Cross Organisational Compatible Workflows Generation and Execution

By

Mohammad Saleem

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University

March, 2012

Abstract

With the development of internet and electronics, the demand for electronic and online commerce has increased. This has, in turn, increased the demand for business process automation. Workflow has established itself as the technology used for business process automation. Since business organisations have to work in coordination with many other business organisations in order to succeed in business, the workflows of business organisations are expected to collaborate with those of other business organisations. Collaborating organisations can only proceed in business if they have compatible workflows. Therefore, there is a need for cross organisational workflow collaboration.

The dynamism and complexity of online and electronic business and high demand from the market leave the workflows prone to frequent changes. If a workflow changes, it has to be re-engineered as well as reconciled with the workflows of the collaborating organisations. To avoid the continuous re-engineering and reconciliation of workflows, and to reuse the existing units of work done, the focus has recently shifted from modeling workflows to automatic workflow generation.

Workflows must proceed to runtime execution, otherwise, the effort invested in the build time workflow modeling is wasted. Therefore, workflow management and collaboration systems must support workflow enactment and runtime workflow collaboration.

Although substantial research has been done in build-time workflow collaboration, automatic workflow generation, workflow enactment and runtime workflow collaboration, the integration of these highly inter-dependent aspects of workflow has not been considered in the literature. The research work presented in this thesis investigates the integration of these different aspects. The main focus of the research presented in this thesis is the creation of a framework that is able to generate multiple sets of compatible workflows for multiple collaborating organisations, from their OWLS process definitions and high level goals. The proposed framework also supports runtime enactment and runtime collaboration of the generated workflows.

To my parents and family

for their love, support and sacrifices

Acknowledgements

I am highly thankful to my supervisors Professor Paul Chung, Dr. Shaheen Fatima and Dr. Wei Dai for their guidance, support and encouragement throughout my PhD. I will always remain grateful to them for their continuous help in reviewing my writings and research to make it better and more valuable.

I would like to say special thanks to Loughborough University Innovative Manufacturing and Construction Research Centre (IMCRC), and Engineering and Physical Sciences Research Council (EPSRC), through which the project was funded.

I am indebted to my family for their un-ending love and support. I would also like to thank my friends Aamir Ehsan, Asmatullah, Fayaz Ahmad Khan, Irfan Khattak, Jawad Khattak, Muhammad Athar, Muhammad Shafi, Noor Khan, Riaz Muhammad, Sajjad Wali Khan and Zaresh Khan for their encouragement. I would like to say special thanks to the administration staff of Loughborough University for their help and support.

Table of Contents

List of Figures

List of Tables

Chapter 1: Overview

1.1 Introduction

Business process is the key element of work practice and a crucial part of corporate asset. A Business process can be defined as "*a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships*" [1]. In this age of electronics and internet, an increasing number of organisations are shifting to electronic commerce. The business processes of organisations must be automated to do business electronically. Hence business process automation is in demand more than ever. Workflow is the technology used to model automated business processes.

When two organisations do business together, the need for business process collaboration across multiple organisations arises. Such collaboration is referred as cross organisational collaboration. Since workflow technology is commonly used for business process automation hence there is a need for cross organisational workflow collaboration. As workflows must proceed into execution stage, the collaboration must be done at build time as well as runtime.

This thesis aims at introducing a framework for cross organisational workflow collaboration. The key difference between the proposed framework and existing systems is that it looks at cross organisational workflow collaboration in a very different context. Instead of reconciling existing workflows with each other, it automatically generates compatible workflows for the collaborating organisations from the OWLS process definitions and high level goals of the collaborating organisations, where OWLS is a language to describe web services [2]. So, it does not require the organisations to model their workflows beforehand and enables them to avoid the time consuming process of negotiations to reconcile existing incompatible workflows. The build time cross organisational workflow collaboration is done automatically at the time of workflow generation.

The proposed framework will also support runtime workflow collaboration among the generated workflows. The suggested framework will perform cross organisational workflow collaboration in a more time-effective and resource-effective way. The thesis also reports the prototype implementation for the framework.

This chapter gives an overview of the thesis. Section 1.2 gives the motivation for carrying out the research reported in this thesis. Section 1.3 sets the aims and objectives. Section 1.4 highlights the contributions and Section 1.5 presents the structure of the thesis.

1.2 Research Motivation

With the development of electronics and internet, more organisations are moving to electronic business to save time and resources. In the real world, organisations have to interact with other organisations to do business. For any two organisations to proceed in business, they should have compatible business processes [3]. Since workflow technology is used for representing automated business processes, this means any two organisations cannot proceed in business if they don't have compatible workflows [4]. Compatible means that there should be an agreed sequence of activities exchanging collaborative messages and information [4]. Incompatible workflows should be reconciled before proceeding with business. Considerable amount of effort is needed to ensure that workflows are compatible in the first place [5, 6].

Most of the existing work on workflow collaboration focuses on reconciling existing incompatible workflows. This is the bottom-up approach for collaboration since collaborative process is extracted from the existing local workflows [7]. A considerable amount of time and resources are required to reconcile incompatible workflows, especially if an organisation has to collaborate with many other organisations for business.

In an alternative top-down approach, organisations meet, discuss and design collaborative process and then implement it locally [7]. This is even more time consuming, especially if the organisations have many business partners to collaborate with. Every time an organisation has to collaborate with another organisation, both organisations will have to invest a lot of time and resources in the negotiations to come

up with designing compatible workflows. In case of any changes to the workflow of an organisation, negotiations should be done all over again with all the collaborating organisations, and workflows should be remodelled.

Although the automatic workflow collaboration based on bottom-up approach helps in terms of time and resources, the interacting organisations still have to get involved by giving feedback about the adjustment moves suggested by the automatic collaboration systems. In case of any change to the workflow of an organisation, not only the concerned organisation has to remodel its workflow but the workflows of the interacting organisations also need to be reconciled and adapted. This is a highly time consuming process because the dynamic and complex nature of businesses and high market demand leave business processes prone to changes on a very frequent basis.

To target the above issues we look at another paradigm in the literature known as automatic workflow generation. Automatic workflow generation is an AI planning problem in which a workflow is considered synonymous with a plan [8, 9]. If every activity in a workflow is treated as a web service, a workflow represents a plan of web services to achieve the desired goal state from a given initial state. So the workflow generation problem can be treated as a web services composition problem [9]. In web services composition problem a planner reasons about a pool of available services and a required service that can bring about the desirable effect is added in the plan; executing the plan results in the goal state [8]. Since each service is modelled as an OWLS process, the plan of web services essentially is a workflow of OWLS processes. Executing the workflow achieves the desired goals.

In literature, there has been extensive work on automatic workflow generation [10-17]. The general issue with the existing work is that it targets the automatic generation of workflows for a single organisation, and according to the author's best knowledge, no work has been done on the generation of compatible workflows for multiple collaborating organisations. In the literature, cross organisational collaboration and automatic generation have been treated mostly as separate aspects of workflow and there is very limited work done on the integration of these two related paradigms. Similarly the build time workflow collaboration and runtime workflow collaboration have been handled mostly by separate systems and very few systems support both build

time and runtime workflow collaboration. This is the research gap in the literature that this thesis aims to target.

1.3 Aim and Objectives

The aim of the research is "*to develop a framework that will create multiple sets of compatible workflows for multiple collaborating organisations, from the OWLS process definitions and high level goals of the collaborating organisations. The system will also provide support for runtime execution and collaboration of the generated compatible workflows*". To achieve the overall aim, the following objectives are addressed:

- To review AI planning paradigms and select the most suitable planner for automatic generation of compatible workflows.
- To introduce an integration approach that integrates automatic workflow generation, build-time cross organisational workflow collaboration, workflow enactment and runtime cross organisational workflow collaboration.
- To create a framework that can automatically generate compatible workflows for the collaborating organisations from the loaded OWLS process definitions. The workflows must be able to achieve the specified high level goals from the specified initial states.
- To create a proof-of-concept prototype which allows potential partners to load their OWLS process definitions and specify their initial states and final goals.
- To create a runtime execution mechanism that is able to enact and collaborate the generated workflows at runtime.
- To demonstrate the functionality, scalability and viability of the proposed framework with the help of different multi-organisational business collaboration examples.

1.4 Contributions

Following are the main contributions of the thesis.

- The thesis proposes an integration approach that is based on the integration of automatic workflow generation, build-time cross organisational workflow collaboration, workflow enactment and runtime cross organisational workflow collaboration. The integration approach applies AI planning to the integration of workflow generation and workflow collaboration.

- The thesis presents a framework which creates compatible workflows for multiple collaborating organisations, from the OWLS process definitions and high level goals of the interacting organisations. It is the only framework so far that targets the creation of compatible workflows for multiple collaborating organisations, without involving any reconciliation among the collaborating workflows or any negotiations among the collaborating organisations.

- The proposed framework handles both build-time and runtime workflow collaboration for arbitrary number of organisations. This is a powerful capability that is not common in literature.

- The thesis presents a run-time execution mechanism for the compatible workflows automatically generated for the collaborating organisations. The runtime execution mechanism makes sure that the transfer of data and information among the in-house and cross organisational activities takes place smoothly.

- A novel technique to increase the efficiency of workflow generation process is put forward. The system reasons about the usability of each atomic process in the workflow generation process. The OWLS processes that do not make part of the workflow generation process are not translated to SHOP2 format and they are simply discarded. It makes the translation process easier and time efficient. It can also increase the efficiency of the planning process in certain cases, since the planning engine only has to do planning based on the processes that are strictly used in the workflow generation.

- The proposed framework uses SHOP2 for planning. SHOP2 relies on a good domain model for planning. The OWLS process definitions of the collaborating organisations must be translated into SHOP2 format. Therefore, we develop a

novel algorithm for translating OWLS processes into SHOP2 domain. The translation algorithm makes it possible to create a SHOP2 domain which enables the planner to identify alternate composition paths and hence create multiple valid workflows. The translation algorithm makes it possible to compose atomic processes of the collaborating organisations into compatible workflows, in the correct order.

- SHOP2 needs a good formal domain description for planning. The efficiency and success of the planning process is highly dependent on the domain used for planning. The SHOP2 domain for workflow generation problem is in the form of web services descriptions and OWLS process definitions translated into SHOP2 format. Due to the very complex and diverse nature of web services, it is not always possible to create a clear and efficient formal domain for SHOP2. Such complex and inefficient domains can force SHOP2 into infinite loops. SHOP2 has been extended to enable it to counter such issues and make it more suitable for the workflow generation problem.

- A novel algorithm has been proposed to merge the domains of the collaborating organisations into a single joint domain. The joint domain can be considered as the domain of a single parent organisation, containing collaborating sub-organisations. Multiple joint plans can be created based on the joint domain. An algorithm to divide the joint plans into sub-plans is also proposed. Each sub-plan is basically a workflow for a single collaborating organisation.

1.5 Thesis Structure

The rest of this thesis is structured into the following chapters:

Chapter 2 explains key concepts and terms such as business process, workflow, workflow management systems, control flow, data flow and automatic workflow generation. Recent workflow generation frameworks are also reviewed in Chapter 2.

Chapter 3 reviews work done on business collaboration, workflow collaboration and workflow compatibility, and reviews different build time and runtime workflow collaboration frameworks and approaches.

Chapter 4 considers workflow generation as an AI planning problem, discusses several of the established planning paradigms, and reviews representative planners that fall into those planning paradigms against the requirements of web services composition.

Chapter 5 presents the proposed integration approach that integrates automatic workflow generation, build-time cross organisational workflow collaboration, workflow enactment and runtime cross organisational workflow collaboration. Requirements of the integration approach are highlighted.

Chapter 6 presents the framework for cross organisational compatible workflows generation and execution, which is based on the integration approach. It explains the methodology, design architecture and functionality of the framework. Algorithms involved in the framework, and the technical and implementation details of the prototype developed for the framework are also explained in Chapter 6.

Chapter 7 reports and evaluates the results based on various multi-organisational business collaboration examples. The chapter also discusses the scalability, complexity and viability of the framework.

Chapter 8 concludes the thesis. The chapter gives the summary and main conclusions of the thesis, summarises the novelties and contributions of the thesis and outlines the future work.

Chapter 2: Workflow

2.1 Introduction

Business processes are at the core of productivity for organisations. It refers to a set of connected and ordered activities to achieve a business goal within the context of an organisational structure [1]. For an organisation to do online business with other organisations, its business processes need to be automated, so that the organisations can automatically exchange products and services more efficiently and flexibly. The information and communication technology enabled automation of the business processes create automated business processes. Automated business processes are business processes, modelled and executed with the help of information and communication technologies. With the development of the internet, there has been an increase in electronic and online commerce. Hence the demand for business process automation has increased. Workflow is the technology used for business process automation and it has proved to be a mature and beneficial technology [18]. It has been widely adopted by business organisations across the globe.

The organisations that comprise an online business may be heterogeneous in nature, i.e., they may have their own set of constraints, requirements, standards and goals. Also, online businesses may be dynamic, since the organisations and their business processes may evolve continuously. This heterogeneous and dynamic nature of the online business leaves the organisational workflows prone to frequent changes. This increases the demand for automatic workflow generation, so as to save the effort invested in modelling and adapting workflows. Being prone to frequent changes, business organisations also require reusability of already developed processes, resources and software components. To automatically generate workflows and incorporate reusability in workflows, the idea of web services and web services composition comes into context. Web services are self-contained units of application logic, providing business functionality over the internet [19]. They can be discovered, connected to and executed over the internet. They can also be automatically composed in a proper order to support a workflow [9]. This chapter reasons that automatic workflow generation and web services composition are the same problem, if each activity in the workflow is

represented by a web service. Due to this similarity, the research presented in the thesis uses the idea of web services composition for the automatic generation of workflows for multiple organisations. In the generated workflows, each activity is a web service.

This chapter discusses business process automation, workflow, issues with traditional ways of modelling workflows and some new developments related to workflow. Section 2.2 highlights business process automation, Section 2.3 explains workflow, Section 2.4 describes workflow management systems, Section 2.5 gives description for automatic workflow generation and its similarity to web services composition, Section 2.6 reviews selected approaches and systems for automatic workflow generation and Section 2.7 concludes this chapter.

2.2 Business Process Automation

Business process automation refers to technology enabled analysis, documentation, optimisation, modelling and enactment of activities and services to achieve certain functions. Business process automation reduces human intervention, increases cost saving, enhances the accuracy of the information passed to the business processes and among the business activities, and ensures the reuse of value added tasks. It saves time by programing the repeated routine tasks and makes it easy to track the progress status by automatically creating charts and graphs. It minimizes human error by removing the repeated data entry requirements and automatically making accurate calculations. It ensures the reuse of existing software components and shortens the billing and production cycles.

Workflow is a technology used for business process automation. The workflow is a product of evolution through different stages of business processes automation [20]. Initially, humans carried out all processes, manipulating physical objects. These are characterised as material processes [21], since everything in these processes is rooted in the physical world i.e. storing, moving, measuring, assembling etc. With the development of information technology, the workplace started getting automated by computer programs. Thus information processes came into existence in which certain tasks were fully or partially automated by computer systems to automate the creation, processing and management of information [21]. Later, business processes were introduced to increase the efficiency of the business by capturing it in terms of market

centred descriptions, implemented as material processes or information processes. Once the organisations were able to capture their businesses in terms of business processes, the need for their automation and adaptation arose. Workflow is a concept closely related to automating and reengineering business processes. In the next section, we discuss workflow.

2.3 Workflow

According to Workflow Management Coalition (WfMC)'s definition, a workflow[1] is "*the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" [1]. A workflow has the following two main stages [1]:

- A build time stage, which refers to the stage where workflow descriptions of the business process are defined and changed. This can be automatic or manual.
- A runtime stage where instances of the business process are created and managed. This is the operational stage.

A workflow can be activity based or entity based [22]. An activity-based workflow is based on a set of activities that someone or something has to do [22]. Activity is one of the basic building blocks in mainstream process definition languages like WSFL, XPDL and BPML [23]. An activity is defined as "*a description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resource(s) to support process execution; where human resource is required an activity is allocated to a workflow participant.*" [1].

Figure 2.1 shows an example of an activity based workflow of a manufacturer who exports his goods to overseas market. Vendor is an overseas exporter. The Vendor waits for advance payment from a customer, and starts the manufacturing process after getting advance payment. After manufacturing goods the vendor issues a commercial

[1] Although, by definition, workflow has a more technical orientation and business process is more business oriented, the terms workflow, workflow process and business process are used interchangeably in this thesis.

invoice represented as *Invoice*, carries out factory inspection as an in-house procedure and produces an inspection certificate represented as *InspCert* and sends it to the customer. It waits for the customer's request for making shipment arrangement and after getting the request it makes shipment and insurance arrangement. When the shipment and insurance arrangement is done, the vendor sends the generated insurance certificate and commercial invoice to the customer. This is just a hypothetical workflow to explain the idea of activity workflow. Every logical step in the workflow is an activity e.g. *Goods Manufacture* or *Factory Inspection*. The links between the activities shown by the solid lines represent the sequential order between the activities. The dotted lines represent the data dependencies between the activities.

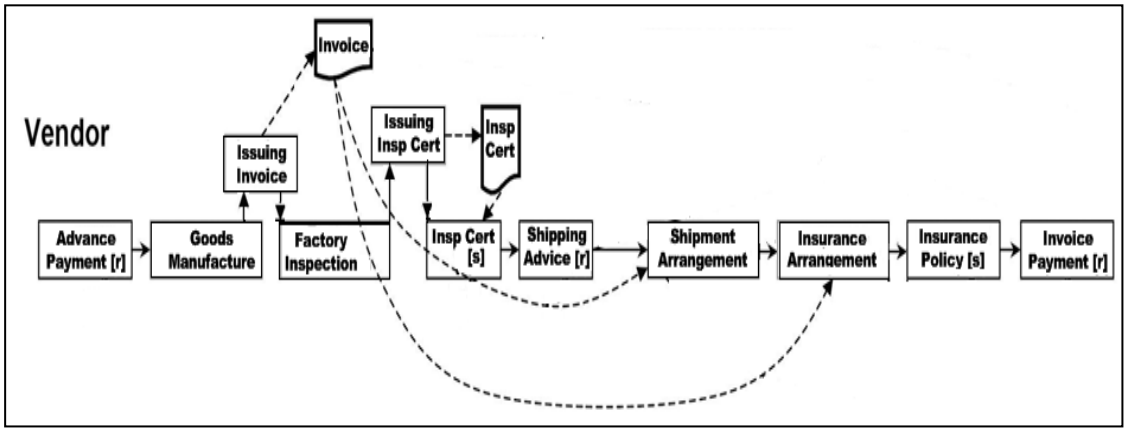

Figure 2.1 Activity based workflow of a vendor (taken from [24])

While activity based workflow is based on events (activities) and deals with the transition and data transfer among the events; the entity-based workflow is centred on a single entity, for instance a document which has an associated state and a set of possible transitions to new states, provided certain conditions are met [22]. In an activity based workflow, all the activities are explicit while in entity based workflow the activities are implicit to the states and based on role/permission mapping. In a given state, authorized actors can do certain actions on the entity. An actor is a user or a group of users in an organisation. Once an actor is satisfied, he can choose one of the available transitions to a new state.

Figure 2.2 shows an example of an entity based workflow of passing a legislative bill. In the private state only the creator of the bill can view it, in the public state the bill is presented to the public for voting, in the voting state the legislators can vote and in the

final review state the executive can review the bill. The bill is either vetoed or ratified by the executive. If the bill is rejected by public or legislative assembly, a new bill has to be created.

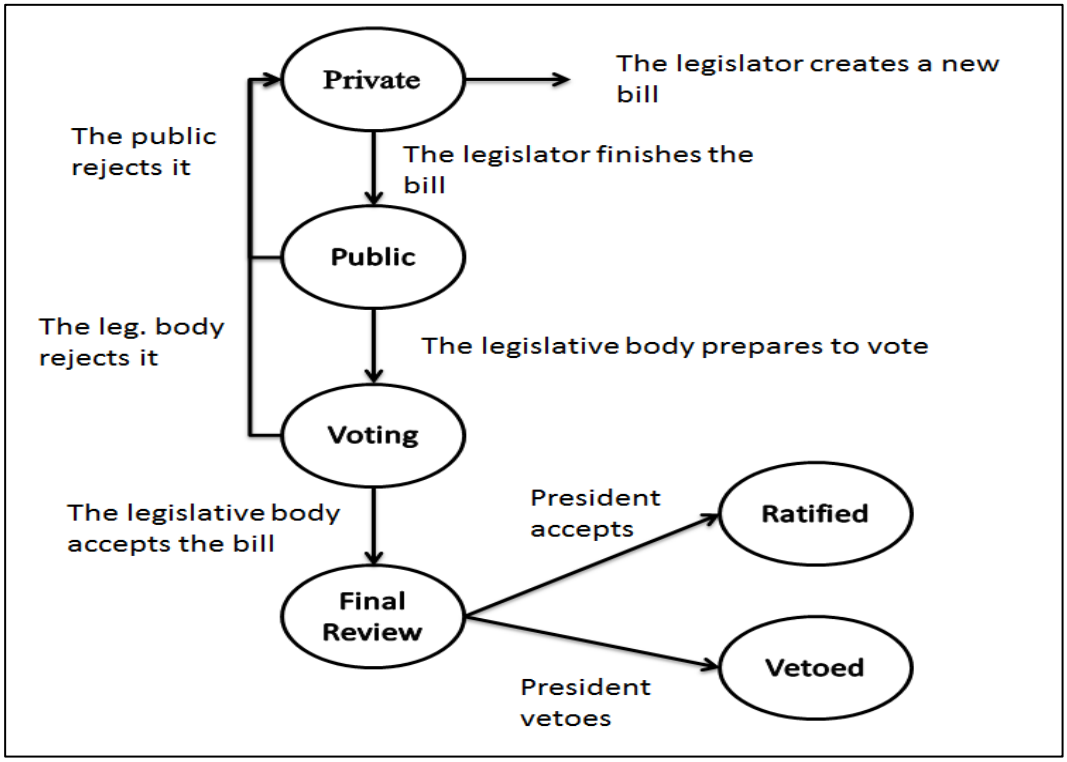

Figure 2.2. An entity based workflow for passing a legislative bill (taken from [22])

The proposed research uses activity based workflow model because it is the most widely used model adopted by most commercial and open source products like IBM's WebSphere MQ Workflow and Enhydra's Shark [24]. Entity based workflow has only been discussed for completion purposes and will not be considered further in the thesis.

A workflow[2] definition is a series of activities linked by control flow, on which data flow rests. Control flow is specified as the transitions between the activities [25]. To have correct control dependencies between activities in workflows, both the sequential order and data dependencies should be considered. At run time, the workflow engine instantiates and triggers activities following sequence specified in control flow. Data flow is maintained by the Workflow management system (WfMS) in the form of transferring workflow relevant data between activities. The data is accessible to

[2] The thesis uses workflow to refer to activity based workflow in the rest of the thesis.

applications and exchangeable between the WfMS and applications. In figure 2.1 the vendor should receive advance payment before manufacturing goods, so *Advance Payment[r]* must be performed before *Goods Manufacture*. This control dependency is specified in the workflow as a sequential order. Similarly *Insurance Arrangement* has a data dependency on *Issuing Invoice* since it needs *Invoice* as input. Therefore, the activity *Issuing Invoice* must be performed before *Insurance Arrangement*. Similarly in Figure 2.2, the legislative body cannot vote for the bill, until it has been passed by the public. And the passed bill should be passed from the *Public* state to the *Voting* state in order to vote over the bill. So there is a control and data dependency between *Public* and *Voting* states.

WfMS is needed to define, create and manage activity based workflows [1]. It enables us to take full advantage from the workflow technology. The following section discusses WfMS; gives the functional areas in which a WfMS provides support and presents its key features.

2.4 Workflow Management Systems

A workflow management system is "*a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications*" [1]. A WfMS provides support in the following three functional areas [26].

- At build time to define and model the workflow process and its constituent activities.
- At run time to support the modelled workflows in an operational environment and take care of the data and control flow.
- At run time to interact with user or external IT applications.

The key features of WfMS are summarised in the Figure 2.3.

