shippingServiceCustomer(loop(m\_4)),A_1) ::$

$\quad (shippingServiceCustomerPT : shippingNotice(shipNotice)$

$\qquad\qquad <= a(shippingService(loop(m\_4)),A_2))$

$\quad then$

$\quad \left(\begin{array}{l} a(shippingServiceCustomer(loop(m\_4)),A_1) < -- \\ (itemsShipped < itemsTotal)\,and\,itemsShipped = itemsShipped + itemsCount \end{array} \right) or\,null$

$a(shippingServiceCustomer,A_3) ::$

$\quad shipRequest => a(shippingService,A_4)$

$\quad then$

$\quad \left(\begin{array}{l} shippingServiceCustomerPT : shippingNotice(shipNotice) <= a(shippingService,A_4) \\ or \\ (a(shippingServiceCustomer(loop(m\_4)),A_1) < -- \\ \qquad (itemsShipped = 0)\,and\,itemsShipped < itemsTotal) \\ or\,null \end{array} \right)$

$a(shippingService(loop(m\_4)),A_2) ::$

$\quad \left(\begin{array}{l} shippingServiceCustomerPT : shippingNotice(shipNotice) => \\ a(shippingServiceCustomer(loop(m\_4)),A_1) < -- shipNotice = yes \end{array} \right)$

$\quad then$

$\quad \left(\begin{array}{l} a(shippingService(loop(m\_4)),A_4) < -- \\ (itemsShipped < itemsTotal)\,and\,(itemsShipped = itemsShipped + itemsCount) \end{array} \right) or\,null$

$a(shippingService,A_4) ::$

$\quad shipRequest <= a(shippingServiceCustomer,A_3)$

$\quad then$

$\quad \left(\begin{array}{l} shippingServiceCustomerPT : shippingNotice(shipNotice) => a(shippingServiceCustomer,A_3) \\ < -- (shipNotice = itemsCount)\,and\,getVariableProperty(shipRequest,shipComplete) \end{array} \right)$

$\quad or$

$\quad (a(shippingService(loop(m_4)),A_2) < -- (itemsShipped = 0)\,and\,(itemsShipped < itemsTotal))$

$\quad or\,null).$

D.5   Health Care Process Described by Extended BPEL4WS

D.5.1   Initial incomplete health care process model

< process name = "healthCareProcess" >
  < variables >
    < variable name = "patientProfile"/ >
    < variable name = "examineResult"/ >
    < variable name = "patientDetail"/ >
    < variable name = "finalDiagnose"/ >
  < /variables >
  < sequence >
    *
            portType = "registrationPT" operation = "createPatientProfile"
            inputVariable = "patientDetail" outputVariable = "patientProfile"/ >
    < invoke myRole = "patient" parnterRole = "attendingPhysician"
            portType = "diagnosePT" operation = "makeFirstDiagnose"
            inputVariable = "patientProfile" **instantiating=TRUE**
            instantiated = "examine patient"/ >
    < incomplete name = "examine patient" outputVariable = examineResult"/ >

      < invoke myRole = "patient" partnerRole = "attendPhysician"
              portType = "diagnosePT" operation = "makeFinalDiagnose"
              inputVariable = "examineResult" output = "finalDiagnose"/ >
      < invoke myRole = "patient" partnerRole = "treatingPhysician"
              portType = "treatingPT" operation = "makeTreatment"
              inputVariable = "finalDiagnose"/ >   < /sequence >
    < /process >

D.5.2   A possible complete health care process instance

```
< process name = "healthCareProcess" >
  < variables >
    < variable name = "patientProfile"/ >
    < variable name = "examineResult"/ >
    < variable name = "patientDetail"/ >
    < variable name = "finalDiagnose"/ >
  < /variables >
  < sequence >
    < invoke myRole = "patient" parnterRole = "registrationService"
            portType = "registrationPT" operation = "createPatientProfile"
            inputVariable = "patientDetail" outputVariable = "patientProfile"/ >
    < invoke myRole = "patient" parnterRole = "attendingPhysician"
            portType = "diagnosePT" operation = "makeFirstDiagnose"
            inputVariable = "patientProfile" instantiating = TRUE
            instantiated = "examine patient"/ >
    < invoke myRole = "patient" partnerRole = "XRayExaminer"
            portType = "XRayPT" inputVariable = "examineResult"
            outputVariable = "examineResult"/ >
    < invoke myRole = "patient" partnerRole = "UltraSoundExaminer"
            portType = "UltraSoundPT" inputVariable = "examineResult"
            outputVariable = "examineResult"/ >
    < invoke myRole = "patient" partnerRole = "attendPhysician"
            portType = "diagnosePT" operation = "makeFinalDiagnose"
            inputVariable = "examineResult" output = "finalDiagnose"/ >
    < invoke myRole = "patient" partnerRole = "accountant"
            portType = "paymentPT" operation = "makePayment"
            inputVariable = "finalDiagnose"/ >   < /sequence >
< /process >
```