A WfMS can be classified as production WfMS, administrative WfMS, collaborative WfMS or adhoc WfMS [27]. A production WfMS is used for automating complex and repetitive activities. They are mainly used for processing large number of similar tasks

to improve productivity. An administrative WfMS is used to automate administrative tasks where flexibility and human interaction are more important than productivity. A collaborative WfMS is used for the automation of business-critical processes where teams of different size communicate either directly or over the internet. Frequent changes occur to the business process in collaborative teams. Collaborative teams are collaborating and contributing teams of different sizes, communicating with each other either directly or over the internet. The ability to handle change requests and communicate effectively, are the success factors for a collaborative WfMS. An Adhoc WfMS is used for automating business processes that are based on unstructured information. Such processes should be created quickly and modified on the fly to adapt to new situations. An adhoc WfMS supports business processes for the routine work and its major success factors are flexibility and adaptability. Most widely used WfMS products are given in Appendix B.

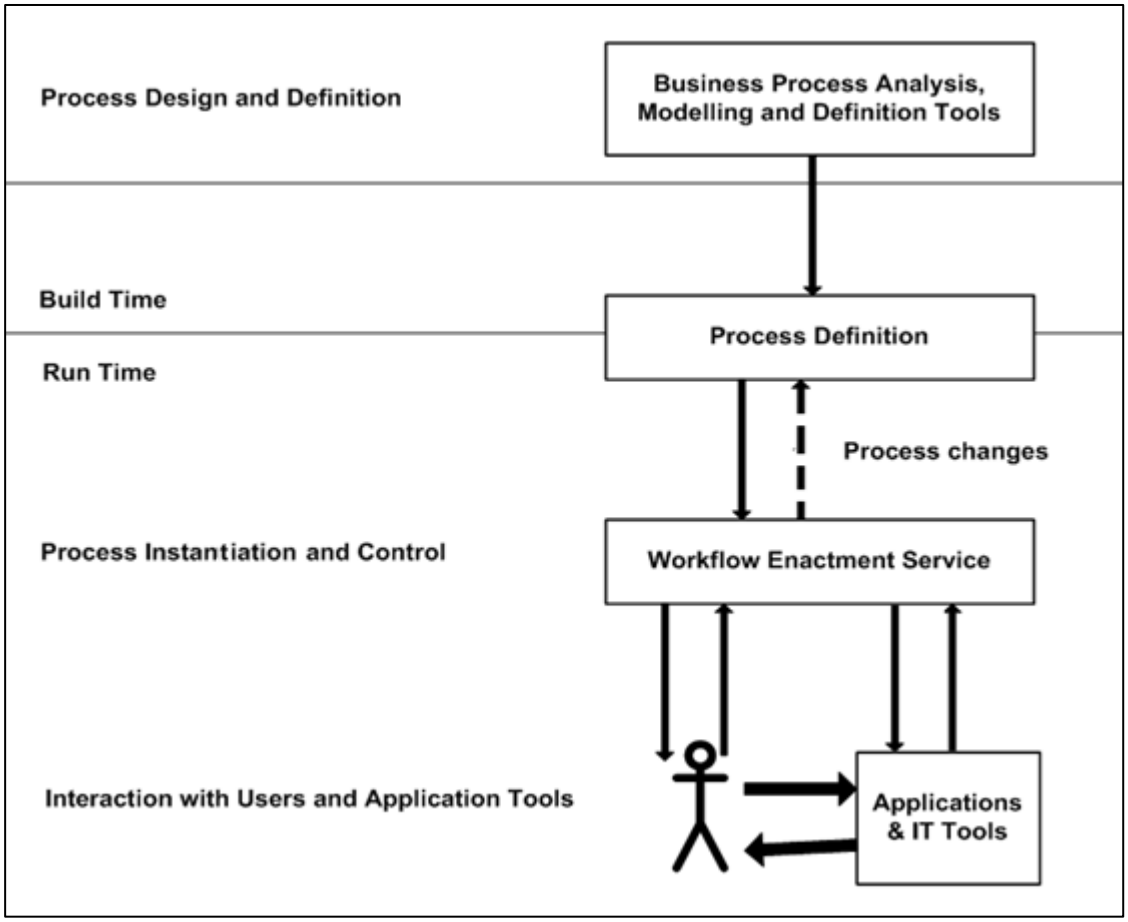

Figure 2.3. Key Features of WfMS (adapted from [26])

Workflow Management systems make the definition, creation and execution of workflows efficient and easy, and thus it helps in saving time and resources. But with the introduction of web services, there has been a higher demand for reusability. Also, due to the very heterogeneous and dynamic nature of business on internet, the chances of changes to the workflows of business organisations are very high. Therefore, the organisation will have to remodel and adapt their workflows very frequently. Workflow modelling and workflow adaptability is a highly time and resource consuming task. An organisation whose workflow is changed will have to model new activities, fix the control and data dependencies in the workflow and make sure that the workflow remains acceptable to the collaborating organisations as well. Therefore, in recent research, the focus has shifted from statically modelling workflows to automatic workflow generation. Automatic workflow generation enhances reusability by using web services. It enables the business organisations to save the time and resources spent on modelling and adapting workflows by generating workflows automatically from the process definitions of the organisations. The next section discusses automatic workflow generation in detail.

2.5 Automatic Workflow Generation

Automatic workflow generation refers to the creation of workflows from high level goals and OWLS process definitions of organisations. Automatic workflow generation is an AI planning problem [8]. AI Planning is a problem solving technique where an agent identifies a solution from an abstract set of possible plans, based on its knowledge about available actions and their results [28]. An agent is a computer system that can work without direct human intervention, interact with other agents, perceive environment and take initiative [29].

Web services composition enables the automatic generation of workflows. Since AI planning identifies an execution plan that reaches the goal state from initial state, given goal and initial states; web services can be orderly organized in a workflow to support the execution plan. If each web service is treated as an activity of the workflow, the generated plan can be treated as a workflow and the web service composition problem can be treated as automatic workflow generation problem [9].

In web service composition, the planning system requires formal domain ontology for planning. Domain ontology refers to the formal representation of the environment where the planning takes place, the operators that operate in the environment and all states that can change in response to an operator's action [8]. It models a specific domain and represents the meaning of terms as they apply to that domain. A web service composition environment is primarily a collection of web services, so the ontology is in the form of web services descriptions.

OWLS is a language for describing web services [2]. It is used to describe the functionality, access point and execution mechanism of web services. In OWLS, each service is modelled as a process [13]. A process can be *atomic*, *simple* or *composite*. *Atomic* process represents a single-step directly executable web service; *simple* process is an abstraction of an *atomic* process or a *composite* process; *composite* process represents a compound web service which can be decomposed into atomic web services. OWLS is a set of ontologies and OWL-S process ontology describes web services composition based on 'action' or 'process' metaphor. It describes simple tasks as simple actions or simple processes and complex tasks as composite actions or composite processes. This similar way of modelling makes it possible to translate OWL-S web services descriptions to AI planning domains [13].

Both web services composition[3] and automatic workflow generation are AI planning problems [8, 9]. If S is the set of all possible states in the world, $S_o \subset S$ is the set of initial states, $G \subset S$ is the set of goal states, A is the set of actions the planning system can perform and $\Gamma \subseteq S \times A \times S$ is the translation relation defining the preconditions and post-conditions of the actions; AI planning can be described as a five-tuple problem $\langle S, S_o, G, A, \Gamma \rangle$ [30]. In terms of web services, S_o is the set of initial states and G is the set of goal states specified by the web services composition requestor, A is the set of all available services and Γ is the set of all changes brought about by the operations of all services [30]. This means that for a set of available services, initial states, goal states and set of preconditions and effects of the available services; AI planning can be

[3] In this thesis automatic workflow generation and web services composition are used interchangeably, assuming that every activity in the workflow is a web service (modelled by an OWLS process).

applied to web services composition problem [8, 9, 13, 31, 32]. A theorem has been presented and proved in [13, 31, 32] to formally state the web services composition as an AI planning problem. If TRANSLATE-PROCESS-MODEL(*K*) represents the method to translate a set *K* of OWLS processes into SHOP2 domain *D*, the theorem is as follows [13, 31, 32].

> *"Let K = {K1,K2, . . . ,Km} be a collection of OWL-S process models, C be a possibly composite process defined in K, S_0 be the set of initial states, and P = (p_1, p_2, . . . , p_n) be a sequence of atomic processes defined in K. Let D = TRANSLATE-PROCESS-MODEL(K). Then P is a composition for C with respect to K in S_0 iff P is a plan for planning problem (S_0, M_C, D) where M_C is the SHOP translation for process C."*

In this thesis, both web service composition and automatic workflow generation refer to arranging web services modelled by OWLS processes in the correct order to create a workflow of web services to achieve a desired goal. Given a high level description of a goal, a planner will reason about all available services in terms of their capabilities; a required service that can achieve the desirable state will be identified and added into a workflow; executing the workflow will result in the state specified by the goal [8].

2.6 Existing Approaches and Systems for Workflow Generation

The following text gives a review of some of the automatic workflow generation approaches and systems.

In 2003, Sirin *et al.* [14]created a semi-automatic web services composition system. Their system enables users to select from a list of web services at each step of composition. The user starts the composition process. He/she selects one of the services registered to the system and the system then checks for web services that can satisfy inputs of the selected service. The system presents the web services that can satisfy the inputs of the selected web service to the user and the user selects one of them to add in the plan. The system then checks again for web services that satisfy inputs of the most recently added web service. The process continues until the composition completes. The system filters the web services based on the constraints specified by the user on the

attributes of the web services. This system is not a fully automatic system and the user has to get involved at every step.

In 2004, Sirin *et al.* [13] extended their semi-automatic web service composition system to a fully automatic system. They implemented an OWLS-SHOP2 translator to translate collections of OWLS process definitions into SHOP2 domain. They provided a sound and complete algorithm for the translation. The translation mechanism translates atomic processes into operators and complex processes into methods. The SHOP2 planner then uses the created domain to produce a valid plan according to the constraints entered by the user and imposed by the relevant web services. The user can accept or reject a plan and user can also re-plan with a new set of constraints. The generated SHOP2 plan is converted to OWLS format by SHOP2toOWL plan converter, and executed by the Execution System.

The system presented in [13] has been used as a foundation for the research presented in this thesis. Although this system is able to automatically generate workflows from the process definitions of an organisation, it aims to plan for a single organisation only and does not aim to support collaboration among multiple interacting organisations. This means the system cannot generate compatible workflows for multiple organisations collaborating together. In a multi-organisational collaboration scenario, the system will create workflows for each organisation individually and then the collaborating organisation will have to reconcile the workflows among themselves, which requires time and resources.

Transplan [17] is another web services composition system which creates plans from OWLS process definitions to achieve high level goals given by users. Transplan uses SHOP2 for planning. Transplan uses the translation algorithm presented by Sirin *et al.* [13] to translate OWLS process definitions to SHOP2 domain descriptions. Transplan can create all possible plans in most of the cases. Transplan adds the preconditions of each operator into the respective delete list of the operator, which is not always true in the real world domains. Transplan also does not support collaboration between multiple organisations. Transplan is not a complete system and it can fail to create a plan in certain scenarios, even though creating a valid plan is possible. One possible scenario is when a plan has two or more branches. If trying the first branch does not lead to a valid

solution, Transplan does not try the second branch even though it might lead to a valid solution.

Wang *et al.* [33] also presented a system for inter-organisational workflow coordination and dynamic workflow composition. They extended dynamic flexibility to the workflow runtime execution stage. Initially dynamic flexibility was only targeted at the design stage. They used intelligent agents for dynamically composing workflows and negotiating web services over the net. Intelligent agents are autonomous problem solving entities that take the state of their environment as input and act on the environment to fulfil certain role [34]. They used intelligent agents to discover, execute and monitor web services. Wang *et al.* used workflow ontology for representing tasks and their relationships. They used an ontology reasoning tool to do the coordination planning, maintain the ontologies and do the decomposition from upper level abstract ontologies into the lower level service ontologies. They targeted inter organisational coordination from the perspective of a single organisation which deals in a heterogeneous environment with adhoc external processes. Wang *et al.* did not take the creation of compatible workflows for multiple organisations into account. The complexity and scalability of the system is not fully known. Also, human operations are needed in complex scenarios and incomplete workflows.

Casati *et al.* [12] presented a platform named EFlow to specify, enact and manage composite services. A graph represents the order of tasks in the composite process. The creation of the graph is static, but it can be modified dynamically. The graph is made up of service, event and decision nodes. Service nodes represent atomic or composite processes, decision nodes represent execution flow and event nodes represent sending and receiving several types of events. The dependency between the nodes is represented by arcs in the graph. Each time a service is activated, a search recipe in the service node is executed. The search recipe returns a reference to the specific service, which it binds to. This is done because in the dynamic internet environment the availability of services may change any time. Casati *et al.* [35] developed a composite service definition language (CSDL) to further enhance the service composition platform by introducing various dynamic and adaptive features. The language has a unique capability of differentiating between invocation of a service and operation with in a service. Service nodes in CSDL specify the aspects of business logic specific to the services i.e. the

search recipe and service invocation. Method nodes represent methods to be invoked on services, their input data, output data and the way the output should be handled. Thus every node only holds its relevant logic and does not have access to the business logic that is irrelevant to it.

SWORD [16] is a developer toolkit which uses a rule based plan generation for building composite web services, and Entity-Relation (ER) model to specify the web services. Model is a world that represents entities and their relationships. Entities are modelled by Horn rules which specify that the post-conditions are achieved if the pre-conditions are true. The service-requester specifies the initial and final states for the composite service, and the rule-based expert system creates a plan that achieves the final states given the initial conditions are true. Ponnekanti and Fox [16] argue that to produce "certain" results a precondition should uniquely determine a post-condition. This means that the post-conditions should be functionally dependent on the pre-conditions to avoid uncertain results. This might be the case with the vast majority of web services composition systems [30].

McIlraith *et al.* [10, 11] used a modified version of ConGolog for web services composition. ConGolog is a variant of Golog Language, having the ability to support concurrency. Golog is a language built on the top of situation calculus providing extra logical constructs. In this approach, an atomic service is considered as a primitive action which changes the state of the world or knowledge of the agent. A composite service is considered as a complex action which is composition of primitive actions. An agent reasons about web services to discover, execute, compose and inter-operate web services. The users can introduce their own constraints and sequence choices. The agents use procedural programming language concepts combined with concepts designed for web services, to represent composition of web services.

The common problem with all the above systems is that all the above reviewed systems target the automatic generation of workflows for single organisations only. Their ultimate aim is to make the generation and execution of workflows time efficient and resource efficient in the context of a single organisation. They do not target the generation of compatible and interoperable workflows for multiple collaborating organisations. The only framework so far which targets the generation of compatible workflows for multiple collaborating organisations is presented by Saleem *et al.* [36].

They presented a framework that uses intelligent agents to create compatible workflows for multiple interacting organisations. The presented framework uses SHOP2 for planning. The framework translates the OWLS process definitions of each interacting organisation into SHOP2 domain description using the translation algorithm presented by Sirin *et al.* [13]. An instance of intelligent agent is created for every organisation, which works on behalf of the respective organisation. Each agent has its own instance of SHOP2. Intelligent agents collaborate with each other, before adding new steps in the workflows of the interacting organisations. Any step that makes the workflow incompatible with the workflows of interacting organisations is discarded by the agents, and a new step is tried. An algorithm for checking the compatibility of the interacting workflows is also presented. The problem with this framework is that it does not present any proof-of-concept prototype, nor is there any results and evaluation available. Only architecture of the framework has been presented. It is not known whether this is a feasible and practical approach, since the coordination among agents before adding every step will lead to very intensive communication among the intelligent agents.

2.7 Conclusion

In real business environment, with ever increasing demand for business process automation, business process of one organisation is likely to interact with business processes of other organisations. Therefore, there is a need for automatic cross organisational business process collaboration; otherwise, business process automation will be confined within the boundaries of individual organisations. Since workflow technology is used for business process automation hence there is a need for cross organisational workflow collaboration. Based on the need for cross organisational workflow collaboration, Chapter 3 focuses on cross organisational workflow collaboration and reviews ways for bringing about collaboration among interacting workflows.

The existing automatic workflow generation systems automatically generate workflows for single organisations only and do not cover the generation of compatible workflows for multiple organisations. This dilutes the benefit of automatic workflow generation, since after automatic generation of workflows; the collaborating organisations have to

make sure that their workflows are compatible with each other. Hence, there is a need for an automatic system, which can enable the generation of compatible workflows for multiple collaborating organisations. This thesis proposes such a system.

Chapter 3: Workflow Collaboration

3.1 Introduction

Business organisations interact with other organisations in order to achieve their goals. With the increase in demand for business process automation, organisations are faced with a new kind of business collaboration. The automated business processes of organisations need to collaborate with one another to achieve their goals. For any two organisations to proceed in business, their workflows should be acceptable to each other [3]. If the workflows are not acceptable to each other then they need to be reconciled at build time so that at runtime the workflows can collaborate with each other to ensure that the exchange of information and files between the workflows occur smoothly, and the execution of the workflows go hand in hand to completion. The build time and runtime collaboration between the workflows of multiples organisations is called cross organisational workflow collaboration. Cross organisational workflow collaboration is an active research area and a lot of work has been done in this field recently.

Section 3.2 discusses cross organisational business collaboration and workflow collaboration, Section 3.3 presents a detailed example of business collaboration, Section 3.4 outlines different definitions and concepts related to workflow compatibility, Section 3.5 reviews existing work on build time and runtime workflow collaboration and Section 3.6 concludes the chapter.

3.2 Cross Organisational Business Process/Workflow Collaboration

Business collaboration refers to multiple enterprises working together to achieve a business goal [37]. Such a goal can be a short term opportunistic goal or a long run strategic goal. A common collaboration scenario in the business world is the vendor/customer interactions with an opportunistic goal in pursuit of cashing in on a deal [38]. Since business practices are carried out in the form of workflows hence there is a great potential for cross organisational workflow collaboration.

Collaboration has to be carried out in two stages, i.e. build time and runtime. At build time business processes of business partners are checked for compatibility. In cases

where there are conflicts they are negotiated and reconciled [39]. Manual collaboration can be very time consuming. In addition, an organisation could be collaborating with many different organisations and negotiating workflow compatibility with many different can be very challenging. Therefore automatic support services are needed for collaboration.

For any two business partners to proceed in conducting B2B e-commerce transactions, their workflows involved in the transactions must be compatible with each other at the business level [3] i.e. they have a commonly agreed sequence of exchanging collaborative messages and information. Collaborative messages can be a business object like a purchase order or a service invocation request. The point where exchange of collaborative messages and information takes place between two interacting workflows is called interface activity [4, 40]. The set of all interface activities in a workflow is called interface process. Section 3.4 defines workflow compatibility and discusses its relevant concepts in further detail.

Collaborative systems are physically distributed and logically decentralised [41]. At runtime the task is to deliver all locally created messages correctly to business partners. Individual processes should be enacted as if they are running in an organisation's own environment and hence current workflow technology should be used as much as possible. Technical collaboration details need to be transparent to the WfMS's enactment service. So the runtime requirement is to establish a collaboration service by providing a proper communication channel to fulfil the message exchange task while taking into account loosely coupled collaborative structure, operational autonomy and minimum efforts for adoption alongside current WfMS's [39].

The next section presents a detailed example of workflow collaboration.

3.3 Workflow Collaboration Example

A common form of workflow collaboration is the Vendor/Customer scenario. Figure 3.1 shows the workflows of a vendor and a customer collaborating with each other, and also models their interface activities. Figure 3.2 shows the extracted interface processes of the vendor and customer. This example has been taken from [24] with slight modifications.

The vendor in this example is an overseas exporter. The vendor waits for the advance payment from a customer, checks the received payment and then starts the manufacturing process. After manufacturing the goods the vendor issues a commercial invoice represented as *Inv*, carries out factory inspection as an in-house procedure and produces an inspection certificate and sends it to the customer. The inspection certificate is represented as *InspCert*. It waits for the customer's request for making shipment arrangement. After getting the request it sends the commercial invoice to the customer and makes shipment and insurance arrangement. When the arrangement is done the vendor sends the insurance certificate and bill of lading to the customer, and applies for a certificate of origin to the local authority. The bill of lading is represented by *BL*. The vendor then sends the certificate of origin to the customer. The vendor waits for the payment for the invoice and the process completes after handling the payment.

The customer is an overseas importer. Customer sends advance payment to the vendor and waits for the inspection certificate which is a proof of quality of the goods. It reviews the inspection certificate and if satisfied then it produces and sends shipment arrangement request to the vendor. The request is represented by *SA*. After receiving the commercial invoice, bill of lading and insurance certificate, the customer takes delivery of the goods, carries out a presale inspection and waits for the certificate of origin. The customer needs the commercial invoice and bill of lading to get goods from the shipping company. Certificate of origin is required to get an import permit from the local authority. After receiving the certificate of origin the customer approves payment and sends full payment for the invoice to the customer.

In Figure 3.1 and 3.2, the interface activities of the vendor are labelled as *a_r*, *b_s*, *c_r*, *d_s*, *e_s*, *f_s*, *g_s* and *h_r*. The interface activities of the customer are labelled as *a_s*, *b_r*, *c_s*, *d_r*,*e_r*, *f_r*, *g_r* and *h_s*. An activity name followed by "*_s*" or "*_r*" means it is a sending or receiving activity respectively in the collaboration process. These are the points where exchange of collaboration message, information and files takes place between the vendor and customer.

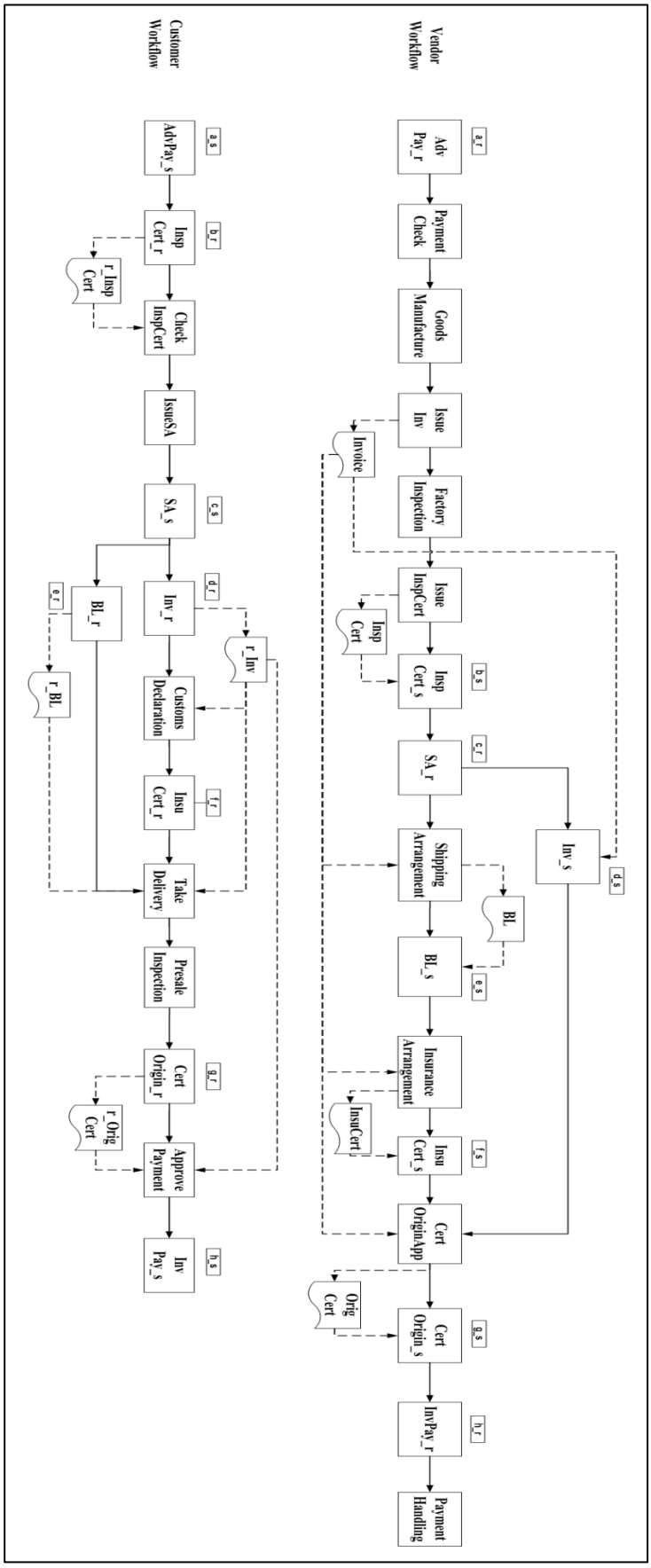

Figure 3.1 Workflow of Collaborating Vendor and Customer (adapted from [24])

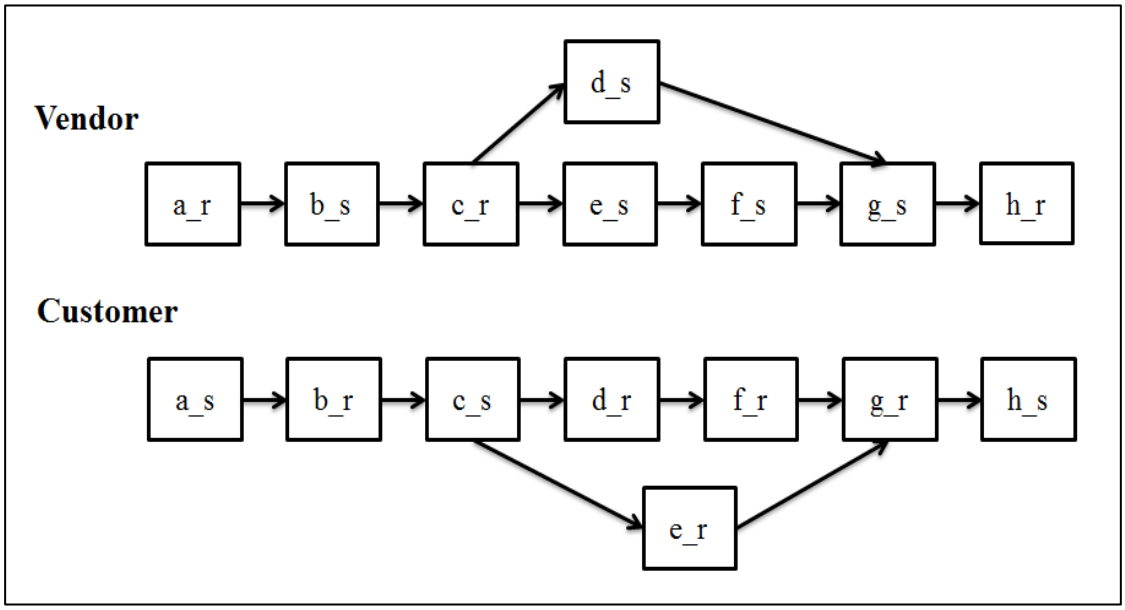

Figure 3.2 Interface Processes for Vendor and Customer

3.4 Workflow Compatibility

Any two workflows are compatible if they have a commonly agreed sequence of exchanging collaborative messages and information [3]. There can be various criteria for compatibility. Some of them are strict while others are relaxed. [4] defines two forms of compatibility.

Definition 1 Absolute Compatibility: Two collaborative workflows are absolutely compatible if:

- Interaction points are modelled as interface activities,
- All interface activities for the two workflows are paired,
- Message delivery takes place through a communication channel,
- The sequence of interface activities of both workflows is exactly the same.

This is a very strict criterion and there is a huge cost attached to achieve this. A less strict criterion is enactable compatibility [4].

Definition 2 Enact-able Compatibility: Two collaborative workflows are enactable compatible if:

- The first 3 conditions of absolute compatibility are met, and
- The sequence that interface activities occur in both workflows does not cause a deadlock in message exchange.

As enactable compatibility is too relaxed, so its effectiveness is not always guaranteed. For example, consider the interface processes *A* and *B* as shown in Figure 3.3. Although both processes are enactable compatible, *A.g_r* has to wait for the message *g* until activity *B.g_s* is completed. The wait could be considerable which might not be acceptable to *A*.

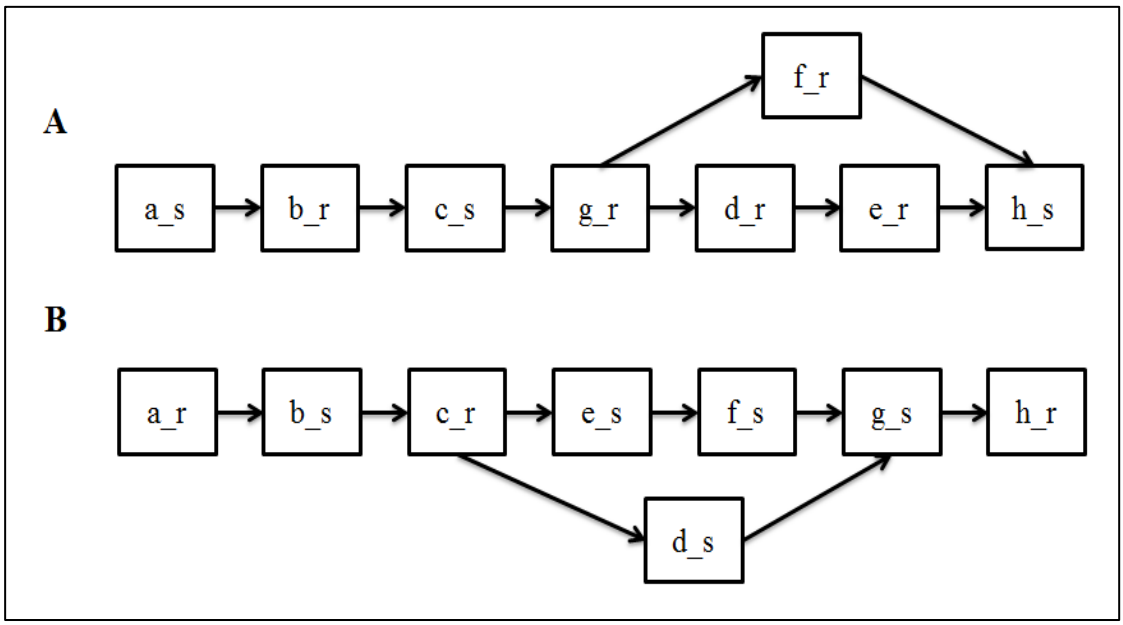

Figure 3.3 Enactable compatibility might cause unnecessary delay (taken from [4])

[42] defines another type of compatibility.

Definition 3 Business Collaborative Compatibility: Two collaborative workflows are of business collaborative compatible if:

- The first 3 conditions of Absolute Compatibility are met and,
- The workflows have at least one common path, i.e. one common sequence of corresponding interface activities.

Business collaborative compatibility leads to a more efficient collaboration as compared to enactable compatibility, since enactable compatibility allows collaboration whenever

there is no deadlock in the sequence of interface activities, while business collaborative compatibility allows collaboration only when there is an agreed and similar sequence of interface activities after traversing all possible sequences. The common trace guarantees that successful collaboration can take place as corresponding activities on a common trace will always be reached by both interface processes in a timely manner, and hence no collaborative message will be left unattended.

Any incompatibilities between the workflows of two interacting business organisations have to be reconciled before collaboration can take place [39]. Reconciliation can be done in two ways [7]. In the top-down approach, people meet and discuss and design collaborative process and then implement it locally. This is a very time consuming process especially when the organisation has many business partners, as manual negotiations have to be carried out with all business partners. The negotiations with all business partners have to be repeated after any changes to the workflow of the organisation. In the bottom-up approach, collaborative processes are extracted from partner organisations and are compared to identify incompatibilities and are adjusted to make them compatible. Negotiation is a way to remove conflicts between workflows. In the collaboration context conflicts are the differences in the interface processes of two organisations that make them incompatible. Conflicts result from dissimilarities and dissimilarities can be structural or behavioural [4]. Structural dissimilarity is the difference between two digraph representations of interface processes. Behavioural dissimilarity is the difference captured after traversing every possible flow trace [4]. The process of negotiation involves [43-47].

- Two partners with their respective interests come together with the intention of reaching some agreement.
- They identify their conflicts (if any) through communication.
- Partners try to reconcile the conflicts by using a range of strategies such as concession making, contending, problem solving, inaction, withdrawal [48] or exploration of mutual gains [49].
- If reconciliation is successful, an agreement is reached. Agreement could either be in favour of one partner or it can be a win-win situation.

In the negotiation process, two core principles should be observed [50]:

- Negotiation is a voluntary activity. Any party can break out at any time.
- A win-win situation is a successful outcome where both partners are satisfied.

A lot of work has been done on collaborating workflows of interacting organisations, both at build time and runtime stages. The next section reviews selected build time and runtime collaboration systems and approaches; their benefits and shortcomings are discussed and research issues which current collaboration systems fail to address are identified.

3.5 Existing Work on Workflow Collaboration Approaches/Systems

Workflow collaboration should be tackled both at build time and runtime. Existing work on build time and runtime workflow collaboration is reviewed in this section.

3.5.1 Existing Work on Build time Workflow Collaboration

Most of the work on cross organisational workflow collaboration in the literature deals with build time collaboration. At build time the workflows of collaborating organisations are checked for compatibility. If they are incompatible then they are reconciled accordingly. The earlier research focuses on finding a common sequence of activities in existing workflows [51]. This is normally done through manually discussing the business processes and reaching an agreement [5, 52]. The more recent research has targeted reconciling existing incompatible workflows [39, 53]. The reconciliation, if successful, leads to compatible workflows.

RosettaNet is a consortium of major electronic companies. RosettaNet provided the Partner Interface Processes (PIPs) standard which defines a broad set of supply chain processes and data elements [54]. PIPs also defines common interface tasks for supply chain collaboration; each organisation is allowed to plug in their internal processes within the interface processes. Compared to a single concrete workflow modelling approach, the abstract partner interface process approach makes the workflows of the collaborating organisations less dependent on each other [24]. In the concrete workflow modelling approach the workflows of all collaborating organisations and the

coordination process between them is treated as a single workflow spanning across organisational boundaries [24].

Schulz and Orlowska [5] proposed a three tier cross organisational workflow model i.e. coalition workflow, workflow view and private workflow. Coalition workflows are constructed on the basis of agreement reached among the business partners. Coalition workflows contain abstract services. Partners form workflow views by choosing tasks from the coalition workflows which they want to implement privately. Based on the coalition workflow, relationships between the chosen tasks are obtained. Workflow views contain abstract processes, known to the interacting organisations, outsourcing their implementation to private workflows. Routing activities like splits and joins are added to the workflow views. Private workflows are known to its owning organisations only. The workflow views and private workflows are connected through state dependencies. New private workflow could be developed or existing private workflows can be reused. This model requires interacting partners to meet and reach agreement before the workflow collaboration can be implemented. This is a highly time consuming process especially when an organisation has to do business with many partners. This model focuses on structural and control flow aspects of workflow, and does not deal with runtime data dependencies.

Aalst [52] came up with an approach that uses Message Sequence Charts to capture the communication structure in cross organisational workflows. Message Sequence Charts is a graphical language that can be used for the visualisation of communication between systems and processes. Aalst used workflow nets (WF-nets) for modelling workflows. WF-net [55] is a kind of Petri net in which tasks are modelled by transitions, and arcs are used to model dependencies. Aalst applied a three step Public-to-Private approach to inter-organisational workflows. In the first step, the partner organisations agree on a common public workflow; in the second step, the common public workflow is divided over the interacting organisations; and in the third step, the organisations create their private workflows autonomously. To ensure that the overall inter-organisational workflow is correct, a notion of inheritance is used. Each private workflow should be a subclass of the public workflow. Aalst developed an analysis tool named Woflan to verify the correctness of the workflow definitions.

Krukkert [51] proposed a solution in the openXchange project. Two activity diagrams are taken as input and compared to find out all common execution sequences. If any common sequence is found then a common activity diagram is constructed for collaboration. The enactment stage is not considered. The approach assumes all activities to be atomic and represent activity diagrams as state transition systems [56]. This is done to eliminate the parallel structures in activity diagrams [24]. For the solution to work there must be a common activity sequence in the workflows or activity diagrams of the participating organisations. If a common sequence is not found then collaboration cannot proceed. To target this problem, a mechanism to reconcile the conflicts in the activity sequence is needed.

Byde *et al.* [53] developed a negotiation framework which claims to conduct automatic negotiation over business to business (B2B) processes. A unified process is created from the processes of the collaborating organisations; the unified process is compared to individual processes to detect any differences and the cost required to remove those differences. The framework suggests adjustments to each party to remove incompatibilities. Although Byde *et al.* claim that the system can negotiate and reconcile incompatible workflows, on a closer look it is actually an extension of an approach presented by Aalst [52, 57] that can only reconcile differences in activity content [24]. The issue with the approaches put forward by Aalst and Byde *et al.* is that both these approaches only target differences caused by different activity content. Any difference regarding activity sequence is considered irreconcilable. Also, the techniques used to find the differences between the workflows have not been described.

Chen and Chung [39] presented an approach for reconciling existing workflows to bring about compatibility and support runtime execution. A software collaboration agent extracts the interface processes from two workflows that are intending to work together and gives an offer to a candidate provider which evaluates the offer and creates a counteroffer. The partner then either accepts or rejects the offer. The process of offer generation, counter-offer generation, acceptance and rejection goes on iteratively till negotiation is terminated or reconciliation is achieved. If compatibility is attained then the partners move on to enacting their workflows. This framework provides computer support for the reconciliation process and saves time and resources for the

organisations. However, the organisations are still involved in accepting or rejecting the offers and counteroffers during the reconciliation process.

3.5.2 Existing Work on Runtime Workflow Collaboration

Since workflows need to be executed, only build time collaboration is not enough. There needs to be runtime collaboration so that the transfer of files and information happens smoothly and the sequential, parallel or branching navigation of cross organisational activities are properly followed.

One way to achieve this is to treat business partners as workflow participants and expand the standalone centralised enactment service across the organisational boundaries, as discussed in [24]. In this situation business partners will need to share private data, common process definition and a centralised workflow engine. This approach couples the business partners very closely and it is also a very expensive and rigid approach. Sub-flow invocation mechanism is another alternative in which a chained process is started and all the nested sub-flows in the hierarchical workflow are completed [24]. This approach suits hierarchical workflows and requires workflow management systems and process definition language to have sub-flow invocation mechanism. This is a technically feasible approach but it is not practically viable.

Chen and Hsu [58] proposed a system to collaborate cross organisational process execution which is called Collaborative Process Manager (CPM). All interacting partners have a copy of the cross organisational workflow which is distributed to them at runtime. Each organisation is responsible for executing its own activities in the workflow through its local workflow engine. The organisations recognise their activities on the basis of role matching. Each organisation informs the interacting organisations about its progress, so that they can be prepared for the subsequent activities. The communication between partners is done through message passing with the relevant data. CPM targets distributed runtime environment and has a single global view of the workflow so collaboration management is quite straightforward [24]. The problem with this approach is that it assumes all interacting organisations run compatible workflow engines, which is not always the case in the real business world. This approach also needs common workflow specification languages to be extended to include collaborative process definition.

Chen and Chung [59] developed a bottom-up cross organisational workflow enactment approach. The approach is WfMS independent and the enactment is done via progressive linking enabled by run-time agents. To ensure that the control flow, data flow and communication aspects of cross organisational workflows are addressed properly, the concept of interaction point is introduced. Each interaction point is modelled as interface activity and agents make sure that outgoing data and incoming data are delivered to the corresponding activities accordingly. A form filling approach is used to ensure this. The corresponding agents complete a form with activity ID, interaction identifier and other relevant data and attach any required documents to the form, after an interface activity is executed. The form represents the progress of interoperation and can be used for historical record. It is assumed that the cross organisational workflows to be enacted are already compatible as compatibility issue would have been dealt with using the method described in [39].

Biegus and Branki [60] proposed an agent based framework named InDiA for interoperability between heterogeneous workflows. They used an extended process definition language to define a coordination dialogue. The centrally defined dialogue is then sent to all interacting organisations. In the dialogue, messages are arranged in the form of request-response pair. A message is either a termination message called final, which terminates the dialogue, e.g. acceptance or rejection messages; or a non-final message which requires a response, e.g. enquiry, request for amendment etc. Whenever there is an interaction point in the workflow execution, the coordination dialogue is referenced. For a sending activity, the activity requests the dialogue agent to deliver the message. A dialogue agent checks whether the message follows the dialogue flow and, if so, delivers the message. For a receipt activity, the dialogue agent checks the message flow and receives the message. The problem with InDiA is that it only allows one organisation to send a message at a time. This means if Organisation 1 sends a message, Organisation 2 must receive the message before it can send any message. Also, partial results cannot be communicated between the workflows.

The system presented by Wang *et al.* [33] for inter-organisational workflow coordination and dynamic workflow composition enables inter-organisational workflow enactment. They have extended the workflow compositional ability from build time to runtime. This means that in case of unexpected results, dynamic composition can be

done at runtime to create an alternative workflow. The system is based on the integration of software agents, web services, and workflow ontology. At build time, agents work on the behalf of web services to build flexible interaction patterns while at runtime the web services solve the interoperability problems in the real world environment. The enactment and communication infrastructure is provided by web services and the coordination aspects are dealt with by the agents. The coordination agents manage, plan and coordinate the workflow while users and resource agents are used to perform tasks. Technically the discovery, deployment and communication aspects are provided by UDDI, WSDL and SOAP. BPEL4WS provides specifications for composition and enactment. The issue with the system presented by Wang *et al.* is that it is only able to enact the workflow of a single organisation which spans across multi-organisational boundaries. It can deal with the enactment of a workflow in a heterogeneous environment having adhoc external processes, but is not able to enact the workflows of two independent organisations collaborating with each other.

The common issue with the discussed runtime systems is that none of the discussed systems is able to enact the compatible workflows of more than two independent collaborating organisations, while also taking advantage of the reusable and interoperable aspects of web services.

3.6 Conclusion

This chapter discussed key concepts related to cross organisational business collaboration with workflow management systems and reviewed current work. Some work has been done on creating compatible workflows using the top-down approach where the organisations first meet and negotiate the collaboration and then implement it locally. This is a very time consuming approach.

The bottom-up approach tries to find common sequences in existing workflows. The problem is that if no common sequence is found then collaboration cannot proceed.

Systems that support the process of reconciling existing workflows have been suggested. Although this approach saves time and resources, the collaborating organisations still have to get involved with the reconciliation process by analysing,

accepting and rejecting adjustment offers suggested by the systems to bring about reconciliation.

Workflow collaboration also needs to be dealt with at runtime so that the transfer of data and information takes place correctly and smoothly. Most of the existing work done on runtime collaboration uses message passing.

The issues with existing workflow collaboration systems are:

- Most collaboration systems require the users to get involved in the reconciliation process.
- The automatic systems that identify common paths are very limited.
- The systems cannot reconcile the workflows of more than two organisations at the same time.
- Very few systems target both build time and runtime collaboration.

To address these issues, Chapter 5 proposes a novel approach for automatic cross organisational compatible workflows generation, based on the idea of AI planning.

Chapter 4: Planning Technologies

4.1 Introduction

As web services composition is an AI planning problem [9], its accuracy and efficiency depends on the planner used for planning. This chapter discusses major AI planning paradigms and reviews the suitability of different planning paradigms and their representative planners for the web services composition problem.

Section 4.2 reviews different planning paradigms and their representative planners. Section 4.3 reasons about the suitability of these planners for the web services composition problem and defends the selection of SHOP2 as the most suitable planner for the proposed framework. Section 4.4 discusses the structure of the SHOP2 planning problem and Section 4.5 concludes the chapter.

4.2 Planning Paradigms

AI planning is a vast field and extensive research has been done to apply AI planning to web service composition [30, 61, 62]. The following sections discuss state-space based planning, graph based planning, partial order refinement planning, planning as satisfiability, planning as logic programming, planning with control knowledge and temporal planning. Representative planners from each planning paradigm are also discussed.

4.2.1 State-Space based Planning

State based planning aims to solve a problem by searching useful instantiations for operators in a state space, that result in the desired state [61]. A state space consists of finite set of states *S*, finite set of actions *A*, a transition function *f* describing how actions map one state to another, and a cost function $c(a; s) > 0$ which gives the measurement of cost associated with performing an action *a* in a state *s [63]*. A state model is a state space extended with initial goal s_0 and set of desired goals S_G [64].

State based planning can be forward state search (also called progression), or backward state search (also called regression). A progression planner, as the name suggests, starts

at the initial state and searches for action instances that bring it closer to the goal, while a regression planner starts at the goal state and searches for action instances that bring it closer to the initial state.

A solution of a state model is a sequence of actions $a_{0,}\ a_{1,\ldots,}a_n$ that results in a state trajectory $s_{0,}\ s_1 = f(s_0)_{,\ldots,}s_{n+1} = f(a_n,\ s_n)$ such that each a_i is applicable in s_i and s_{n+1} is a goal state, i.e., $a_i \in A(s_i)$ and $s_{n+1} \in G$ *[64]*.

The heuristic search planner (HSP) [65] is a state space based forward planner which uses Additive Heuristics h_{add} to guide a hill climbing search from the initial state to goal state. At each step, the child node with minimal h_{add} value is selected and the process continues until the goal is reached. In successive work, Bonet and Geffner developed the HSP2 planner [66], which also employs h_{add} but uses best-first search [67] instead of hill climbing search. Nodes are weighed by an evaluation function $f(n) = g(n) + W * h(n)$, where $g(n)$ is the accumulated cost, $h(n)$ is the estimated cost of the goal, and W is a constant. Higher values of W mean faster plan search and lower plan quality [68].

Another planner with heuristics is the fast forward (FF) planner [69]. It relies on forward search in state space, guided by heuristics that estimate goal distances using a relaxed problem, in which negative effects of actions are ignored. The number of actions in the solution to the relaxed problem is a goal distance estimate. The FF planner evaluates all successor states to find the state with a better heuristic value than the current state. This is basically breadth-first search to find a state with strictly better evaluation than the current state. This new local search strategy is called enforced hill climbing. The FF planner also uses the information from the planning graph to identify at each state those actions that have useful effects and prefers them over superfluous operators [69]. The concept is also called *helpful actions.*

Metric-FF which is an advanced version of FF was presented in [70, 71]. It supports numerical state variables which can be used in numerical constraints in preconditions (e.g. *count* > *4*0) and in arithmetic operations in effects (e.g. *cash*- = 5).

4.2.2 Graph Based Planning

Blum and Furst [72] introduced a graph planning framework which formalizes the construction, annotation and analysis of planning graphs. In a planning graph, a plan is a flow in the network flow sense. A planning graph is a directed levelled graph with a layer of proposition nodes followed by layer of action nodes and so forth. Each layer is associated with a time step.

Blum and Furst [72] introduced a planner named GRAPHLAN based on planning graphs. GRAPHLAN operates in two steps. In the first step it expands the graph until a level is reached where all propositions are present with no mutex relation between any pair. Two actions are mutex if they have inconsistent effect, competing needs or inference. Inconsistent effects occur when one action negates the effect of another action, inference occurs when one of the effects of an action negates precondition of another action, and competing needs occur when a precondition of one action is mutually exclusive with a precondition of another action. This phase is called graph expansion. The second step is the solution extraction phase and GRAPHLAN searches for potential plans in this phase. GRAPHLAN uses backward search algorithm for this purpose. GRAPHLAN is only restricted to STRIPS operators and has no support for conditional and universal operators. If too much irrelevant information is contained in the specification of the planning task, performance can decrease drastically [73].

Koehler *et al.* [74] presented the interference progression planner (IPP) to handle conditional and universal operators. It is an extension of GRAHLAN planner. Removing Irrelevant Operators and Initial Facts from Planning Problems (RIFO) strategy [74] has been added to IPP planner to address GRPAHLAN problem of performance issues for planning tasks with irrelevant information. The authors show that the computational overhead for this additional functionality is negligible. STAN [75] is another planner based on GRAPHLAN, improving GRAPHLAN in time and resource efficiency. It uses bit vectors for internally representing preconditions and effects which is very efficient and allows for resource efficient representation of planning graphs. It carries out mutex checks between actions and facts using bit manipulation operations which, again, are very efficient.

Sensory Graphplan SGP [76] is an extension to Graphplan that supports conditional effects, uncertain effects and uncertainty in initial situation [77]. It extends a separate planning graph for each possible world, keeps track of mutual exclusion across the worlds and searches backwards for a plan that works in all possible worlds.

4.2.3 Partial Order Refinement Planning

In partial order refinement planning, the nodes in the search space are formulated to be partial plans and not states, and the arcs are plan refinements. The planners based on partial order refinement planning are called Partial Ordered Planners (POP). POP planners are also called Partial Order Causal Planners (POCL). POCL planners produce partially ordered plans. In the plans produced, not all actions are in fixed order and a plan may have various linearisations.

Chapman [78] presented the TWEAK system in 1987 based on partial ordered planning which could handle conjunctive and disjunctive preconditions and conjunctive effects, and he proved his system to be sound and complete. SNLP [79] is a more advanced planner and it uses the notion of threat and safety conditions. UCPOP [80] supports actions that have conditional effects, universally quantified preconditions and effects, and universally quantified goals. Generally speaking POCL planners have been outperformed by planning graph and heuristic state space planners [61].