Appendix E

Publications List

Journal Papers:

- L.Guo, Dave Robertson, Yun-Heh Chen-Burger, *"Using Multi-agent Platform For Pure Decentralised Business Workflows"*. Submitted to journal of Web Intelligence and Agent Systems.

Conference Papers:

- L.Guo, Dave Robertson, Yun-Heh Chen-Burger, Jianquan Wang *"Conducting The Agile Negotiation Process Involved in The BPEL4WS Model on the Multi-agent Platform"*. CNAIS2005.

- L.Guo, Dave Robertson, Yun-Heh Chen-Burger *"A Novel Approach For Enacting Distributed Business Workflow on the Peer-to-Peer Platform"*. The proceedings of 2005 IEEE Conference on E-Business Engineering. (ICEBE 2005) (SCI and EI Indexed).

- L.Guo, Dave Robertson, Yun-Heh Chen-Burger *"A Generic Multi-agent System Platform For Business Workflows Using Web Services Composition"*. The proceedings of 2005 IEEE Intelligent Agent Technology (IEEE/WIC/ACM IAT-2005) (SCI and EI Indexed).

- L.Guo, Dave Robertson, Yun-Heh Chen-Burger *"Enacting the Distributed Business Workflows Using BPEL4WS on the Multi-Agent Platform"*. The proceedings of the MATES 2005 conference, volume number 3550 of (LNAI).

- L.Guo, Dave Roberston and Yun-Heh Chen-Burger *"Mapping a Business Process Model to a Web Services Model"*. Proceedings of the Third IEEE International Conference on Web Services, (ICWS 2004) (SCI and EI Indexed).

Workshop Papers:

- 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 , COLA 2004, Beijing, China, September 20-24, 2004.

Bibliography

[AJ00]   W.M.P Aalst and S Jablonski. Dealing with workflow change: Identi-
           fication of issues and solutions. In *International Journal of Computer
           Systems Science & Engineering*, volume 15(5), pages 267–276, 2000.

[AS96]   G. Alonso and H. Schek. Research issues in large workflow management
           systems. In *in Proc. of NFS Workshop on Workflow and Process Automa-
           tion in Information Systems*, pages 126–132, May 1996.

[ASHT98]   G. A. Bolcer A. S. Hitomi, P. J. Kammer and R. N. Taylor. Distributed
           workflow using http: Example using software pre-requirements. In *The
           20th International Conference on Software Engineering*, Appril 1998.

[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.

[AWS02]   Barbara Staudt Lerner Eric K. McCall Leon J. Osterweil Alexander Wise,
           Aaron G. Cass and Stanley M. Sutton. Peep-to-peer technologies and col-
           laborative work management: The implications of napster for document
           management. In *Workflow Handbook*, pages 81–94, 2002.

[BBN02]   Q. Z. Sheng B. Benatallah, M. Dumas and A. H. Ngu. Declarative compo-
           sition and peer-to-peer provisioning of dynamic web services. In *Proceed-
           ings of the 18th International Conference on Data Engineering (ICDE02)*,
           pages 297–308, 2002.

[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.

[BV04]   P. Buhler and J. M. Vidal. Enacting bpel4ws specified workflows with multiagent systems. In *Proceedings of Workshop on Web Services and Agent-Based Engineering*, 2004.

[CBL05]   Yun-Heh Chen-Burger and Fang-Pang Lin. A semantic-based workflow choreography for integrated sensing and processing. In *Proceedings of The 9th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA)*, May 2005.

[CBR98]   Yun-Heh Chen-Burger and Dave Robertson. Formal support for an informal business modeling method. In *Conference proceedings of The Tenth International Conference on Software Engineering and Knowledge Engineering*, June 1998.

[Coa99]   Workflow Management Coalition. Workflow management coalition terminology & glossary, Feb 1999.

[Coo02]   M. D. Coon. Peer-to-peer workflow management white paper. 2002.