Nguyen and Kambhampati [81] introduced RePOP, which is a POP implementation using planning graph to compute an estimation of the cost of achieving a set of sub-goals. RePOP has the ability to detect failing plans before they get selected for refinement. REPOP outperforms state based planners and also retains the flexibility of POP frameworks [82].

The versatile heuristic partial order planner (VHPOP), presented in [83, 84] is a more recent advanced planner which use *A** algorithm to search through the state space. The *A** algorithm uses an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of getting to n from the start node and $h(n)$ is the estimated cost of reaching the goal node from n. The cost of a plan is equal to number of actions in that plan. VHPOP uses h_{add} heuristics and uses relaxed planning graph to extract data for h_{add}. VHPOP is able to

handle durative actions by attaching time related (temporal) annotations to open conditions. In durative actions, time duration plays a key role.

4.2.4 Planning as Satisfiability

In this approach the planning problem is expressed as a reasoning problem for which problem solving algorithms exist.

4.2.4 .1 Planning as Propositional Satisfiability

Kautz and Selman [85] presented planning through satisfiability checking in which a planning problem is formulated as a set of axioms. Axioms are logical statements that are assumed to be true. The axioms have a property that any model of the axioms corresponds to a valid plan. This is achieved by crafting axioms that would remove actions that would result in anomalous models. Axioms are needed to ensure that actions whose preconditions are not fulfilled are ruled out. Axioms do not have quantifications or terms. All predicates take time step as a trailing argument. For example for the block world problem [86], to achieve $on(B,A)$ from an initial situation $on(A,B) \wedge on(B, Table)$, the planning problem can be expressed as [61]:

$$on(A, B, 1) \wedge on(B, Table, 1) \wedge clear(A, 1) \wedge on(B, A, 3)$$

Further, the preconditions of the *move* operator can be formalized as:

$$\forall x, y, z, i.move(x, y, z, i) \supset (clear(x, i) \wedge clear(z, i) \wedge on(x, y, i))$$

SATPLAN [85] and BLACKBOX [87] are planners based on this approach. BLACKBOX combines features from SATPLAN and GRAPHLAN.

LGP system [88] and LGP-td [89] are also based on satisfiability checking but use a best-first search algorithm and planning graph for search heuristics.

SAT based planners perform very competitively [90]. Since SAT based planners model states explicitly by attaching a time step to the end of all axioms and predicates, it is very easy to formulate requirements on states and thus expressing complex goals [91, 92].

4.2.4.2 Planning as Description Logic Satisfiability

Berardi *et al.* [93] targeted the planning task as logical satisfiability problem using description logic. They defined a community of web services, characterized by a common set of actions and a set of web services specified in terms of the common set of actions. A web service in the community may execute some of its operations and may delegate the rest of its operations to another service in the community. The interaction protocols that are offered by the services are expressed as execution trees. An execution tree is a tree of states in which each node represents a possible state and each arc represents an action. The web services composition problem is to find an execution tree which is composed of actions of web services in the community, and matches a desired execution tree.

4.2.4.3 Planning as Petri net Reachability

Narayanan and McIlraith [15] suggested that a web service planning task could be carried out by creating a Petri net [94] based on atomic web services. A Petri net can be defined as a directed, connected, and bipartite graph in which each node represents either a place or a transition [95], where a bipartite graph is a graph having two disjoint sets of vertices. The Petri net represents all possible combinations of atomic operations. The desired goal is specified as a state of the Petri net [15]. They argued that satisfiability checking can be used to determine whether or not the goal state is reachable.

4.2.5 Planning as Logic Programming

Planning as Logic programming is an approach to encode action laws of planning domain as a logical representation, amendable to formal reasoning methods [61].

A Logic program is composed of a set of Horn clauses of the form $A \leftarrow B_1,\ldots\ldots,B_n$. A Horn clause is a disjunction of literals with at the most one positive literal, i.e. $A \vee \neg B_1 \vee \ldots \vee \neg B_n$.

A given goal G is achievable in the planning domain if and only if a related goal G^* is true in some stable model of the logic program. Di-mopoulos *et al.* [96] suggest that if

logic programs are properly encoded, performance of this approach can keep up with general purpose planning algorithms like GRAPHPLAN and SATPLAN.

In context of Web services, the SWORD toolkit [16] is based on this approach and it uses Prolog to reason about information providing services and extract plans directly from execution trace of Prolog interpreter.

4.2.6. Planning with Control Knowledge

This section reviews planning techniques that incorporate and exploit task dependent control knowledge to achieve good performance.

4.2.6.1 Hierarchical Task Network Planning

Hierarchical Task Network (HTN) planning [97-99] provides a powerful strategy to deal with large and complex real world problems. HTN planning assumes certain operators only if its preconditions hold before its execution. Operators represent primitive tasks. In addition to operators representing primitive tasks, HTN planning also support methods to decompose complex tasks into subtasks.

According to [99], there are three types of tasks in HTN Planning:

- Goal task which is desired final state,
- Primitive tasks,
- Compound tasks that can be decomposed in several subtasks.

Ordered Task Decomposition is a very popular variant of HTN planning and it enables the agent to plan the tasks in the same order in which they will be executed. This reduces the planning complexity to a great extent. Simple Hierarchical Ordered Planner (SHOP) [100] is a planner based on ordered task decomposition, which accepts goals as tasks lists and not as declarative goals. Tasks in tasks list can either be primitive or compound tasks. The desired task is decomposed in subtasks and the subtasks are further decomposed until the resulting set of tasks only contains primitive tasks so that these primitive tasks can be executed directly by calling atomic tasks. The planning is successful if the decomposition of desired task into primitive subtasks is done without

violating certain constraints. Testing is done in each round of decomposition to see if there is any violation of the given conditions.

Wu *et al.* [31] used SHOP2 [101] for web services composition. SHOP2 is an ordered task decomposition planner. Wu *et al.* presented a transformation method from OWLS to hierarchical task network [31, 32]. Since OWLS processes and HTN tasks both specify actions to get certain task done, the transformation is natural. SHOP2 can deal with large and complex problem domains.

4.2.6.2 High Level Program Execution

In high level program execution the task is to find a sequence of actions which would constitute a legal execution of a given high level program. To search out applicable actions, reasoning is done regarding preconditions and effects. Golog [102] is a logic based high level programming language, focused on specification and execution of complex actions in dynamic domains. Golog is based on situation calculus.

ConGolog [103] is an extension of Golog. It supports concurrent processes by interleaving atomic actions in its component processes; and supports interrupts and actions out of control of the interpreter. Mcllraith and Son [10, 11] extended ConGolog with ability to include user constraints, sense actions as external calls and more flexible variants of Golog sequence construct. Narayanan and McIlraith [15] transformed OWLS processes to situation calculus and Golog, so that OWLS processes can describe atomic and complex actions offered by Web services. The web services composition problem is to find an execution of the Golog program that satisfies the properties defined in the goal.

4.2.6.3 Planning as Model Checking

Planning as (by) model checking (PBM) [104, 105] approach formulates planning problem as semantic model checking problem. It is a formal verification technique to check whether a property holds in a certain system that is formalized as a finite state model.

A solution in PBM can be weak, strong or strong cyclic. A weak solution does not guarantee to achieve the goal, a strong solution guarantees to achieve the goal and a strong cyclic solution guarantees the goal assuming the loop will terminate finally

[105]. Algorithms that will always terminate and that will achieve weak [104], strong [106] or strong cyclic [107] solutions have been suggested.

MIPS [108] is a planner using planning as model checking approach and it is based on Binary Decision Diagrams (BDD). MIPS reduces the state description length by identifying implicit domain knowledge, e.g. state invariants.

Proplan [109] and BDDPlan [110] are also based on PBM approach however lack a MIPS-like pre-compilation phase and both of these are outperformed by MIPS. MIPS, Proplan and BDDPLan are designed for deterministic domains. Deterministic domains are domains where the result is completely determined by the inputs.

Model Based Planner (MBP) [111] and Universal Multi-agent Obdd-based Planner (UMOP) [112] are designed for nondeterministic domains and have the capability to handle uncertainty. In non-deterministic domains, the result is not completely determined by the inputs and an action can lead to several possible states.

4.2.7 Temporal Planning

Temporal planning refers to the ability of planners to deal with time related (temporal) aspects of planning domains and problems. Most mature planning paradigms have been extended to support temporal aspects of planning. Examples of such temporal aspects are [61]:

- Durative actions: A durative action refers to an action in which duration of the action is important e.g. a truck being loaded must not move for the duration of the loading action. Classical planning approaches formalize actions to be non-temporal, but in real world, duration of actions is often important. This makes exact time and time efficiency of an action interesting to the planner.

- Validity intervals of propositions: Classical planners assume that a proposition does not change automatically with time, and changes only if it is changed explicitly by an agent using an operator. But in real world, many ground atomic formulas (atoms) are dependent on time, e.g. a sale offer might be valid for three days. Such ground atoms are called fluents.

- Concurrent actions: Classical planning usually assumes that only one action is executed at a time but in real world it may be necessary that two or more actions are executed at the same time. Also actions that are independent can be executed in parallel.

Bacchus and Kabanza [113] presented specifications for temporally extended goals, which made it possible to express a set of acceptable sequence of states along with the final desired state to achieve. TLPlan [114, 115] is a planner that supports goals specified in an extended version of the Metric Interval Temporal Logic (MITL) [116]. Thus TLPlan addresses temporally extended goals and it is based on a forward-chaining planning algorithm. TLPlan treats each state as a database and checks it against a formula that is inverse of the desired goal. It prunes each state that satisfies the inverse formula. Temporally extended goal formula can also be extended to the notion of domain control knowledge, which guides the planner about the desired and undesired properties of the states it is going to identify.

TALPlanner [117] is an extension of TLPlan. It is also based on a forward search planning engine. It uses TAL language to specify the planning goals and domain control knowledge. TAL is narrative-based linear metric time logic and it is especially good for reasoning about actions in incompletely specified dynamic environments. TALPlanner takes a TAL goal formula as input and outputs a solution to the goal formula.

4.3 Relevance of Planners for Web Service Composition

Generally speaking, the core requirements of planners to deal with web service composition problem are as follows [61]:

- The domain complexity should support universally quantified effects, explicit mark-up of sensing actions and nondeterministic service results. The planner must be efficient enough to perform well in complex web domains
- Support for complex goals i.e. “hints” to guide the planner about the sequence of actions.

- The ability to deal with incomplete information calls for support of sensing actions, for example to query a database to get a list of certain companies; sensing actions help the planner acquire data it needs.
- Support for nondeterministic behaviour is also needed. A web service might fail or might output invalid or wrong data.
- There needs to be support for parallel activities since workflows can have parallel activities and independent activities can also be executed in parallel.

A review of the planners on the basis of the above requirements is given below.

Table 4.1 Web services composition requirements vs. domain independent planners (adapted from [61])

Planner	**Domain Complexity**	**Extended Goals**	**Sensing**	**Non-deterministic Actions**
FF (state based)	PDDL 2.1 level 1	No	No	No
FF-Metric (state based)	PDDL 2.1 level 1, level 2	No	No	No
HSP 2.0 (state based)	PDDL/ADL without complex precond./goals	No	No	No
IPP (GRAPHLAN based)	PDDL/ADL	No	No	No
SGP (Graphplan based)	PDDL/ADL, without complex negations in precond./goals	No	Support sensing actions that determine the truth value of formulas	Produces contingent plans
STAN4 (Graphplan, state	The STRIPS + Equality	No	No	No

based)	subset of PDDL			
VHPOP (POP based)	PDDL 2.1 level 1 and 3	No	No	No
BLACKBOX (SAT, Graphplan based)	PDDL/STRIPS with Restrictions	No	No	No
LGP (SAT based)	PDDL 2.1 levels 1,2,3	No	No	No

Table 4.1 (adapted from [61]) contrasts a collection of representative domain independent (neoclassical) planner implementations against the collection of core requirements for web service composition problem.

The table can be summarized as:

- All neoclassical planners support domain complexity i.e. a significant subset of ADL.
- Except SGP which supports incomplete initial state and sensing operations, none of the neoclassical planners support the requirements discussed above.

To use neoclassical planners for web services composition problem, a proper architecture that decomposes the planning problem into a set of sub problems that match the planner capabilities is needed [61]. Haigh [118] presented an Execution Monitoring & Re-planning Architecture, in which a controller decomposes the problem into a sequence of simpler problems. The splitting up of a problem into a sequence of smaller planning problems makes the planning for sensing actions possible for classical planners [119].

Table 4.2 Web services composition problem requirements vs. domain dependent planners (adapted from [61])

Planner	**Domain Complexity**	**Extended Goals**	**Sensing**	**Non-deterministic Actions**
SHOP2 (HTN	PDDL/ADL	yes, as HTN	HTN	HTN Methods

based)	with metrics and time	Methods	methods may contain explicit sensing actions	can be designed to deal with nondet. actions
ConGolog (High level prog.exec.)	Sit.Calc.	yes, as Golog program executions, incl. userdef. constraints	Golog program may contain sensing actions/subgoals	Golog programs can be designed to deal with non-det. actions
MIPS (Planning as Mod.Check.)	PDDL/STRIPS + negative preconditions and univ. conditional effects	supports Computation Tree Logic (CTL)	No	No
MBP (Planning as Mod.Check.)	PDDL2.1+extensions	temporally extended goals as supported by NuPDDL	Yes	Yes, can create strong(cyclic) or weak plans
TLPlan (Temporal)	ADL + metrics	temporally extended goals as supported by MITL	No	No
TALPlanner (Temporal)	PDDL 2.1 (or TAL)	TAL narratives	Yes	No

Table 4.2 (adapted from [61] with modifications) contrasts a collection of representative domain dependent planner implementations against the collection of core requirements for web service composition problem. It is evident that there is a much broader support for the given requirements.

The summary of the table is:

- All representative domain dependent planners support domain complexity and extended goals.
- SHOP2, MBP and ConGolog support sensing actions and non-deterministic actions.

It is obvious that domain knowledge is the key to solving web services composition problem. In web services composition, the domain knowledge is in the form of web services descriptions. Therefore, it becomes necessary to translate the web services descriptions to domain control information. Wu *et al.* presented a technique for transforming web processes descriptions from OWL-S to HTN methods for SHOP2, which is based on ordered task decomposition [31, 32]. In SHOP2, sensing actions are passed to the planner as part of the task list and the planning agent does not decide on its own for the need of sensing actions but acts as directed in the task list. This is a practical and useful approach [61].

ConGolog is an alternative approach, supporting most of the discussed requirements. Narayanan and McIlraith [15] transformed OWL-S processes to situation calculus and Golog. So a web service can be translated from OWL-S into domain control knowledge in ConGolog. The main advantage of ConGolog for Web Service Composition problem is that it supports parallel activities and therefore it can create parallel workflows. Generally speaking, SHOP2 is believed to be more efficient than ConGolog [13, 31, 32].

MBP is also able to plan in non-deterministic domains and support sensing actions, but it is outperformed by SHOP2 in efficiency [120].

SHOP2 will be used for planning in the proposed research. The reasons for selecting SHOP2 are as follows:

- SHOP2 supports complex domains, extended goals, sensing actions and non-deterministic actions.
- There has been active research in applying SHOP2 to web service composition problem [31, 32, 121].

- SHOP2 and OWLS have a similar mechanism for representing atomic tasks and decomposing composite tasks into atomic tasks.
- SHOP2 was one of the top four planners in the planning competition 1999 [31], based on efficiency.
- There is a java implementation of SHOP2 freely available for use.

4.4 Structure of SHOP2 Planning Problem

A SHOP2 planning problem generally has the following structure [61]:

- Description of the possible actions in formal language (domain theory).
- Description of the initial state of the world.
- Description of the desired goal.

Domain theory, initial state and desired goal must be formalised for planning. To perform a planning task it is necessary to have a full domain description. The domain is a collection of operators and methods. *A* SHOP2 operator can be formally defined as an expression of the form $O = (h(\vec{v})\ Pre\ Del\ Add)$ [13], Where

Pre = the list of preconditions of *O*
Add = the list of positive effects of *O*
Del = collection of all negative effects of *O*
h is a primitive task with a list $\vec{v}$ of inputs.

An atomic process *IssueInv* that takes *goods* as the precondition and creates *Invoice* as the output can be represented as a SHOP2 operator as follows:

```
(:operator (!IssueInv)
    ((goods))
    ()
    ((Invoice))
)
```

The operator *IssueInv* can be executed when its precondition *goods* holds. *IssueInv* produces *Invoice* as the post-condition after the execution. The operator *IssueInv* has no negative post-conditions and so the delete list is empty. An atomic processes is represented as an operator, its inputs and preconditions are represented as preconditions of the operator and its outputs and effects are represented as post-conditions of the operator.

A method can be formally defined as $M = (h(\vec{v}) \; Pre_1 \; T_1 \; Pre_2 \; T_2 \; ...)$ [13], Where

Pre_i is a precondition expression
T_i is a set of subtasks.

A sample composite method that defines a composite process for a hypothetical vendor having atomic processes *create_goods* and *dispatch_goods* can be given as:

(:method (*CompositeProcess*)
    (*goods*)
        (*!dispatch_goods*)
    (*payment*)
        (*!create_goods*)
)

The method *CompositeProcess* states that if the precondition *goods* holds then the planner must execute the operator *dispatch_goods*, and if the precondition *payment* holds then the planner must execute the operator *create_goods*. So *goods* is Pre_1 and *dispatch_goods* is T_1 from the formal definition. Similarly, *payment* is Pre_2 and *create_goods* is T_2. The *CompositeProcess* method gives a mechanism for decomposing the composite process of the hypothetical vendor organisation into atomic tasks.

The SHOP planning problem can be defined as [13], "*A planning problem for SHOP2 is a triple (S,T, D), where S is initial state, T is a task list, and D is a domain description. By taking (S, T, D) as input, SHOP2 will return a plan* $P = (p_1 p_2 ... p_n)$*, that is, a sequence of instantiated operators that will achieve T from S in D.*"

4.5 Conclusion

This chapter provides a discussion of AI planning technologies. Major planning paradigms and their representative planners are discussed, and the relevance of these planners to web services composition problem is explained.

The domain for web services composition can be very complex, since it can have a large number of web services from very diverse sources. Therefore, an efficient planner is needed to plan for the complex domains in a reasonable time. Similarly web services can fail or return a wrong result, so there is a need to create alternate plans so that alternative plans can be executed in the case of failure. Extended goals are also important, because they can provide hints to the planner on how to proceed with planning.

Existing domain independent (neoclassical) planners do not support most of the requirements of web services composition. Since automatic workflow generation is based on web services composition, domain independent planners are not well suited for automatic workflow generation. On the other hand, domain dependent planners are well suited for automatic workflow generation. Among the reviewed planners, SHOP2 and ConGolog are the most suitable planners for automatic workflow generation problem. The proposed framework uses SHOP2 for planning. SHOP2 has been selected due to its efficient planning, similar representation of tasks as OWLS, support for web service composition requirements and its free availability as Java source.

Chapter 5: Cross Organisational Compatible Workflows Generation and Execution: An Integrated Approach

5.1 Introduction

With the increase in demand for automatic workflow generation and automatic workflow collaboration, active research has been done in both these fields. Systems to automatically generate workflows have been reported [10-16, 33, 35, 36]. Other systems have been presented to deal with automatic workflow collaboration among interacting organisations at build time to ensure compatibility among collaborating workflows [5, 39, 51-53]. Work has also been done on runtime collaboration among compatible workflows so that the interacting workflows can be executed together and the exchange of data and information can be carried out, not only among the activities within the organisations, but also among the interacting activities that are on the boundaries of the collaborating organisations [33, 58-60]. To ensure the execution of collaborating workflows of multiple organisations, the runtime collaboration also has to ensure that the sequences of activities within the organisation and on the boundaries of the interacting organisations are followed.

Previous research generally deals with either automatic workflow generation or workflow collaboration, but not the integration of both. Similarly, most of the workflow collaboration systems support workflow collaboration either at build-time or runtime. To exploit the maximum benefits of business process automation, workflows should be automatically generated at build time and executed at runtime, in coordination with the workflows of the collaborating organisations. If automatic workflow generation, build time workflow collaboration, workflow enactment and runtime workflow collaboration are targeted as independent aspects of workflows, the overall benefits of business process automation are not achieved. A system that is able to generate workflow automatically for a single organisation but leaves the organisation to reconcile the workflow with the interacting organisations in case of incompatibilities provides limited

benefits. The basic objective behind automatic workflow generation is to avoid continuous remodelling and adaptation when changes are required. Similarly there are limited benefits in a workflow collaboration system that supports the reconciliation process but requires the users to model their workflows initially and be involved in the reconciliation process by making decisions about every adjustment in the workflow. The integration of these aspects can take the workflow modelling and workflow collaboration from semi-automation to a full automation, which is the basis for saving time and resources.

Therefore, there is a need for a framework that will enable the complete automation of workflow modelling, workflow collaboration and workflow enactment process. This thesis reports the development of such a framework. The proposed framework is based on an integration approach. The integration approach combines automatic workflow generation, build time workflow collaboration, runtime enactment and runtime workflow collaboration to exploit the benefits of each one of these components. This chapter discusses this integration approach in detail.

Section 5.2 highlights the assumptions made for the integration approach, Section 5.3 outlines the requirements for workflow collaboration, Section 5.4 explains the integration based cross organisational compatible workflows generation and execution approach, Section 5.5 discusses cross organisational control flow and data flow and Section 5.6 concludes the chapter.

5.2 Assumptions

To define a starting point and clear context for the integration approach, the following assumptions have been made.

5.2.1 Naming Convention

The planning is done on the basis of preconditions and effects of operators generated by translating OWLS processes into the SHOP2 domain. This means that the interacting organisations in general will follow a similar naming convention for the inputs, outputs and processes names of the interface activities so that compatible workflows could be generated and collaboration could be carried out at runtime among the sending and

receiving processes. A standard name/description map or an ontology can be maintained to serve as a guide in this regard. OWLS processes are required to use unique names for unique inputs, preconditions, outputs and post conditions.

5.2.2 Multi-Lateral Collaboration

To ensure maximum usability of the proposed approach and framework, it is assumed that arbitrary number of organisations can collaborate with each other. This assumption is in line with the real world business environment in which more than two organisations can collaborate simultaneously, e.g. in a Vendor/Customer/Supplier scenario three organisations need to collaborate together.

5.2.3 Readiness for participation

The collaboration among the organisations is based on the interface activities. The organisations must have access to sending and receiving web services in order for the data to be delivered across cross organisational boundaries. Also, in some cases the organisation should be involved in making decision on the data being sent and received in order to enable the flow of execution of the generated workflows.

5.2.4 OWLS Processes

Any atomic, simple or composite OWLS processes can be passed to the proposed framework. Composite processes are assumed to have a complete decomposition into atomic processes. Such composite processes are executable. The effects and outputs of the atomic and composite processes are assumed to be unconditional. It is assumed that all atomic services in the workflows will execute without failure and generate their expected outputs and effects.

5.2.5 Planning

In the planning process, it is assumed that the world is not changed by any other agent and the initial state contains all the necessary information of the domain for the planning to be done.

5.3 Requirements

In order to bridge the gap in the integration of workflow generation, build time workflow collaboration, workflow enactment and runtime workflow collaboration, a number of requirements have been identified.

5.3.1 Loose Coupling

Business collaboration takes place in a distributed environment. The collaborating organisations are independent business entities having autonomous business processes. The organisations should be able to change their workflows without affecting the workflows of the partner organisations. The collaborating organisations should have just sufficient knowledge about the workflows of the collaborating organisations and should not depend on the computational or representational details of each other's workflow. Web services architecture is a loosely coupled architecture and it limits the effects of the changes in the collaborating workflows and simplifies design [122].

5.3.2 Reusability

Since frequent changes may occur to the workflows due to the continuous evolution process in online business, demand for reusing existing software components and services has increased. Web services architecture provides a mechanism for reusing existing units of work done. As web services are self-contained units of application logic [19], which can be can be discovered, connected to and executed over the internet, they enhance reusability in business processes. Organisations can outsource the implementation of an activity to a service already developed by another organisation as long as the service can provide the desired functionality. The service provider could provide the services for free or charge a fee for it.

5.3.3 Cohesion

Cohesion is the degree of functional relatedness in the operations of a service [122]. In workflows, the services and service operations should be closely related and should contribute towards the execution of one problem and related tasks. Logically, the services should all belong to the same general category. Cohesion increases the clarity

of business processes, decreases coupling, increases the potential for reusability and simplifies further improvements.

5.3.4 Interoperability

In business collaboration process, the interacting organisations are distributed and independent business entities. They may use diverse platforms, databases and applications. Despite the diversity, cross-platform interoperability among the collaborating business organisations should be supported. Web services can provide such interoperability. Web services use standard light-weight XML based messaging protocols and WSDL access descriptions to allow interoperability among diverse organisations.

5.3.5 Modularity

Workflows can be created from diverse and independent sources. Therefore, automatic generation and collaboration systems must support modularity. They must produce situation specific instantiated workflows by integrating web services from diverse organisations. The SHOP2 methods and OWLS composite processes support modularity. The SHOP2 planner can plan about web services from diverse sources and order the necessary ones into a workflow which will achieve the desired goals when executed.

5.4 Cross Organisational Compatible Workflows Generation and Execution Approach

Cross organisational compatible workflows generation and execution approach integrates automatic workflow generation, build time workflow collaboration, runtime workflow collaboration and workflow enactment. This approach enables the development of a framework that can create compatible workflow for multiple collaborating organisations and support the enactment of the generated workflows.

At build time, workflow collaboration is done at the time of automatic generation of workflows for multiple collaborating organisations. At runtime, collaboration is carried

out among the collaborating organisations by enacting the generated compatible workflows.

5.4.1 Automatic Workflow Generation

Workflows are generated at build time by using AI planning. The SHOP2 planner is used for planning. It reasons about a pool of available web services and the web services that are able to provide desired actions are added to the workflow until the workflow is able to achieve the overall goals, or all possible choices are tried and no valid workflow is possible. The planning is done by mapping the inputs, preconditions, outputs and effects of the services as preconditions and post-conditions of atomic operators, to order the services that need to be executed to form the workflow to achieve the desired goals. Execution of the workflow achieves the desired result.

SHOP2 needs a domain for planning. A domain is the formal description of the environment in which the planning takes place. For automatic workflow generation problem in this project, the domain is a collection of OWLS process definitions which are grounded in actual web services. Since SHOP2 requires the domain to be in HTN format, the OWLS process definitions are translated into HTN format. An algorithm has been developed to translate OWLS processes into SHOP2 format. SHOP2 plans for the domain of HTN formalised process definitions to create workflows of atomic processes which have their grounding in actual web services and can be executed directly.

5.4.2 Build-time Workflow Collaboration

Workflow collaboration at build time is not carried out in the usual business process reconciliation way. Rather, the approach incorporates the collaboration stage into the workflow generation stage. The interacting organisations operations are dealt with together, i.e. the domain descriptions of all interacting organisations are renamed and grouped together into a single joint domain together with the set of goals. All possible workflows which can achieve the joint goals are generated. After workflow generation, the activities within each joint workflow are then separated into a set of compatible workflows for the different organisations involved. The fact that planning is done to generate joint workflows that are able to achieve the combined goals of all interacting organisations ensures that each workflow can be separated into a set of compatible

workflows for the organisations. If an activity creates an incompatibility deadlock then it will not get added in the joint plan in the first place.

5.4.3 Workflow Enactment

After automatic generation of compatible workflows, runtime support is provided for workflow enactment. Since the generated workflows are, in essence, the workflow of atomic OWLS processes having their grounding in actual WSDL web services, the OWLS enactment mechanism can be used to directly enact the atomic OWLS processes. The OWLS enactment mechanism can enact the OWLS atomic process by executing the web services in which the atomic process is grounded. The output generated by a service will be used as the input of another corresponding service and thus the workflow execution can continue till the end.

5.4.4 Runtime Workflow Collaboration

Since the integration approach has to deal with workflow enactment of multiple organisations, collaboration is also required at runtime. The collaboration among cross organisational activities is enabled by using sending and receiving activities, also known as interface processes [4]. Whenever a sending activity is encountered, the data, information or documents to be sent are uploaded to a central server. Whenever a receiving activity is encountered the uploaded data, information or documents are downloaded from the server and processed. The uploading and downloading technique is used because:

1. if an organisation has to send a document to many different partners, it does not have to do it many times. It can upload it to the central server and all partners can download it accordingly,

2. it also decouples the collaborating organisations from each other completely at runtime, which is a desired quality [52, 57].

5.5 Cross Organisational Control Flow and Data Flow

Workflows are based on control flow and data flow. Control flow refers to the transitional links between the activities in the workflow [25]. Data flow is the flow of

information from one activity to another but may not follow the control flow. The data flow rests on top of control flow [24]. Since the generated workflows are essentially workflows of atomic processes, the control flow is automatically modelled on the basis of the inputs, preconditions, outputs and effects of the atomic processes. After the activities have been ordered in a workflow, an atomic process can only be executed only if the atomic processes that come before it get executed and generate the outputs that goes in as inputs for atomic process(es) downstream. A record of generated outputs is kept in a hashmap at runtime, so that they can be used as inputs to the corresponding activities. The sequential order is followed in the similar way for both in-house and external processes.

The cross organisational data flow is not so straightforward. To enable the correct handling of cross organisational data flow, the concept of sending and receiving activities has been adopted from [4] to ensure the exchange of information between cross organisational activities. Sending activity is a point of interaction where information is sent to the collaborating partners. A receiving activity is a point of interaction where information is received from collaborating partners. For this project, the naming convention for a sending activity is represented by following the activity name with the characters "_s" e.g. an activity that sends insurance certificate to a customer can be represented as *InsuCert_s*. The name of a receiving activity is represented by following its name with the characters "_r" e.g. an activity that receives advance payment can be represented as *AdvPay_r*. Sending activity uploads the information, data, document and/or messages to a central server while receiving activities download the respective information, data, document and/or messages from the server for analysis and decision.

5.6 Conclusion

This chapter has identified that the separation of workflow generation, build time workflow collaboration, runtime enactment and runtime workflow collaboration limits the overall benefits. Since these aspects are highly related functionally, their integration combines and strengthens the individual benefits of each one of these aspects.

On the other hand, the workflow collaboration systems must support low coupling, cohesion, interoperability, reusability and modularity. Web services architecture has the potential to support these requirements through its standard XML interface and light weight messaging protocols.

To exploit the benefits of automatic workflow generation, build time workflow collaboration, workflow enactment and runtime workflow collaboration, an approach based on the integration of these four related aspects is presented. To deal with the requirements of the workflow collaboration, web services composition is used for automatically generating compatible workflows for multiple organisations.

To show the advantages of the integration approach over the stand alone approaches, a framework is proposed to automatically generate compatible workflows for multiple collaborating organisations. Chapter 6 discusses the architecture, implementation and technical details of the proposed framework.

Chapter 6: A Framework for Cross Organisational Compatible Workflows Generation and Execution

6.1 Introduction

As identified in Chapter 5, there is a strong interdependence among automatic generation, build time collaboration, runtime enactment and runtime collaboration aspects of workflow. Their integration strengthens and combines the individual benefits of each one of these aspects of workflow. To target the research gap in literature regarding their integration, an integrated approach for cross organisational compatible workflow generation and execution has been presented in Chapter 5. The thesis proposes a novel framework to exploit the advantages of the integrated approach. The framework enables the generation of compatible workflows for multiple collaborating organisations, from their process definitions. The framework also supports the enactment of the generated workflows and provides runtime collaboration among the enacted workflows. This chapter presents the framework in detail.

Section 6.2 presents the modifications made to SHOP2 planner in order to make it suitable for a multi-organisational web services domain, Section 6.3 gives the design and architecture, and Section 6.4 presents the detailed functionality of the cross organisational compatible workflow generation and execution framework, Section 6.5 explains the implementation and technical details and Section 6.6 concludes the chapter.

6.2 Adapting SHOP2 for Workflow Generation Problem

The collection of OWLS processes of the collaborating organisations are represented in the form of a SHOP2 method, which represents the inputs and preconditions of the OWLS processes (represented as preconditions in SHOP2) and services (tasks) in an *if-then-else* format. SHOP2 adds the task lists in the workflow on the basis of their preconditions. SHOP2 adds the task list whose precondition is true in the current state

of the world into the workflow (plan) and keeps on checking with the updated state of world.

Inputs and preconditions of the OWLS processes are represented as preconditions of SHOP2 operators. The preconditions, especially the preconditions representing data inputs, will remain true in the entire lifecycle of planning until explicitly made false by an operator. If the atomic processes do not explicitly make their inputs false, SHOP2 will keep repeatedly adding the first task list whose preconditions are true in the workflow. This will create an infinite loop. We cannot assume the processes to always explicitly make their inputs false, since more than one process could be dependent on the same input. This means another task list could be added in the plan on the basis of the input or its combination with other inputs.

Similarly, if a precondition in the *if-then-else* method is true for which the task list is to decompose a method, the method will keep repeatedly getting decomposed into primitive tasks and the loop will continue infinitely. In both cases of the infinite loop, SHOP2 will never reach the solution. To remove this issue, the proposed framework uses an extension of SHOP2 which does not add a task repeatedly in the plan. Similarly it does not decompose a method repeatedly into tasks.

The extended SHOP2 algorithm is shown in Figure 6.1. The algorithm is an extension to the SHOP2 algorithm presented in [13]. The main SHOP2 algorithm has been extended to avoid repeatedly adding the same tasks in the workflow (plan) and decomposing the same method repeatedly.

If '*s*' is the current state of the world, '*T*' is the task list and '*D*' is the domain

```
1. procedure SHOP2(s, T, D)
2       if T is empty then return empty plan
3       Let t be the first task in T
// To avoid adding operators repetitively in the plan, an operator whose precondition
   is true in the current state of the world and which has not been added in the plan
   before, is added in the plan .
4       if t is a primitive task then
5               Find an operator o = (h Pre Add Del) in D such that
                        o is not already added in the plan AND h unifies with t AND s
                                satisfies Pre
6               if no such o exists then return failure
7               Let s0 be s after deleting Del and adding Add
8               Let T0 be T after removing t
9               return [o, SHOP2(s0, T0, D)]
// To avoid decomposing methods repetitively, a method whose precondition is true in
   the current state of the world and which has not been decomposed before, is
   decomposed into operators.
10      else if t is a composite task
11              Find a method m = (h Pre1 T1 Pre2 T2 . . .) in D such that
                        h unifies with t and m has not been decomposed already
12              Find the task list Ti such that
                        s satisfies Prei and does not satisfy Prek, k < i
13              if no such Ti exists then return failure
14              Let T0 be T after removing t
                        and adding all the elements in Ti at the beginning
15              return SHOP2(s0, T0, D)
16      end if
17 end SHOP2
```

Figure 6.1 Extended SHOP2 Algorithm for Workflow Generation

6.3 Architecture

Figure 6.2 shows the general architecture of the developed cross organisational compatible workflows generation and execution framework. Although there can be more than two collaborating organisations, for clarity the figure only depicts two.

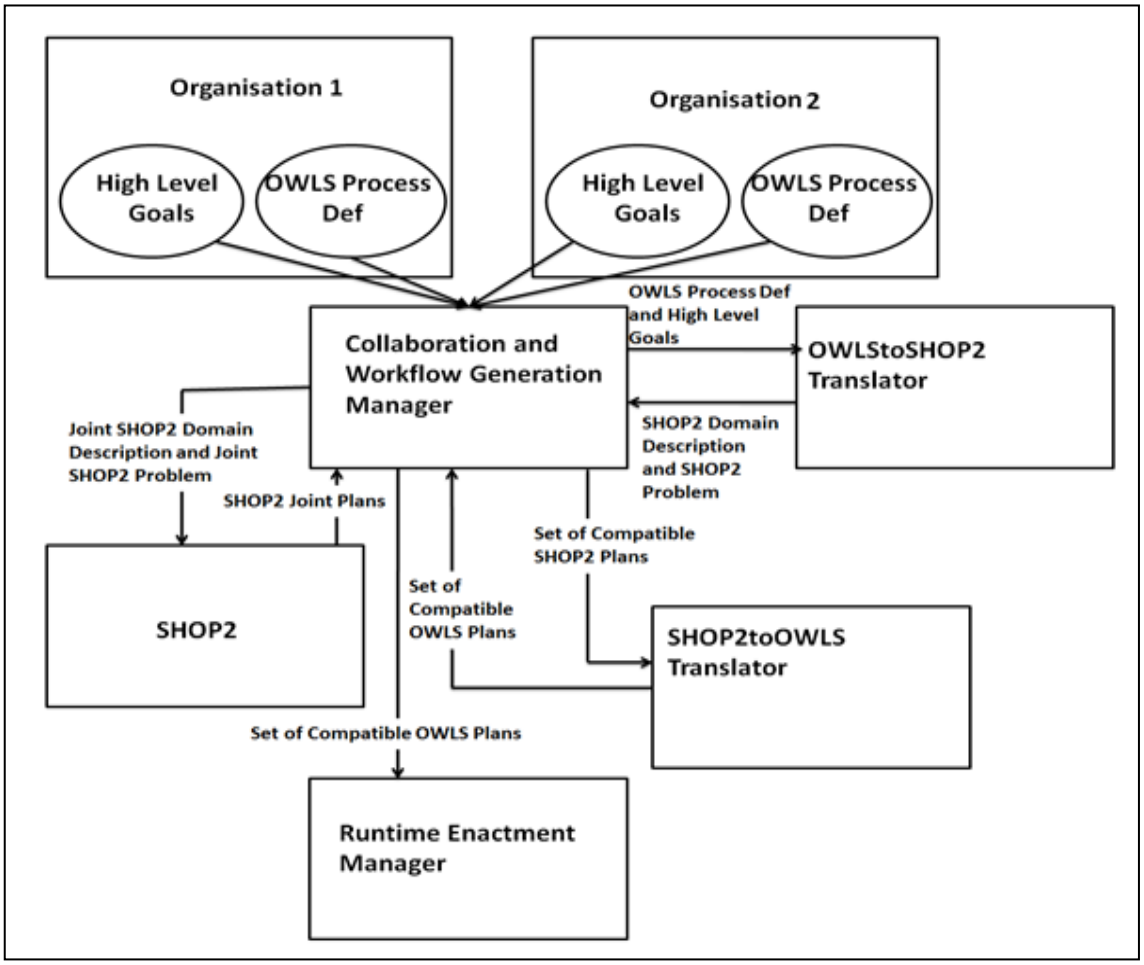

Figure 6.2 Architecture of the Developed Framework

As shown in the figure, the interacting organisations pass their OWL-S process definitions and high level goals to Collaboration and Workflow Generation Manager (CWGM). CWGM removes all those processes from the process definitions that are not used in workflow generation. CWGM passes the remaining process definitions to OWLStoSHOP2 translator, which translates them into SHOP2 domain descriptions. OWLStoSHOP2 translator also translates high level goals into a SHOP2 problem. Preplanning analysis of the domain and problem is done so that operators, inputs, preconditions, outputs and effects of collaborating organisations can be tracked. CWGM identifies operators in the domain that can enable the creation of multiple plans. Based on identified operators, methods are inserted into the domain description to ensure the creation of multiple plans. The inserted methods are used by SHOP2 to identify alternate composition paths, and hence to create multiple plans.

CWGM collapses SHOP2 domain descriptions of all interacting organisations into a single joint SHOP2 domain. SHOP2 problems of all interacting organisations are collapsed into a single joint SHOP2 problem. The joint SHOP2 problem and the joint SHOP2 domain are passed to SHOP2 planner which creates all possible joint plans. A joint plan is a plan for all collaborating organisations which achieves their combined goals from their combined initial states. Each joint plan is subdivided to create a set of collaborating plans, one plan for each organisation, compatible with each other.

The set of compatible plans with the least number of activities is highlighted to the interacting organisations for execution. The interacting organisations select the highlighted set of compatible interacting plans or any other set of compatible interacting plans for execution, according to their preferences. The selected set of compatible SHOP2 plans is transferred to SHOP2toOWLS translator to translate the SHOP2 plans into OWLS workflows. The selected set of compatible plans represents a set of compatible workflows of OWLS processes at this stage. OWLS workflows are further passed to Runtime Enactment Manager which executes actual WSDL services modelled by the activities (OWLS processes) in OWLS workflows and makes sure that the transfer of information and data among the collaborating organisations takes place smoothly.

6.4 Functionality

Figure 6.3 shows the flow of functionality of the developed framework. The developed framework takes OWLS process definitions of the collaborating organisations as input, reads the process definitions, translates them into HTN format, merges the domains together, creates multiple sets of compatible workflows and executes the selected set of compatible workflows. The following sub-sections discuss the detailed functionality of each step in the flow diagram, and present the algorithms involved in each step.

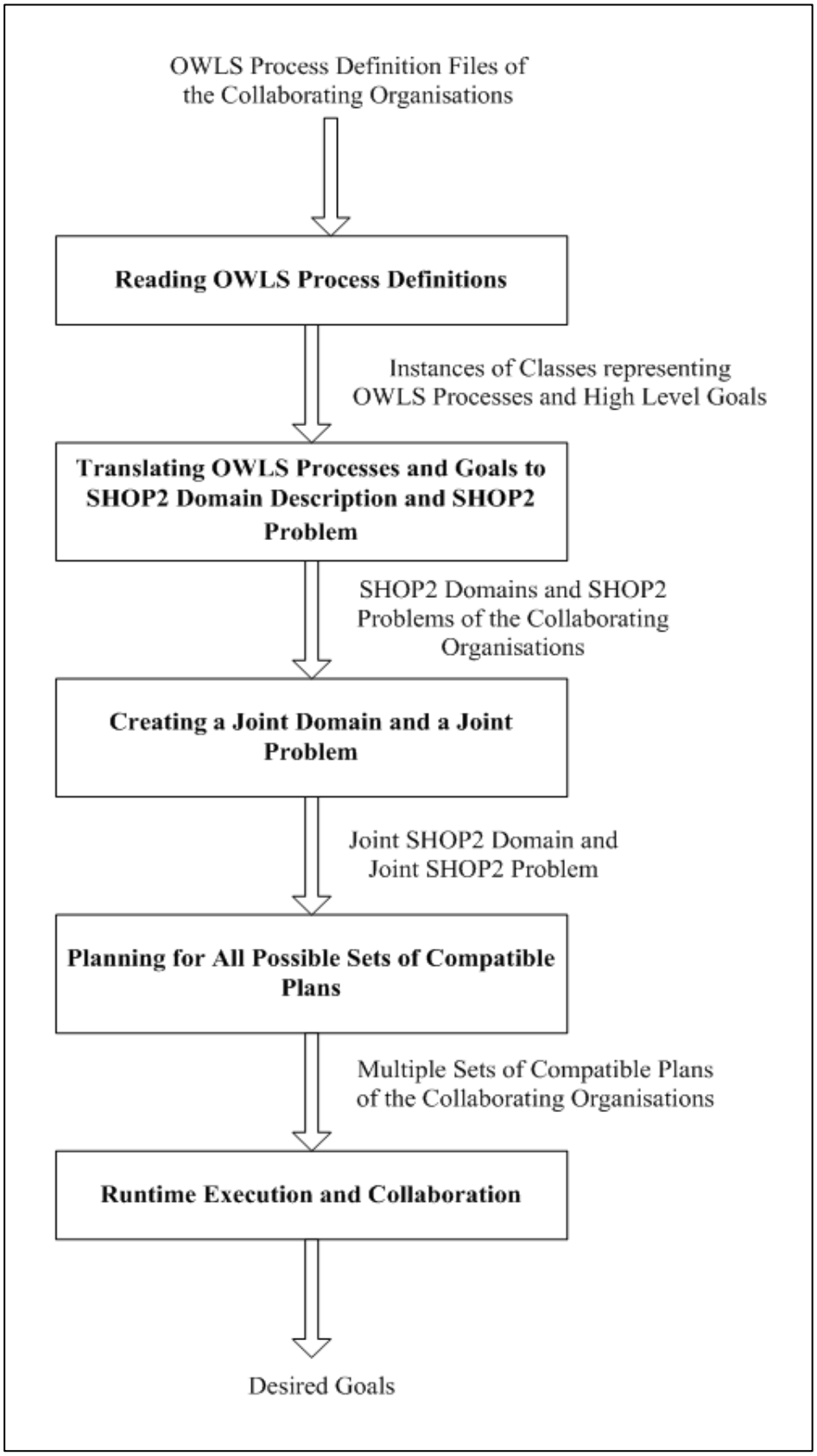

Figure 6.3 Flow Diagram of the Functionality of the Developed Framework

6.4.1 Reading OWLS Process Descriptions

A prototype system has been implemented to demonstrate the functionality of the framework shown in Figure 6.2. Collaborating organisations can load their OWLS process definitions to CWGM using an interactive GUI of the prototype. The collection of OWLS process definitions are loaded in the form of an OWL file that imports the atomic, simple and composite processes of the organisation. The OWLS process definitions can also be loaded in the form of a single composite process. Figure 6.4 shows the collection of OWLS processes of a hypothetical customer organisation. As shown in the figure, the OWL file imports all OWLS processes of the customer. The imported processes are loaded to the system. The OWLS processes can model a local web service or a web service out of the boundaries of the organisation.

```
<owl:Ontology rdf:about="">
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/AdvPay_s.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/InspCert_r.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/CheckInspCert.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/IssueSA.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/SA_s.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/BL_r.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/INV_r.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/CustomsDeclaration.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/InsuCert_r.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/TakeDelivery.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/PresaleInspection.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/CertOrigin_r.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/ApprovePayment.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/InvPay_s.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/IrrelevantProcess.owl" />
        <owl:imports rdf:resource="http://158.125.103.196/OWLS%20processes/Customer/AnotherIrrelevantProcess.owl" />
</owl:Ontology>
```

Figure 6.4 Collection of OWLS Processes of Customer

OWLSReader module of the CWGM reads the owls process definitions included in the OWL file. The module is based on OWLS API [123]. OWLS API is a Java API for programmatic access to read, execute and write OWLS service descriptions. The initial states and goal states of the collaborating organisations can be selected from the GUI. All processes are loaded from the OWLS process definitions and prefixed with organisation number for keeping track of the operators and workflows in the collaboration process. For example an atomic process *PaymentCheck* of the first

organisation that loads its processes will be prefixed with *Org1* and will become *Org1PaymentCheck*. Figure 6.5 shows the GUI of the implemented prototype. As shown in the figure; processes, inputs, preconditions, outputs and effects are loaded to the system from OWLS process definitions; and initial states and goal states can be selected at GUI.

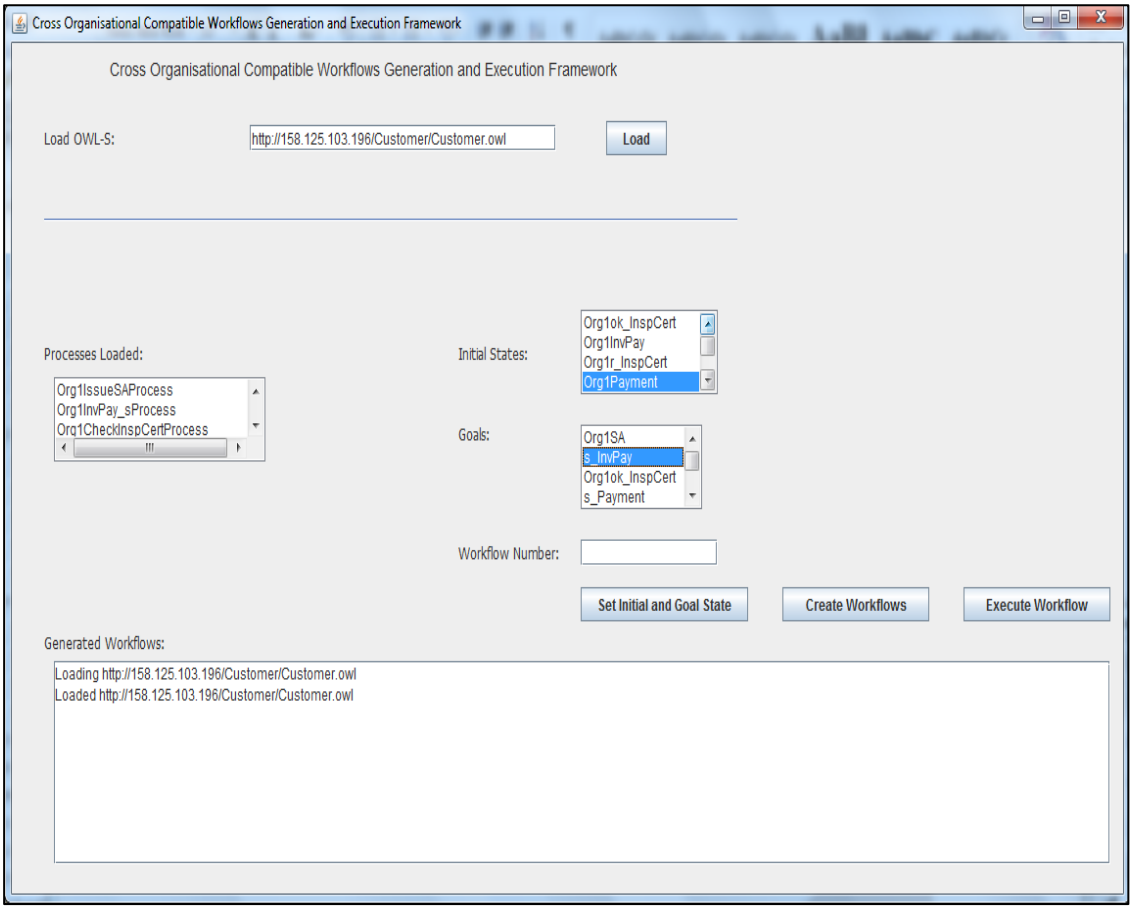

Figure 6.5 GUI of the Prototype

After reading OWLS process definitions of collaborating organisations, the loaded processes are checked for their usability in the workflow generation process. The processes that are not used in the workflow generation are deleted from the list of loaded processes. This makes the translations of OWLS processes into SHOP2 domain time efficient, since only the processes used in planning for workflow generation are translated. This also makes the planning process quicker, since the SHOP2 planner only plans about processes strictly used in the planning process. The algorithm Remove-Unused-Processes(*Q*) which takes a list *Q* of OWLS atomic processes definitions as input and returns a modified list of OWLS atomic process definitions is given below. A

process that has no other process from the same organisation or collaborating organisations dependent over it and none of its outputs or effects belong to the set of goal states is deleted from the list of the loaded processes. The algorithm is called recursively every time a process is removed, until there is no unused process or the list is empty. The recursive call makes sure that there is no unused process in the modified list.