[DGCIS95] Mark Hornick1 Diimitrios Georgakopoulos1 Contact Information and Amit Sheth2. An overview of workflow management: From process modeling to workflow automation infrastructure. In *Distributed and Parallel Databases*, pages 119–153, April 1995.

[DGS95]   M. Hornick D. Georgakopoulos and A. Sheth. An overview of workflow management: From process modelling to infrastructure for automation. In *Journal on Distributed and Parallel Database Systems*, pages 3(2):119–153, April 1995.

[EGD97]   R. Cingil E. N. Tatbul P. Koksal E. Gokkoca, M. Altinel and A. Dogac. Design and implementation of a distributed workflow enactment service. In *The 2nd IFCIS Conference on Cooperative Information Systems (CoopIS97)*, pages 89–98, June 1997.

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

[FCP96]   B. Pernici F. Casati, S. Ceri and G. Pozzi. Workflow evolution. In *15th International Conference on Conceptual Modeling (ER'96)*, pages 438–455, Oct 1996.

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

[Fis02]   L. Fischer. Lighthouse point, fla.: Future strategies. In *Workflow Handbook 2002*, 2002.

[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.

[GAM97]   A. El Abbadi G. Alonso, D. Agrawal and C. Mohan. Functionality and limitations of current workflow management systems. In *Research Report, IBM Almaden, Research Center*, 1997.

[Geo88]   M. P. Georgeff. Communication and interaction in multi-agent planning. In *Distributed Artificial Intelligence*, 1988.

[GPW03]   A. Finkelstein G. Piccinelli and S. L. Williams. Service-oriented workflow: The dysco framework. In *Proceedings of 29th Euromicro Conference (EUROMICRO03)*, pages 291–297, 2003.

[HA00]   C. Hagen and G. Alonso. Exception handling in workflow management systems. In *IEEE Transactions on Software Engineering*, volume 26(10), pages 943–958, Oct 2000.

[Hav01]   J. D. Havard. Interaction as a framework for flexible workflow modelling. In *The 2001 International ACM SIGGROUP Conference on Supporting Group Work*, pages 32–41, Sept-Oct 2001.

[JB96]   S. Jablonski and C. Bussler. Workflow management - modelling concepts, architecture and implementation. In *International Thomson Computer Press*, September 1996.

[JBR99]   M. z. Muhlen J. Becker, C.v. Uthmann and M. Rosemann. Identifying the workflow potential of business processes. In *in Proceedings of the 32nd Hawaii International conference on System Sciences*, Jan 1999.

[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.

[JWB96]   G. Vossen J. Wainer, M. Weske and C. Bauzer. Medeiros scientific workflow systems. In *The Proceeding of NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*. May 1996.

[JXT]   Jxta platform. Technical report. http://www.jxta.org/.

[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.

[JYR04]   Y. Yang J. Yan and G. K. Raikundali. Critical issues in extending p2p-based swindew p2p-based swindew system for incomplete process support. In *Proceeding of the 8th International Conference on Computer Supported Cooperative Work in Design (CSCWD04)*. May 2004.

[LL02]   K C. Laudon and J P. Laudon. *Management information systems*. London: Prentice Hall International, seventh edition, 2002.

[LMCM01] Hamideh Afsarmanesh Lui M. Camarinha-Matos. Virtual enterprise modeling and support infrastructures: applying multi-agent system approaches. In *Mutli-agents systems and applications*, pages 335–364, 2001.

[MM96]   H. Ledgard M. Marcotty. The world of programming languages. Springer-Verlag, 1996.

[Moh97]   C. Mohan. Recent trends in workflow management products, standards and research. In *in Proc. of NATO Advanced Study Institute (ASI) on Workflow Management Systems and Interoperability*, August 1997.

[Moh98]   C. Mohan.   Workflow management in the internet age, advances in databases and information systems.   In *2nd East-European Symposium on Advances in Databases and Information Systems (ADBIS'98)*, volume 1475, pages 26–34, Sept 1998.

[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.

[MS02]   P. Mangan and S. Sadiq.   On building workflow models for flexible processes. In *13th Australasian Database Conference (ADC'02)*, 2002.

[NH94]   L. Nastansky and W. Hilpert. The groupflow system: A scalable approach to workflow management between cooperation and automation. In *in Proceedings of the 24th Annual Conference of the German Computer Society during the 13th World Computer Congress (IFIP94)*, pages 473–479, September 1994.