Remove-Unused-Processes(*Q*)

Let *Q* be a collection of OWL-S atomic processes definitions
    If *Q* is empty
        Return *Q*
    Else
        For each atomic process definition Q_o in *Q*
            If Q_o does not have a process definition from *Q* dependent over it and none of its outputs or effects belong to the set of goal states
                Remove Q_o from Q
                Let Q' be *Q* after removing Q_o
                Return Remove-Unused-Processes(*Q'*)
        End For each
    End If
End Remove-Unused-Processes

6.4.2. Translating OWLS Process Definitions to SHOP2 Domain Descriptions

The OWLStoSHOP2 translator module translates OWLS process definitions into SHOP2 domain descriptions. The OWLStoSHOP2 translator also translates initial states and high level goals selected from the GUI into a SHOP2 problem. The similar mechanism of representing tasks and decomposition of complex tasks into primitive tasks in OWLS process ontology and HTN planning makes the translation straightforward [13, 31, 32]. A translation algorithm has been implemented to translate OWLS processes into SHOP2 format.

The developed translation algorithm to translate OWLS atomic processes into SHOP2 operators is an extension of a sound and complete translation algorithm put forward by

Sirin *et al. [13]*. Unlike the algorithm proposed in *[13]*, the developed algorithm represents both inputs and preconditions of OWLS processes as SHOP2 preconditions. Thus, the inputs of OWLS processes are also used in planning.

The developed algorithm to translate composite and simple OWLS processes into SHOP2 format follows a very different approach as compared to the translation mechanism proposed in *[13]*. Sirin *et al. [13]* translated simple processes and composite processes directly into SHOP2 methods, while the developed algorithm decomposes the composite processes until they only contain atomic processes. Then, the atomic processes are all grouped together and translated into a single SHOP2 method.

In the developed algorithm, atomic processes are translated into SHOP2 operators. Simple processes and composite processes are decomposed until they contain only atomic processes which are subsequently translated into SHOP2 operators. The translated atomic processes are then grouped together in the form of an *if-then-else* method. The *if-then-else* method acts as the top-level composite process of the respective organisations. The purpose of planning is to create an execution path for this automatically generated top level composite process.

The implemented algorithm Translate-Atomic-Process(*Q*) to translate OWLS atomic processes into SHOP2 operators, is described below. It takes a definition *Q* of an atomic process *A* as input and outputs a SHOP2 operator *O*.

Translate-Atomic-Process(*Q*)

Let *Q* be the definition of an atomic process *A* and *O* be a SHOP2 operator
Pre = collection of all preconditions and inputs of *A* in *Q*
Add = the list of positive effects and outputs of *A* in *Q*
Del = collection of all negative effects of *A* in *Q*
Return *O* = (*A*($\vec{v}$) *Pre Del Add*)

End Translate-Atomic-Process

Translate-Atomic-Process(*Q*) translates an atomic process into a SHOP2 operator. It translates the

1) preconditions and inputs of the atomic process into the preconditions of the SHOP2 operator,

2) positive effects and outputs of the atomic process into positive post-conditions of the SHOP2 operator, and

3) negative effects and outputs of the atomic process into negative post-conditions of the SHOP2 operator.

The algorithm Translate-Composite-Process(Q) translates an OWLS composite process into a set of SHOP2 operators. It takes a definition Q of a composite process C as input and outputs a set L of SHOP2 operators. The algorithm is as follows.

Translate-Composite-Process(*Q*)

Let Q be the definition of a composite process C and O be a set of SHOP2 operators
($b_1, \ldots, b_n$) is the list of atomic and composite processes in C as defined in Q
for $i = 1, \ldots, n$
If b_i is an atomic process and q_i is the definition of b_i
O_0 = Translate-Atomic-Process(q_i)
Add O_0 into L
Else If b_i is a composite process and q_i is the definition of b_i
O = Translate-Composite-Process(q_i)
Add O into L
Else If b_i is a simple process and q_i is the definition of b_i
O = Translate-Simple-Process(q_i)
Add O into L
End If
End for
return L
End Translate-Composite-Process

Translate-Composite-Process(Q) translates a composite process into a set of SHOP2 operators. It calls Translate-Atomic-Process(q_i), if its component process is an atomic process, to translate the component atomic process into a SHOP2 operator. If its

component process is a composite or simple process, its recursively calls Translate-Composite-Process(q_i) or Translate-Simple-Process(q_i) to translate it into a set of SHOP2 operators.

The algorithm Translate-Simple-Process(Q) to translate OWLS simple processes into a set of SHOP2 operators is described below. It takes the definition Q of a simple process as input and outputs set L of SHOP2 operators. The algorithm is as follows.

Translate-Simple-Process(*Q*)

Let Q be the definition of a simple process S and L be an initially empty set of SHOP2 operators

($b_1, \ldots, b_n$) is the list of atomic and composite processes collapsing in S as defined in Q

for i = 1, . . . , n

If b_i is an atomic process and q_i is the definition of b_i

O_0 = Translate-Atomic-Process(q_i)

Add O_0 into L

If b_i is a composite process and q_i is the definition of b_i

O = Translate-Composite-Process(q_i)

Add O into L

End If

End for

return L

End Translate-Simple-Process

Translate-Simple-Process(Q) translates a simple process into a set of SHOP2 operators. It checks each of its constituent processes and

1. calls Translate-Atomic-Process(q_i) for each atomic process to translate it into a SHOP2 operator, and
2. calls Translate-Composite-Process(q_i) for each composite process to translate it into a set of SHOP2 operators.

The basic focus of the developed framework is to compose the atomic processes of the collaborating organisations into compatible workflows of OWLS services, capable of

achieving the desired goal states from the initial states, as defined by the collaborating organisations. Unlike [13, 31, 32], the developed framework is not focussed on finding an execution path for already defined composite processes. We believe that forming an execution path for an already built composite process limits the strength of workflow generation by limiting the automation in our scenario. Therefore, the composite processes are decomposed to atomic processes and then the atomic processes are used to create a single SHOP2 *if-then-else* method to guide the composition process. The user is encouraged to pass atomic OWLS process descriptions to the system.

The algorithm Create-BP (*O, G*) to create a recursive *if-then-else* SHOP2 method *BP* from the translated SHOP2 operators is as follows. We call the generated method *BP,* as it is a method to be added to the SHOP2 domain to represent the top-level business process of the respective organisation. Create-BP (*O, G*) takes a set *O* of SHOP2 operators and a set *G* of goals states as input and returns a SHOP2 *if-then-else* method *BP*.

Create-BP (*O, G*)

Let $O=\{O_1,O_2 \ldots O_m\}$ be the set of SHOP2 operators
G = conjunct of all goal states as specified by the organisation
Nil = empty task list
for i = 1, . . . , m
Pre_i = (conjunct of preconditions of O_i)
End for
Return $M = (BP() \; G \; Nil \; Pre_1 \; O_1 \; BP \; Pre_2 \; O_2 \; BP \; \ldots \; Pre_m \; O_m \; BP)$
End Create-BP

Create-BP(*O, G*) creates a recursive SHOP2 method, which groups the operators in an *if-then-else* format. An operator is executed when its preconditions hold. If all goal states in *G* are achieved, *Nil* is called to quit the method. As obvious in the expression $M = (BP() \; G \; Nil \; Pre_1 \; O_1 \; BP \; Pre_2 \; O_2 \; BP \; \ldots \; Pre_m \; O_m \; BP)$, the *BP* after every $Pre_i \; O_i$ makes this a recursive expression which will be called recursively until the goals states are achieved or the planners fails to find any valid plans.

The collection of OWLS processes passed to CWGM is treated as a top-level business process of the respective organisation and hence translated into a SHOP2 domain which

is used to find the execution plan for the top-level business process. The implemented algorithm Translate-OWLS-SHOP2(*P*) to translate a collection of OWLS processes into SHOP2 domain is as follows. It takes a collection *P* of OWLS processes and a set *G* of goals states as input, and creates a SHOP2 domain *D* as output.

Translate-OWLS-SHOP2 (*P, G*)

Let *P* be a collection of OWL-S processes, *K* be the set of definitions of OWLS processes in *P*, *G* be the conjunct of all goal states as specified by the organisation, *L* be a set of SHOP2 operators and *D* be a SHOP2 domain

Procedure:

D = Ø

For each atomic process definition *Q* in *K*

O$_0$ = Translate-Atomic-Process(*Q*)

add *O*$_0$ into *L*

End For each

For each simple process definition *Q* in *K*

O = Translate-Simple-Process(*Q*)

add *O* into *L*

End For each

For each composite process definition *Q* in *K*

O = Translate-Composite-Process(*Q*)

Add *O* into *L*

End For each

M = Create-BP (*L*,*G*)

Add *L* to *D*

Add *M* to *D*

Return *D*

End Translate-OWLS-SHOP2

The Translate-OWLS-SHOP2(*P*,*G*) works as follows.

1. It translates each of the constituent processes of *P* into SHOP2 operators by calling the relevant algorithms,

2. then it creates *an if-then-else* method *M* from the set *L* of SHOP2 operators and set *G* of goals states. *L* represents the set of all operators created by translating OWLS atomic processes, and set *G* represents the conjunct of all goal states as specified by the respective organisation.

3. then it adds the SHOP2 operators and SHOP2 method to the domain and returns the domain.

The algorithm Create-SHOP2-Problem(*K*) to generate a SHOP2 problem from the high level goals and initial state of the world as selected by the user is given as follows.

Create-SHOP2-Problem(*K,s,G*)

Let *K* be a collection of OWL-S processes, *D* be a SHOP2 domain created by translating *K*, *s* be the conjunct of initial states specified by the respective organisation and *G* is the conjunct of goal states specified by the respective organisation.
D = TRANSLATE-OWLS-SHOP2 (*K*,*G*)
return (*s*,*T*,*D*)
End Create-SHOP2-Problem

Where *T* is a task list containing book keeping operators and methods that keeps track of *G* and calls *BP* to do the actual planning. *BP* is the *if-then-else* method generated by the algorithm Create-BP (*O*, *G*) to represent the top level business process of the respective organisation.

6.4.3 Inserting SHOP2 Methods in the Domain

If two processes are fully or partially dependent on the same inputs or preconditions, they can be executed in parallel provided the inputs/preconditions of the processes represented as preconditions in SHOP2 hold in the current state of the world. Since SHOP2 does not support concurrency, therefore, such situation can create alternative composition paths. These alternative composition paths can lead to the generation of multiple sets of compatible workflows. The generation of multiple sets of compatible workflows enables the users to select from all possible available options, based on their constraints and preferences. It enables the users to select an alternative set of

compatible workflows for execution if the execution of the selected set fails. To ensure that the alternative composition paths are created as separate plans, SHOP2 methods are introduced into the domain. The implemented algorithm Insert-SHOP2-Methods(*K*) for introducing SHOP2 methods into the domain is as follows. It takes a set *K* of atomic processes which can be executed in parallel and a SHOP2 domain *D* as input, adds SHOP2 methods to the domain *D* and returns the modified domain.

Insert-SHOP2-Methods(*K, D*)

Let $Q=(Q_1,...Qx)$ be the definitions of a set of loaded atomic processes $K=(K_1,\ldots,K_x)$ that can be executed in parallel and *D* be the SHOP2 domain for the respective organisation.

$T= (T_1,....T_y)$ is the set of all possible and valid *ordered* task lists of component processes of *K*

for i = 1, . . . , y

Pre_i = conjunct of preconditions of the first task in T_i

$M_i = (K(\vec{v})\ Pre_i\ T_i)$

End for

$M = \{M_1,...,M_y\}$

add *M* to *D*

return *D*

End Insert-SHOP2-Methods

The outcome of this algorithm is a modified domain in which a SHOP2 method is inserted for every possible alternate composition path. Each method represents a valid and ordered sequence of tasks. The precondition for each method is the conjunct of preconditions of the first task in the respective *ordered* task lists. The set of all possible and valid ordered task lists of component processes of *B* is generated through an algorithm based on assigning levels to the activities based on their inputs. The Identify-Alternate-Paths(*Q*) algorithm creates all possible alternate composition paths from the list of atomic process definitions passed to it as input .

Identify-Alternate-Paths(*Q*)

Let *Q* be a collection of OWL-S atomic processes definitions

For each atomic process definition Q_o in *Q*

If (Q_o is dependent only on initial states)

Assign level 0 to Q_o

Else

Assign a level to Q_o = level of highest level process on which Q_o is dependent + 1

End If

End For each

For each two or more atomic processes having the same level

identify P = processes that can make part of alternate composition paths

create L = create all possible and valid composition orders

End For each

Return L

End Identify-Alternate-Paths

The algorithm works as follows.

1. Activities dependent on initial conditions are assigned level '0'. The level assigned to an activity is one more than the highest level activity on which it is dependent.

2. If two or more activities have the same levels, there can be alternative composition paths. All operators that can make part of the alternate composition paths are identified.

3. The identified operators are arranged to create a set of all possible and valid *ordered* task lists. This process is repeated every time there are two or more operators with the same level.

Figure 6.6 shows levels of the activities and alternative paths for the customer organisation discussed in Section 3.3. The collaboration example shown in Section 3.3 has further been used as a test case in Section 7.2 to automatically generate compatible workflows.

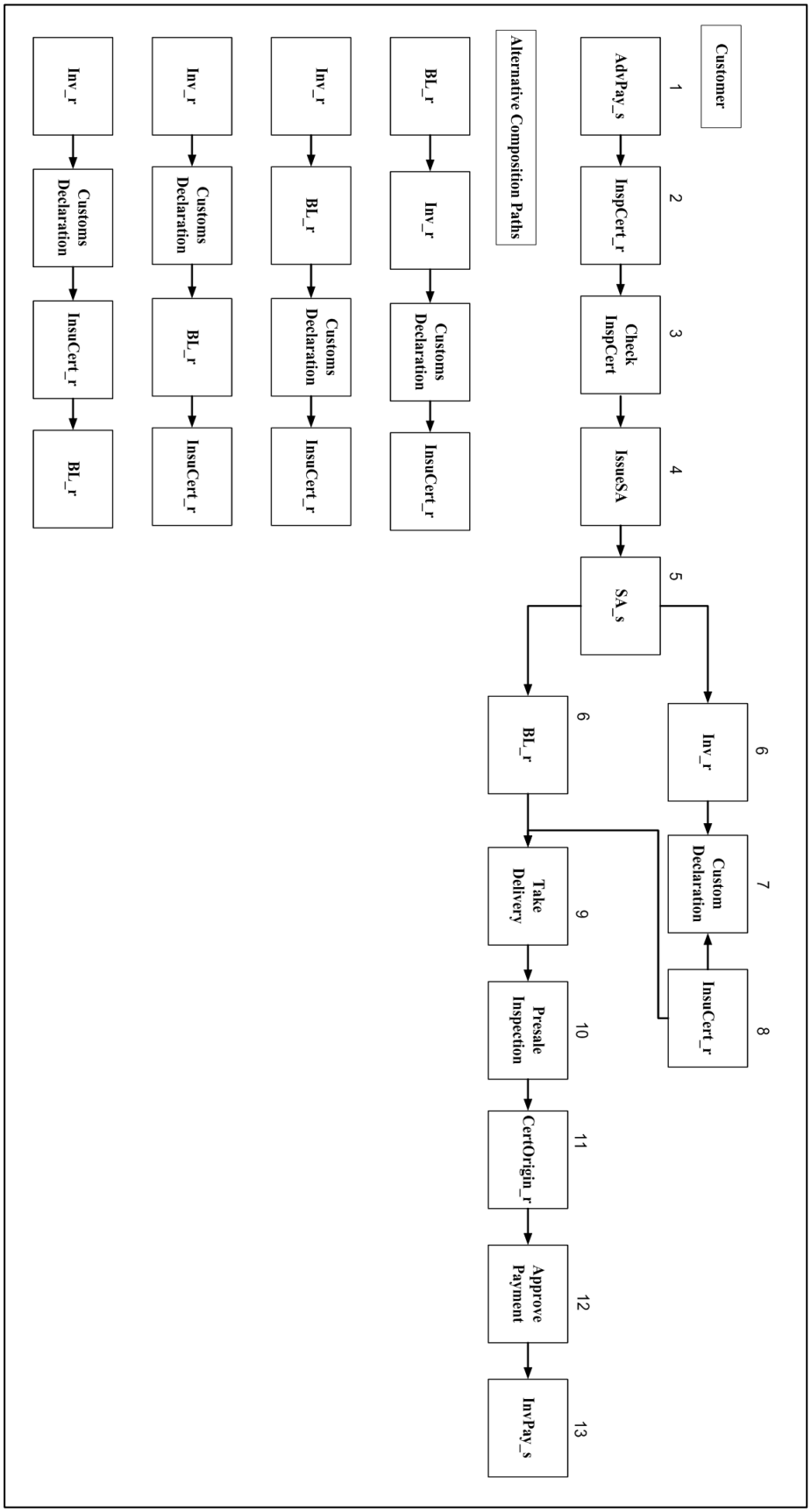

Figure 6.6 Level of Activities and Alternative Composition Paths for Customer

As shown in the figure, *Inv_r* and *BL_r* have the same level so alternate composition paths are possible. *Inv_r*, *BL_r*, *Customs Declaration* and *InsuCert_r* make part of alternate composition paths. *Inv_r*, *BL_r*, *Customs Declaration* and *InsuCert_r* activities can be arranged in four possible and valid orders. The parallel branches between *SA_s* and *Take Delivery* can be replaced by any of these sequential compositions of activities.

6.4.4 Creating a Joint Domain

The domain descriptions for all interacting organisations are collapsed in a single joint domain. This way all interacting organisations are considered part of a single organisational structure having cross organisational boundaries. The SHOP2 *BP* methods representing the top level business processes of each interacting organisation in an *if-then-else* format are joined together to create a single joint SHOP2 *JBP* method. The generated SHOP2 method represents the high level business process of the single organisational structure having cross organisational boundaries. The novel algorithm Create-Joint-SHOP2-Domain (*D*) to create a joint SHOP2 domain is given below. It takes a set *D* of SHOP2 domains of all collaborating organisations and produces a joint SHOP2 domain *JD*.

Create- Joint-SHOP2-Domain (*D*)

Let $\{Org_1, Org_2, ..., Org_m\}$ be the set of all collaborating organisations, $D = \{ D_1, D_2, ..., D_m \}$ is the set of domains of $\{Org_1, Org_2, ..., Org_m\}$ respectively and *JD* is a SHOP2 domain. Let *O* be an empty set of operators, *M* be an empty set of methods and *G* be an empty set of goal states.

$JD = Ø$

for i = 1, . . . , m

let O_i = set of operators in D_i

add O_i into *O*

let M_i = set of methods in D_i

add M_i into *M*

let G_i = conjunct goals of the Org_i

add G_i into *G*

End for

Add *O* to *JD*

Add *M* to *JD*

JBP = Create-JBP(*G*, *JD*)

Add *JBP* into *JD*

return *JD*

End Create- Joint-SHOP2-Domain

Create-Joint-SHOP2-Domain(*D*) combines the operators and methods of the collaborating domains and merges them into the joint domain. It then merges the *BP* methods of all collaborating organisations into a single joint *BP* (*JBP*) method which acts as a joint high level business process of the single organisation created by combining all collaborating organisations. The developed novel algorithm CREATE-JBP(*G*, *JD*) to create the *JBP* method is as follows. JBP(*G*, *JD*) takes a joint domain *JD* and a set *G* of goal states to achieve as input and creates a SHOP2 method *JBP* which acts as the top level business process of the joint organisation.

Create-JBP(*G*, *JD*)

Let *G* = conjunct of goals states of all collaborating organisations and *JD* be the SHOP2 joint domain.

$O = \{O_1, O_2, \ldots O_m\}$ be the set of operators in *JD*

$Pre_o = \{Pre_{o1}, Pre_{o2}, \ldots Pre_{om}\}$ be the set of conjuncts of preconditions of $\{O_1, O_2, \ldots O_m\}$ respectively

$M = \{M_1, M_2, \ldots M_n\}$be the set of methods in *JD*

$\{Pre_{m1}, Pre_{m2}, \ldots Pre_{mn}\}$ be the set of conjuncts of preconditions of $\{M_1, M_2, \ldots M_n\}$ respectively

Nil = empty task list

Return *JBP* = (*JBP*() *G Nil* $Pre_{o1}\ O_1$ *JBP* $Pre_{o2}\ O_2$ *JBP*…$Pre_{om}\ O_m$ *JBP* $Pre_{m1}\ M_1$ *JBP* $Pre_{m2}\ M_2$ *JBP*… $Pre_{mn}\ M_n$ *JBP*)

End Create-JBP

Create-JBP(*G*, *JD*) creates a recursive SHOP2 method *JBP*, which groups the operators and methods of all collaborating organisations in an *if-then-else* format. An operator is executed or a method is decomposed when its preconditions hold. If all goal states in *G* are achieved, *Nil* is called to quit the method. In the expression *JBP* = (*JBP*() *G Nil*

$Pre_{o1}\ O_1\ JBP\ Pre_{o2}\ O_2\ JBP \ldots Pre_{om}\ O_m\ JBP\ Pre_{m1}\ M_1\ JBP\ Pre_{m2}\ M_2\ JBP \ldots\ Pre_{mn}\ M_n\ JBP$), calling *JBP* after every $Pre_{oi}\ O_i$ and every $Pre_{mi}\ M_i$ makes it a recursive expression and *JBP* will be called by the planner recursively until valid plans are found or the SHOP2 returns a failure.

The implemented algorithm Create-Joint-SHOP2-Problem(*P*,*D*,*s*,*G*) for generating a joint SHOP2 problem by combining the SHOP2 problems of the collaborating organisations is as follows. It takes a set *P* of SHOP2 problems, set *D* of SHOP2 domains, set *s* of initial states and set *G* of goals states of the collaborating organisations as inputs and returns a joint SHOP2 problem.

Create-Joint-SHOP2-Problem (*P*,*D*,*s*,*G*)

> Let $P=\{P_1,P_2,...,P_m\}$ be the set of SHOP2 problems of the collaborating organisations, $D = \{ D_1, D_2, \ldots, D_m \}$is the set of domains of the collaborating organisations, *JD* is the joint SHOP2 domain, *s* = conjunct of initial states of all collaborating organisations and *G* = conjunct of goals states of all collaborating organisations.
>
> JD = Create-Joint-SHOP2-Domain(D)
>
> return (*s*, *T*, *JD*).

End Create-Joint-SHOP2-Problem

Where *T* is the task list containing book keeping operators and methods that keeps track of *G* and calls *JBP* in *JD* to do the actual planning to achieve *G*. *JBP* is the method generated by collapsing *BP* methods of all interacting organisations into a single joint method. SHOP2 takes (*s*, *T*, *JD*) as input to start the planning process to achieve *T* from *s* in *JD*.

6.4.5 Planning for All Possible Sets of Compatible Plans

The modified SHOP2 planner takes the joint SHOP2 problem (*s*, *T*, *JD*) as input and creates $P = (P_1\ P_2 \ldots P_n)$ as a set of multiple valid plans. Every plan P_i in *P* is a sequence of instantiated operators $(O_1,O_2,\ldots,O_m)$ that will achieve *T* from *s* in *JD*. All plans in *P* are joint plans. The joint plans are divided into sub-plans, one for each organisation, compatible with each other. The division is based on the prefix attached to each operator after reading the OWLS process definitions. Operators with the same prefix

are added into the plan for the organisation represented by the "*Org + Organisation Number"*. The control dependencies and data dependencies are kept the same as in joint plans. The set of compatible plans with least number of operators is highlighted to the users for execution. Assuming each operator takes the same time, this would be the least cost heuristic. The users can select the highlighted set or any other set of compatible plans for execution.

The compatibility of the set of compatible plans generated by the division of a joint plan is intuitive. In the joint plan, the compatible plans for all collaborating organisations are arranged together in a particular order which ensures the achievement of the goal states of all collaborating organisations. This means there is an agreed sequence of activities that can ensure the achievement of the goals of every collaborating organisation, which is the definition of compatibility [3].

6.4.6 Runtime Execution and Collaboration

The developed framework provides runtime support for the generated sets of compatible workflows. The selected set of compatible plans is passed to the SHOP2toOWLS translator which converts it into enactable workflows of OWLS atomic processes. At runtime, the control and data dependency among the activities in the set of compatible workflows is followed as specified in the joint workflow that was sub-divided to create the selected set of compatible workflows. Since each activity in the selected set of compatible workflows is basically an OWLS atomic process which is a model of an actual WSDL service, the activity can be enacted directly using the enactment mechanism of the OWLS API. The enactment of an atomic process is a call to the corresponding web accessible program with its inputs instantiated. The runtime enactment manager keeps track of generated outputs in the form of a name-value pair in a hashmap, so that the generated outputs can be passed as inputs to the corresponding processes.

Since workflows of more than one organisations are enacted together, they should be collaborated in order to ensure a smooth transfer of data and information among the cross organisational activities. Exchange of information is carried out between collaborating organisations through interface activities i.e. sending and receiving activities. Every sending activity uploads the sending information to a central server

that has been set up to coordinate the collaboration process. Each collaborating organisation is provided with the access permissions and details of the central server. The uploaded information can be downloaded by the relevant collaborating organisation whenever a receiving activity is encountered in that organisation. The control flow and data flow of cross organisational activities as specified in the joint workflow is followed to ensure that the execution of collaborating workflows can go on hand in hand to completion.

6.5 Implementation

A proof-of-concept prototype has been implemented for the proposed framework. Collaborating organisations load their OWLS process definitions to the GUI of the prototype. The GUI was developed using Swing and AWT classes of Java. OWLS process definitions can be created manually or automatically using OWLS editor of Protégé. WSDL2OWL-S tool can also be used for automatic generation of OWLS processes from WSDL descriptions. We use a modified form of WSDL2OWL-S tool to develop OWLS processes for the example scenarios. WSDL descriptions of the web services are automatically generated from the Java code of the services with the help of Apache Axis2.

The OWLSReader is a Java module based on OWLS API which is a Java based API for programmatic access to read, execute and write OWLS service descriptions. OWLSReader is a Java program which is capable of reading OWLS processes and loading the processes in form of Java objects to CWGM. The loaded OWLS processes are translated into HTN format using OWLStoSHOP2 Translator. OWLStoSHOP2 Translator is a Java module for translating the loaded OWLS processes into HTN format.

CWGM is the most fundamental module in the system. It manages the entire lifecycle of the workflow generation, workflow execution and workflow collaboration process. CWGM is developed using Java. The Planning is done using a modified version of JSHOP2 planner. JSHOP2 is the Java implementation of SHOP2 planner.

SHOP2toOWLS Translator is a Java program to transform the SHOP2 plans into workflows of OWLS processes which can be enacted directly. The Runtime Execution

Manager is a Java module to enact the OWLS processes in the generated workflows. The enactment is based on the execution mechanism of the OWLS API. A process is enacted by calling the corresponding web accessible program which the process models, with its inputs instantiated. The runtime collaboration is also done by Runtime Execution Manager. We use Jsch API to upload and download files over Secure File Transfer Protocol (SFTP). Jsch is the Java implementation of SSH2.

6.6 Discussion

The framework presented in the thesis is closely related to the system proposed by Sirin *et al.* [13]. Both the systems

1. perceive automatic workflow generation as an AI planning problem, and exploit web services composition for automatic workflow generation based on the similarity between the two,
2. use SHOP2 planner for automatic workflow generation, and
3. translate OWLS process descriptions into SHOP2 methods to create domain control knowledge.

The work presented in this thesis extends the application of AI planning to workflow generation as well as workflow collaboration. Below are some of the major extensions and improvements the developed framework has made to the approach taken by Sirin *et al.* [13] for workflow generation.

1. Their system considers automatic workflow generation for a single organisation only. They do not focus on workflow collaboration among business organisations. The implemented framework integrates automatic workflow generation with cross organisational workflow collaboration and is capable of generating multiple sets of compatible workflows for multiple collaborating organisations. Similarly, collaboration is also supported at runtime. Their system lacks this functionality.
2. They limit a service to either have outputs or effects. In real world a service can have effects and outputs at the same time. The framework presented in this thesis does not have this limitation.

3. Similarly, their system executes information providing services (services with only outputs) at planning time to produce the required information. The developed framework does not execute web services at planning time. This is because a service can have both effects and outputs and executing a web service at planning time can have real effects on the world e.g. charging the credit card for a certain amount of money.

4. They look at web service composition as finding an execution path for already defined composite processes, which limits the automation of workflow generation by involving users to define composite processes. The developed framework presented in this thesis looks at web service composition as automatically generating a composite process from the atomic processes and then specialising it to create an execution path for the composite process. The OWLS to SHOP2 translation mechanism of both systems are hugely different due to this reason. They translate the composite processes directly into SHOP2 methods. The developed framework decomposes the composite processes until they only contain atomic processes, translate the atomic processes into SHOP2 operators and then group the operators as an *if-then-else* method.

5. The system presented by Sirin *et al.* creates a single plan, based on the constraints of the user. If the user rejects the plan then the system re-plans for another plan. The developed framework creates all possible set of workflows. This enables the users to select from the generated set of workflows on the basis of their preferences. Also, the developed framework inserts methods in the domain to enable the planner to identify alternative composition paths whenever there are operators that can be executed concurrently. The framework reported by Sirin *et al.* lacks this functionality.

6.7 Conclusion

A framework based on the integration of automatic workflow generation, build time workflow collaboration, workflow enactment and runtime workflow collaboration is presented in this chapter. The basic aim of the presented framework is to create compatible and enactable workflows for multiple collaborating organisations, from their OWLS process definitions and high level goals. The focus is to compose atomic

processes into composite processes rather than forming an execution path for composite processes, which we believe limits the true strength of workflow generation by limiting automation.

The developed framework is the only framework so far which automatically generates compatible workflows for multiple collaborating organisations. The presented framework builds upon the system presented by Sirin *et al.* [13] and extends its functionality in several ways. The next chapter uses two example scenarios of multi-organisational business collaboration to demonstrate the functionality and benefits of the implemented framework.

Chapter 7: Results and Evaluation

7.1 Introduction

This chapter presents the results and evaluation of the developed framework presented in Chapter 6. Two example scenarios of multiple business organisation collaboration are used. The details of the OWLS processes of the collaborating organisations are given and the sets of compatible workflows generated on the basis of their OWLS processes and initial states to achieve their desired goal states are presented. The chapter also explains the execution of the generated workflows. The scalability, efficiency and viability of the presented framework are also discussed.

7.2 Vendor/Customer Business Collaboration Scenario

To illustrate the functionality of the presented framework, the collaboration scenario discussed in Section 3.3 is used. *Vendor* and *Customer* represent the vendor and customer organisations as explained in the example given in Section 3.3.The following sub-sections present the details of their OWLS processes and the compatible workflows generated.

7.2.1 OWLS Processes

The OWLS process descriptions for *Vendor* and *Customer* relevant to the scenario are given in Table 7.1 and 7.2 respectively. The *Vendor and Customer* can have any number of OWLS processes and the presented framework will filter out any that are not relevant to a given application scenario. The OWLS processes simulate the actual activities of the collaborating organisations. Each activity is represented as an OWLS process which is grounded in an actual WSDL service. Appendices C and D give the OWLS and WSDL descriptions and Appendix E gives the Java code for a sample OWLS process *IssueInspCert* which simulates the creation of an inspection certificate.

Table 7.1 *Vendor's* OWLS Processes

S. No	Process Details
1	Name: *AdvPay_r* (Receive advance payment)

	Inputs/ Preconditions: *s_Payment* (Advance payment sent by *Customer*) Outputs/ Effects: *r_Payment* (Advance payment received from *Customer*) Description: This process receives advance payment from the *Customer*.
2	Name: *PaymentCheck* (Check payment) Inputs/ Preconditions: *r_Payment* (Advance payment received from *Customer*) Outputs/ Effects: *ok_PC* (Payment Check OK) Description: This process checks the advance payment received from the *Customer*.
3	Name: *GoodsManufacture* (Manufacture Goods) Inputs/ Preconditions: *ok_PC* (Payment check OK) Outputs/ Effects: *goods* (Manufactured goods) Description: This process manufactures goods.
4	Name: *IssueInv* (Issue commercial invoice) Inputs/ Preconditions: *goods* (Manufactured Goods) Outputs/ Effects: *Invoice* (Commercial Invoice) Description: This process issues a commercial invoice.
5	Name: *FactoryInspection* (Inspect manufactured goods) Inputs/ Preconditions: *Invoice* (Commercial Invoice) Outputs/ Effects: *ok_Insp* (Factory Inspection OK) Description: This process inspects the manufactured goods.
6	Name: *IssueInspCert* (Issue inspection certificate) Inputs/ Preconditions: *ok_Insp* (Factory Inspection OK) Outputs/ Effects: *InspCert* (Inspection certificate) Description: This process issues an inspection certificate.
7	Name: *InspCert_s* (Send inspection certificate) Inputs/ Preconditions: *InspCert* (Inspection certificate) Outputs/ Effects: *s_InspCert* (Inspection certificate sent) Description: This process sends the inspection certificate to the *Customer*.
8	Name: *SA_r* (Receive shipment arrangement notification) Inputs/ Preconditions: *s_SA* (Shipment arrangement notification sent by

	Customer) *s_InspCert* (Inspection certificate sent) Outputs/ Effects: *r_SA* (Shipment arrangement notification received from *Customer*) Description: This process receives the shipment arrangement notification.
9	Name: *Inv_s* (Send commercial invoice) Inputs/ Preconditions: *r_SA* (Shipment arrangement notification received) *Invoice* (Commercial Invoice) Outputs/ Effects: *s_Inv* (Commercial invoice sent) Description: This process sends the commercial invoice to the *Customer*.
10	Name: *ShippingArrangement* (Arrange Shipment) Inputs/ Preconditions: *r_SA* (Shipment arrangement notification received from *Customer*) *Invoice* (Commercial Invoice) Outputs/ Effects: *BL* (Bill of lading) Description: This process arranges shipment of goods.
11	Name: *InsuranceArrangement* (Arrange insurance) Inputs/ Preconditions: *s_BL* (Bill of lading sent) *Invoice* (Commercial Invoice) Outputs/ Effects: *InsuCert* (Insurance certificate) Description: This process arranges the insurance of the goods.
12	Name: *InsuCert_s* (Send insurance certificate) Inputs/ Preconditions: *InsuCert* (Insurance certificate) Outputs/ Effects: *s_InsuCert* (Insurance certificate sent) Description: This process sends the insurance certificate to the *Customer*.
13	Name: *BL_s* (Send bill of lading) Inputs/ Preconditions: *BL* (Bill of lading) Outputs/ Effects: *s_BL* (Bill of lading sent) Description: This process sends the bill of lading to the customer.
14	Name: *CertOriginApp* (Apply for certificate of origin) Inputs/ Preconditions: *s_Inv* (Commercial invoice sent) *s_InsuCert* (Insurance certificate sent)

	Invoice (Commercial Invoice) Outputs/ Effects: *OrigCert* (Certificate of origin) Description: This process applies for certificate of origin.
15	Name: *CertOrigin_s* (Send certificate of origin) Inputs/ Preconditions: *OrigCert* (Certificate of origin) Outputs/ Effects: *s_OrigCert* (Certificate of origin sent) Description: This process sends the certificate of origin to the *Customer*.
16	Name: *InvPay_r* (Receive payment for invoice) Inputs/ Preconditions: *s_OrigCert* (Certificate of origin sent) *s_InvPay* (Payment for the invoice sent by *Customer*) Outputs/ Effects: *r_InvPay* (Payment for the invoice received from *Customer*) Description: This process receives the payment for the invoice from the *Customer*.
17	Name: *PaymentHandling* (Handle payment) Inputs/ Preconditions: *r_InvPay* (Payment for the invoice received from *Customer*) Outputs/ Effects: *ok_PH* (Payment handling OK) Description: This process handles payment.

Table 7.2 *Customer's* OWLS Processes

S. No	**Process Details**
1	Name: *AdvPay_s* (Send advance payment) Inputs/ Preconditions: *Payment* (Advance payment) Outputs/ Effects: *s_Payment* (Advance payment sent) Description: This process sends advance payment to the *Vendor*.
2	Name: *InspCert_r* (Receive inspection certificate) Inputs/ Preconditions: *s_Payment* (Advance payment sent) *s_InspCert* (Inspection certificate sent by *Vendor*) Outputs/ Effects: *r_InspCert* (Inspection certificate received from *Vendor*) Description: This process receives the inspection certificate from the *Vendor*.

3	Name: *CheckInspCert* (Check inspection certificate) Inputs/ Preconditions: *r_InspCert* (Inspection certificate received from *Vendor*) Outputs/ Effects: *ok_InspCert* (Inspection certificate OK) Description: This process checks the inspection certificate received from the *Vendor*.
4	Name: *IssueSA* (Issue shipment arrangement notification) Inputs/ Preconditions: *ok_InspCert* (Inspection certificate OK) Outputs/ Effects: *SA* (Shipment arrangement notification) Description: This process issues the shipment arrangement notification.
5	Name: *SA_s* (Send shipment arrangement notification) Inputs/ Preconditions: *SA* (Shipment arrangement notification) Outputs/ Effects: *s_SA* (Shipment arrangement notification sent) Description: This process sends the shipment arrangement notification to the *Vendor*.
6	Name: *BL_r* (Receive bill of lading) Inputs/ Preconditions: *s_SA* (*S*hipment arrangement notification sent) *s_BL* (Bill of lading sent by the *Vendor*) Outputs/ Effects: *r_BL* (Received bill of lading from *Vendor*) Description: This process receives the bill of lading from the *Vendor*.
7	Name: *Inv_r* (Receive commercial invoice) Inputs/ Preconditions: *s_SA* (Shipment arrangement notification sent) *s_Inv* (Commercial invoice sent by *Vendor*) Outputs/ Effects: *r_Inv* (Commercial invoice received from *Vendor*) Description: This process receives the commercial invoice from the *Vendor*.
8	Name: *CustomsDeclaration* (Declare goods to customs) Inputs/ Preconditions: *r_Inv* (Commercial invoice received from *Vendor*) Outputs/ Effects: *CD* (Customs declaration report) Description: This process declares the delivered goods to customs.
9	Name: *InsuCert_r* (*Receive* insurance certificate) Inputs/ Preconditions: *CD* (Customs declaration report) *s_InsuCert* (Insurance certificate sent by *Vendor*)

	Outputs/ Effects: *r_InsuCert* (Insurance certificate received from *Vendor*) Description: This process receives the Insurance certificate from the *Vendor*.
10	Name: *TakeDelivery* (Take Delivery) Inputs/ Preconditions: *r_InsuCert* (Insurance certificate received from *Vendor*) *r_Inv* (Payment for invoice received from *Vendor*) *r_BL* (Bill of lading received from *Vendor*) Outputs/ Effects: *Delivery* (Goods Delivered) Description: This process takes delivery of the goods.
11	Name: *PresaleInspection* (Presale inspection of goods) Inputs/ Preconditions: *Delivery* (Goods delivered) Outputs/ Effects: *ok_PI* (Presale inspection OK) Description: This process inspects the goods after the delivery is taken.
12	Name: *CertOrigin_r* (Receive the certificate of origin) Inputs/ Preconditions: *ok_PI* (Presale inspection OK) *s_OrigCert* (Certificate of origin sent by *Vendor*) Outputs/ Effects: *r_ OrigCert* (Certificate of origin received from *Vendor*) Description: This process receives the certificate of origin from the *Vendor*.
13	Name: *ApprovePayment* (Approve Payment) Inputs/ Preconditions: *r_ OrigCert* (Certificate of origin received from *Vendor*) *r_Inv* (Commercial invoice received from *Vendor*) Outputs/ Effects: *InvPay* (Payment for invoice) Description: This process approves payment to the *Vendor*.
14	Name: *InvPay_s* (Send payment for invoice) Inputs/ Preconditions: *InvPay* (Payment for the invoice) Outputs/ Effects: *s_InvPay* (Payment for the invoice sent) Description: This process sends payment for the invoice to the *Vendor*.

7.2.2 Results

Based on the OWLS processes given in Table 7.1 and Table 7.2 the system generates 20 sets of compatible workflows. The generation of the 20 sets of compatible workflows is due to the identification of different composition paths when the planner encounters activities that can be executed in parallel. The system takes 6816 milliseconds to generate the 20 sets of compatible workflows.

Figures 7.1 and 7.2 show two different sets of compatible workflows for *Vendor* and *Customer* Collaboration. The remaining sets are given in Appendix F. The graphical representation of the workflows is used to make them more understandable. The solid lines show control dependencies while the dotted lines show data dependencies. Figures 7.1 and 7.2 show that the data dependencies are the same in both set of plans but the control dependencies are different. In the *Vendor*'*s* workflow in Figure 7.1, *ShippingArrangement* has a control dependency on *SA_r*, and *Inv_s* has control dependency on *InsuCert_s*. In the *Vendor*'*s* workflow in Figure 7.2, *Inv_s* has a control dependency on *SA_r* and *ShippingArrangement* has control dependency on *Inv_s*. Similarly, in the *Customer*'*s* workflow in Figure 7.1, *Bl_r* has a control dependency on *SA_s* and *Inv_r* has a control dependency on *BL_r*. In the *Customer*'s workflow in Figure 7.2, *Inv_r* has a control dependency on *SA_s* and *BL_r* has control dependency on *CustomsDeclaration.* Both sets of workflows are valid and compatible.

The compatibility of the workflows can be verified by considering their respective interface processes. Figure 7.3 shows the interface process for the workflows in Figure 7.1. The corresponding interface activities have been labelled with the same alphabet to make them clearer to follow. It can be observed that for every receiving activity there is a corresponding sending activity. Notice that *Inv_r* has to wait for *InsuCert_s* to complete before *Inv_s* to complete, so there is a delay of one activity. But there is no deadlock so the interface processes of both workflows are compatible.

In the workflows presented in figure 7.1 and 7.2, the initial state is *payment,* which means that the customer has to make an advance payment to start the collaboration process. The final goal for *Vendor* is *ok_PH,* which means that payment should be successfully handled. The final goal for *Customer* is *s_InvPay,* which means that the payment for the invoice is sent to the *Vendor* at the end of its business process.