[NOW05]   D. Robertson N. Osman and C. Walton.   Run-time model checking of interaction and deontic models for multi-agent systems. In *Proceedings of EUMAS*, December 2005.

[NRdW05] Roman P.J. van der Krogt Nico Roos, Cees Witteveen and Mathijs M. de Weerdt. Diagnosis of single and multi-agent plans. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

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

[OWL04]   Owl web ontology languagereference. Technical report, Feburary 2004.

[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.

[Sch99]   M. T. Schmidt. The evolution of workflow standards. In *IEEE Concurrency*, pages 44–52, July-Sept 1999.

[Sie99]   R. Siebert. An open architecture for adaptive workflow management systems. In *Transactions of the SDPS: Journal of Integrated Design and Process Science*, volume 3(3):29-41. Society for Design and Process Science, Sept 1999.

[SJS02]   C. Hahn S. Horn R. Lay J. Neeb S. Jablonski, R. Schamburger and M. Schlundt. A comprehensive investigation of distribution in the context of workflow management. In *in Proceedings of 8th International Conference on Parallel and Distributed Systems*, pages 187–192, Jone 2002.

[SJT97]   K. Stein S. Jablonski and M. Teschke. Experiences in workflow management for scientific computing. In *Proceeding of 8th International Workshop on Database and Expert Systems Application*. Sept 1997.

[SKL02]   P. Wagstrom S. Krishnan and G. Laszewski. Gsfl: A workflow framework for grid services. 2002.

[SLS99]   A. Goh S. Liu and E. Soong. State-based modelling of flexible workflow executions in distributed environments. In *ournal of Integrated Design and Process Science*, volume 3(2), pages 49–62. Austin: Society for Design and Process Science, 1999.

[SPJC97] E. Park S. Paul and RainMan J. Chaar. A workflow system for the internet. In *Internet, in Proc. of ACM SIGPLAN Conference On Object-Oriented Programming Systems, Languages and Applications (OOPSLA97) Workshop on Business Object Design and Implementation III*, Oct 1997.

[SSO01]   W. Sadiq S. Sadiq and M. Orlowska. Pockets of flexibility in workflow specifications. In *The 20th International Conference on Conceptual Modelling (ER'01)*, volume 2224, pages 513–526, Nov 2001.

[SV96]   M. Singh and M. A. Vouk. Scientific workflows: Scientific computing meets transactional workflow. In *Proceeding of NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, PART II-Reference Papers*. May 1996.

[UDD02]   http://uddi.org/pubs/programmersapi-v2.04-published-20020719.htm. Technical report, 2002.

[VAM01]   S. A. Chun V. Atluri and P. Mazzoleni. A chinese wall security model for decentralised workflow systems. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 48–57, 2001.

[vdAvH02]   W.M.P. van der Aalst and K.M. van Hee. Workflow management: Models, methods, and systems. In *MIT Press, Cambridge, MA*, 2002.

[Wal04a]   C. Walton. Model checking agent dialogues. In *2004 Workshop on Declarative Agent Languages and Technologies (DALT)*, July 2004.

[Wal04b]   C. D. Walton. Model checking multi-agent web services. In *Proceeding of AAAI Symposium of Semantic Web Services*, 2004.

[Wes98]   M. Weske. Interaction as a framework for flexible workflow modelling. In *Proceedings of 31st Hawaii International Conference on System Sciences*, 1998.

[Wes02]   M. Weske. A formal framework to support workflow adaptation. In *International Journal of Software Engineering and Knowledge Engineering*, volume 12(3), pages 245–268, June 2002.

[WSD01]   http://www.w3.org/tr/wsdl. Technical report, 2001.

[XML06a]   http://www.w3.org/tr/2006/rec-xml-20060816/. Technical report, 2006.

[XML06b]   http://www.w3.org/tr/xmlschema11-2/. Technical report, 2006.

[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.

[Yan02a]   Y. Yang. Enabling cost-effective light-weight disconnected workflow for web-based teamwork support,. In *Journal of Applied Systems Studies*, volume 3(2), 2002.

[Yan02b]   Y. Yang. Tool interfacing mechanisms for programming-forthe-large and programming-for-the-small. In *in Proceedings of the 9th Asia Pacific Software Engineering Conference (APSEC's 02)*, pages 359–365, Dec 2002.

[YY01]   Z Weiming Shen Yuhong Yan, Maamar. Integration of workflow and agent technology for business processmanagement. In *Predeedings of Computer Supported Cooperative Work in Design*, pages 420–426, July 2001.