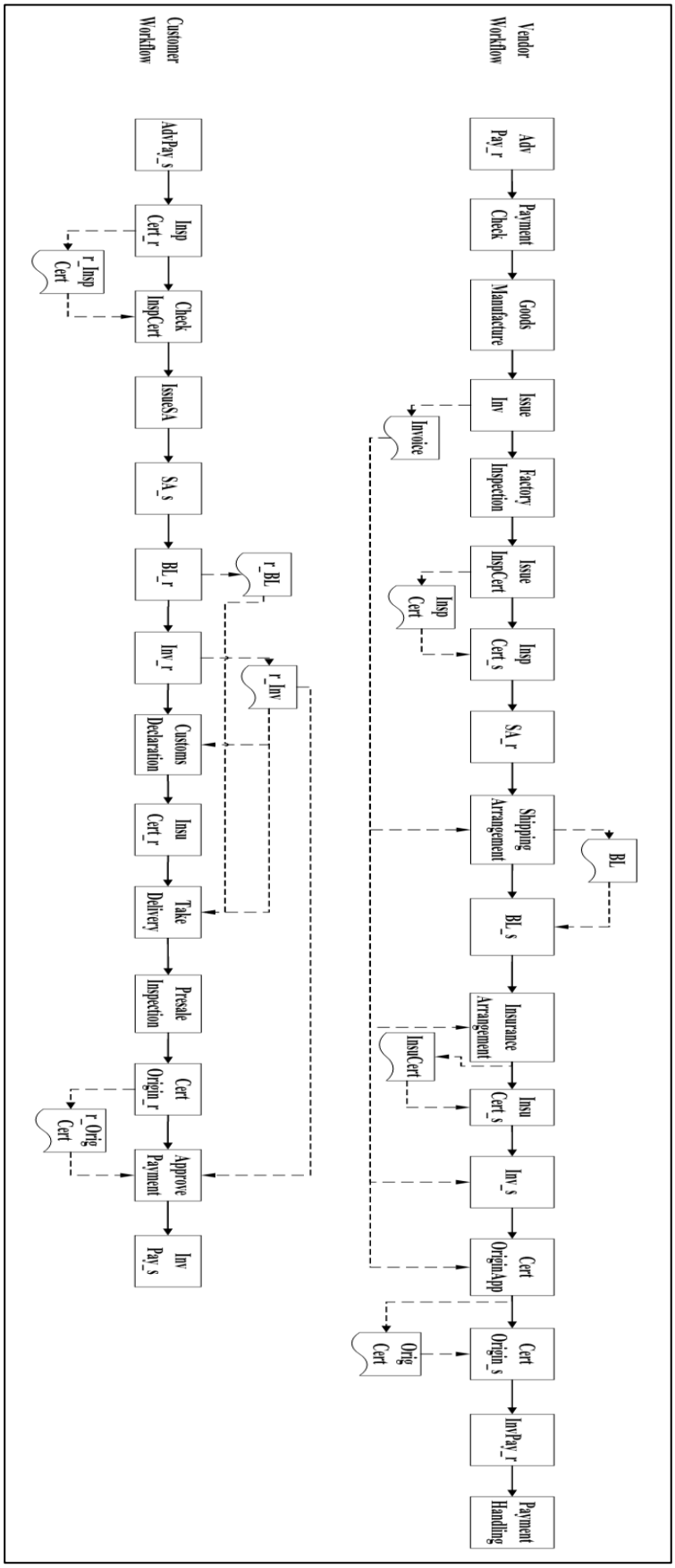

Figure 7.1 A Set of Compatible Workflows for *Vendor* and *Customer*

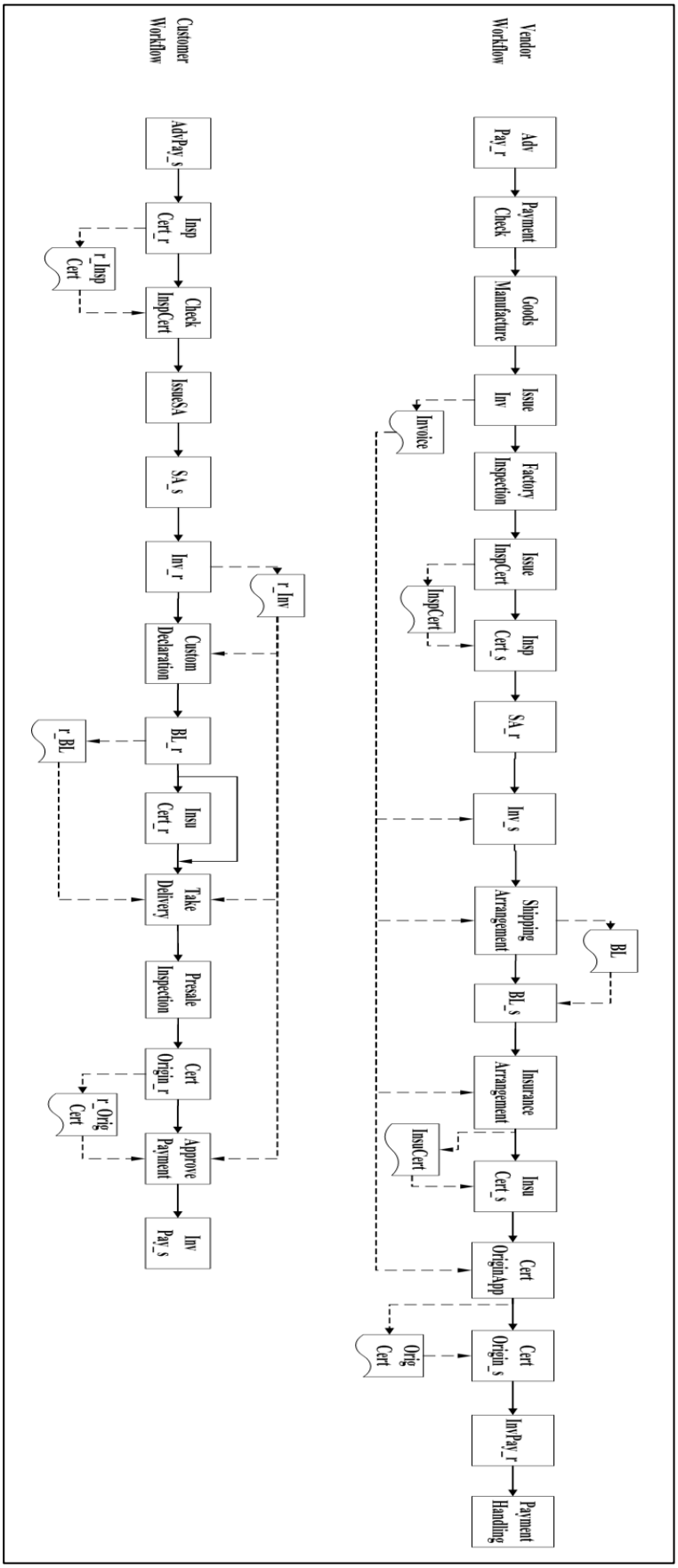

Figure 7.2 Another Set of Compatible Workflows for *Vendor* and *Customer*

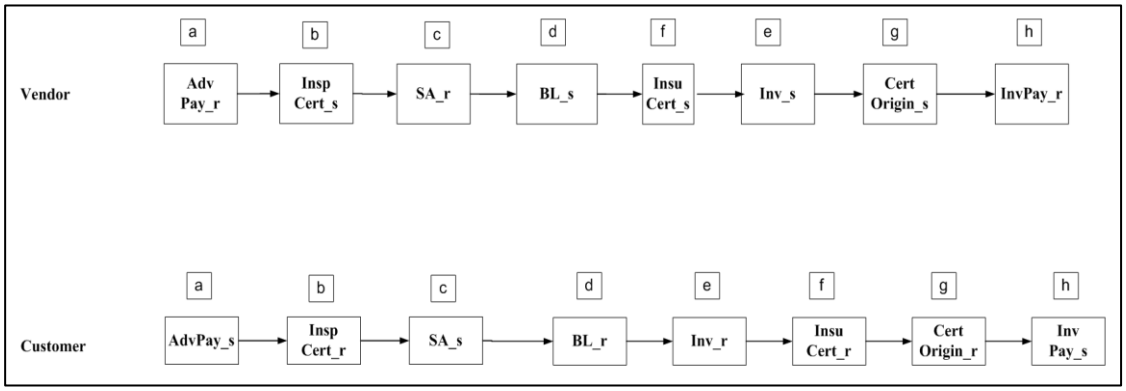

Figure 7.3 Interface Processes for *Vendor*/*Customer* Workflows in Figure 7.1

After workflow generation, the user selects one from the sets of compatible workflows for execution. The sequential order of the activities specified by the control dependencies must be followed at runtime, e.g. *AdvPay_r* must be executed before *PaymentCheck*. Similarly, the data dependencies must also be followed at runtime. For example, *Shipping Arrangement* activity must be executed after *IssueInv* in both sets of compatible workflows, since *Shipping Arrangement* needs commercial invoice (*Invoice*) which is generated by *IssueInv*.

For cross organisational activities, the sending activities upload the data to a central server which is downloaded by the receiving services. For example, in Figure 7.1, *InspCert_s* is a sending service which uploads inspection certificate to a central server and *InspCert_r* is a receiving activity which downloads the inspection certificate. The complete execution of the compatible workflows achieves the desired goals.

The developed framework generates workflows that only consist of necessary OWLS processes to achieve the goals states from initial states. To illustrate this, a scenario where the collaborating organisations select *goods* and *SA* as the initial states and *s_OrigCert* and *r_OrigCert* as the goals states is presented. This means that in the initial state of the world the goods are already manufactured by the *Vendor* and shipment arrangement notice is already issued by the *Customer*. The desired goal is to achieve a world state in which the *Vendor* sends the certificate of origin to the *Customer* and the *Customer* receives the certificate of origin form the *Vendor*. In this scenario several of the OWLS processes in Table 7.1 and 7.2 become irrelevant to the workflow generation process as they are not required in the workflows to achieve the goal states from the initial states. The framework discards the irrelevant processes from

the set of loaded processes to make sure that they are not translated to SHOP2 format or used in the planning. This approach saves time.

The system takes 5794 milliseconds to generate the 20 sets of compatible workflows. Figure 7.4 and 7.5 show two of the 20 sets of compatible workflows generated. It is obvious that the workflows only consist of OWLS processes necessary to achieve the goal states from the initial states. In both figures the data dependencies are the same while the sequential control dependencies are different. In the *Vendor's* workflow in Figure 7.4, *ShippingArrangement* has a control dependency on *SA_r* and *Inv_s* has control dependency on *BL_s*. In the *Vendor's* workflow in Figure 7.5, *Inv_s* has a control dependency on *SA_r* and *ShippingArrangement* has control dependency on *Inv_s*. Similarly, in the *Customer's* workflow in Figure 7.4, *CustomsDeclaration* has a control dependency on *Inv_r* and *Bl_r* has a control dependency on *CustomsDeclaration.* In the *Customer*'s workflow in Figure 7.5, *CustomsDeclaration* has a control dependency on *BL_r* and *BL_r* has control dependency on *Inv_r*. At runtime, the users are asked to provide the values for the initial inputs and the OWLS processes in the generated workflows are executed to achieve the desired goals.

Although the length of the generated workflows is different to the scenario shown in Figure 7.1 and 7.2, but the number of generated sets of compatible workflows is similar i.e. twenty sets of compatible workflows are generated. This is because the OWLS processes that can be executed concurrently are the same in the new scenario as well. The concurrent OWLS processes are ordered in alternative composition orders to generate multiple sets of compatible workflows. This is done by the methods insertion algorithm explained in Section 6.4.3. As shown for the *Customer* in Figure 6.6, *Inv_r, BL_r, Customs Declaration* and *InsuCert_r* occur in parallel branches of the workflow and can be arranged in four possible and valid orders. Similarly, in the *Vendor's* OWLS process given in Table 7.1, *Inv_s*, *ShippingArrangement*, *BL_s*, *InsuranceArrangement* and *InsuCert_s* occur in parallel branches of the workflow and can be arranged in five alternate orders. This means that twenty possible joint plans can be generated from the OWLS processes of both collaborating organisations and hence twenty possible sets of compatible workflows can be generated. If the concurrency is removed from the *Customer's* OWLS processes in Table 7.2 by making the changes shown in Table 7.3, then only five sets of compatible workflows can be generated. This is because *BL_r*

must now occur before *Inv_r* in the *Customer's* workflow and so there is only one possible sequential order of processes for the *Customer* and five possible orders of processes for the *Vendor*. Figure 7.6 and Figure 7.7 show two of the five sets of compatible workflows.

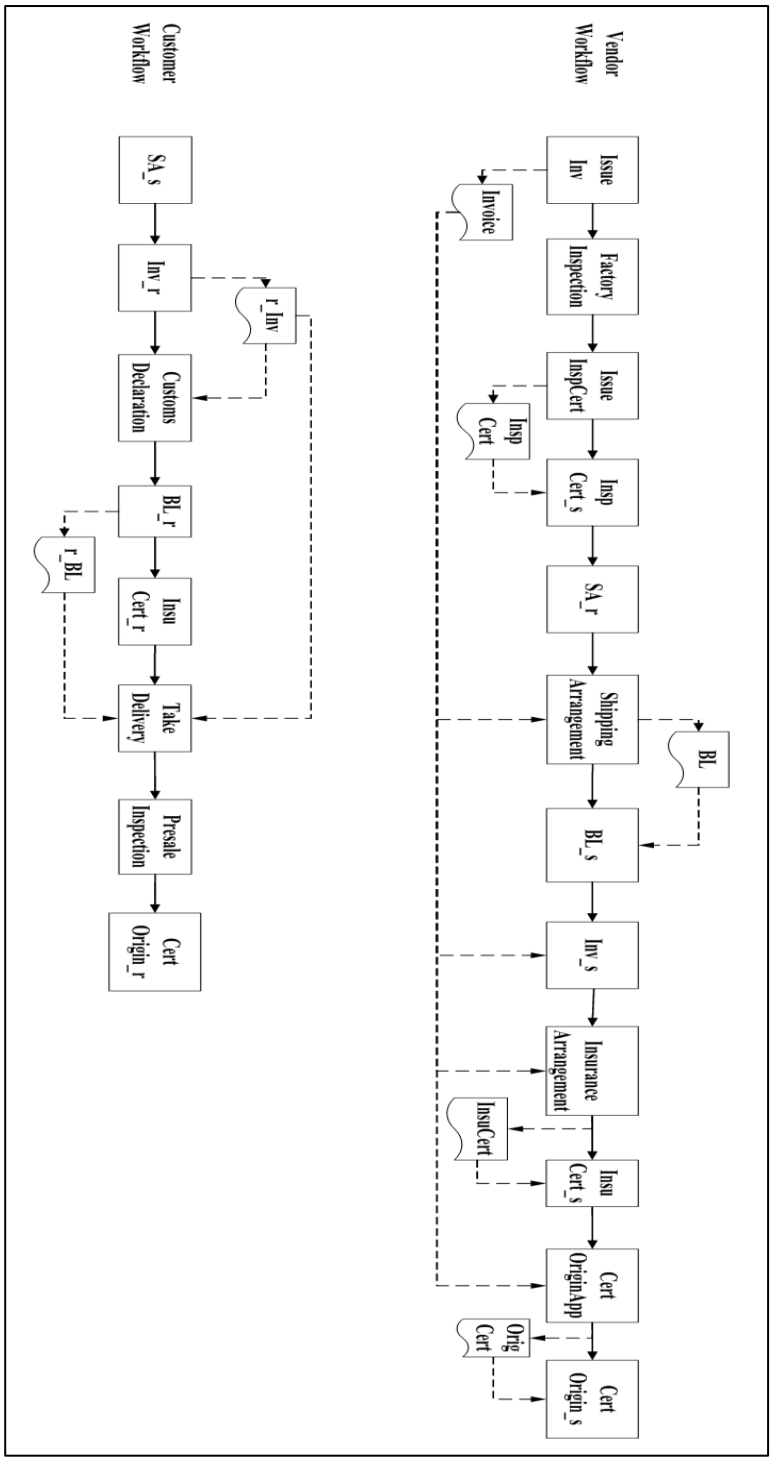

Figure 7.4 An Alternative Length Set of Compatible Workflows for *Vendor* and *Customer*

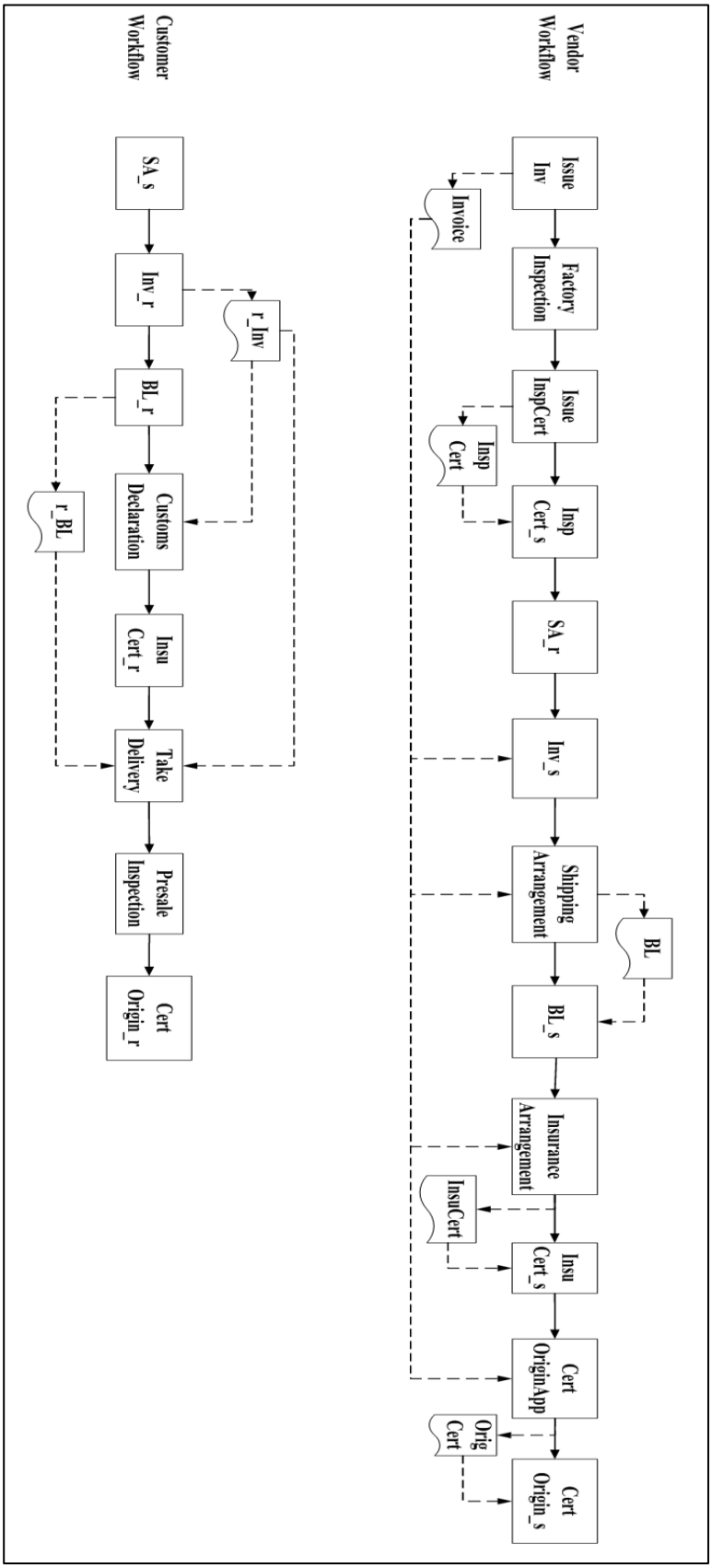

Figure 7.5 Another Alternative Length Set of Compatible Workflows for *Vendor* and *Customer*

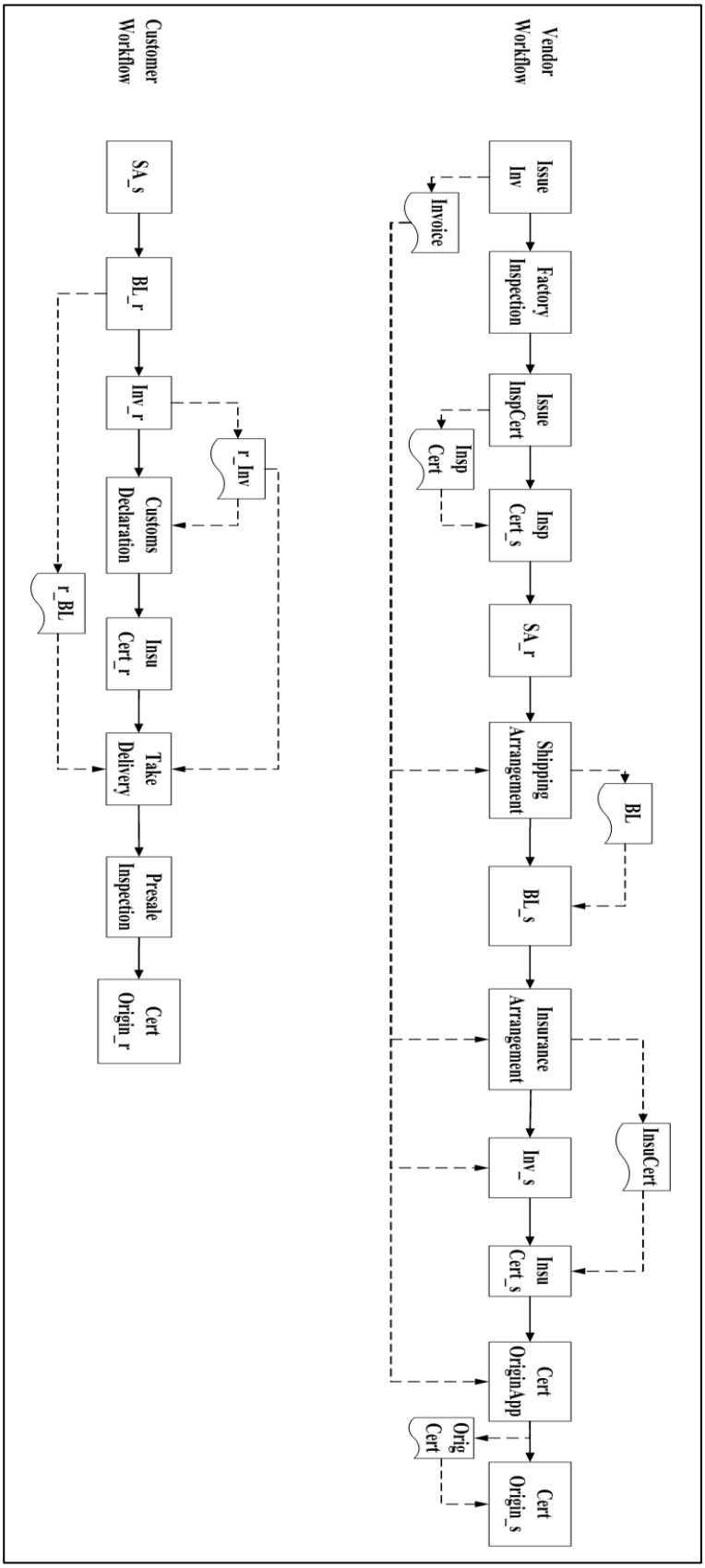

Figure 7.6 An Alternative Length Set of Compatible Workflows for *Vendor* and *Customer* (No Concurrency in *Customer's* OWLS Processes)

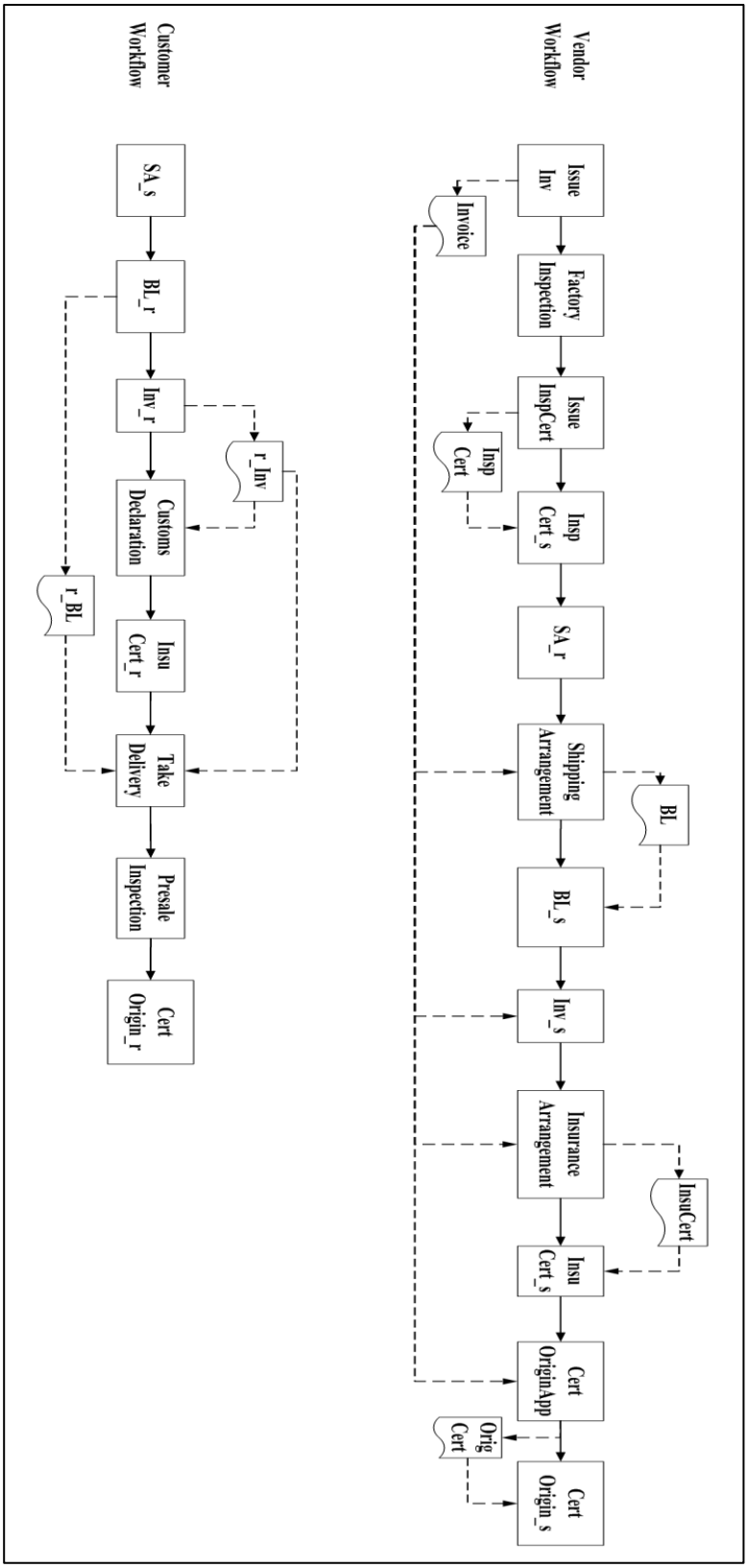

Figure 7.7 Another Alternative Length Set of Compatible Workflows for *Vendor* and *Customer* (No Concurrency in *Customer's* OWLS Processes)

Table 7.3 Changes to *Customer's* OWLS Processes to Remove Concurrency

S. No	Process Details
1	Name: *Inv_r* (Receive commercial invoice) Inputs/ Preconditions: *r_BL* (Bill of lading received from *Vendor*) *s_Inv* (Commercial invoice sent by *Vendor*) Outputs/ Effects: *r_Inv* (Commercial invoice received from *Vendor*)

As obvious in the Figure 7.6 and 7.7, the workflow for the *Customer* is the same in both figures while the workflow for the *Vendor* is different in both the figures. In the *Vendor*'*s* workflow in Figure 7.6, *InsuranceArrangement* must be executed before *Inv_s*. In the *Vendor*'*s* workflow in Figure 7.7, *Inv_s* must be executed before *InsuranceArrangement*. In all five sets of compatible workflows generated, the *Customer's* workflow will be the same, while the *Vendor's* workflow will be different.

7.3 Retailer/Wholesaler/Manufacturer/Supplier Business Collaboration Scenario

To illustrate the generality of the framework to handle multiple organisations, a scenario involving four organisations is used, namely retailer, wholesaler, manufacturer and supplier. It is a common business collaboration scenario in the real world and therefore we have used it as an example to test the prototype. The retailer, manufacturer, wholesaler and supplier are represented by *Retailer*, *Manufacturer*, *Wholesaler* and *Supplier* respectively. The details and descriptions of OWLS processes of each of the organisations are given in the next section.

7.3.1 OWLS Processes

The OWLS processes for *Retailer*, *Wholesaler*, *Manufacturer* and *Supplier* are given in Table 7.4, 7.5, 7.6, and 7.7 respectively. Only the processes relevant to this collaboration scenario are given.

Table 7.4 *Retailer's* OWLS Processes

S.No	**OWLS Process Details**
1	Name: *QuotationInqPrep* (Quotation inquiry preparation)

	Inputs/ Preconditions: *goods_req* (Goods required) Outputs/ Effects: *RInq* (*Retailer's* inquiry for quotation) Description: This process creates a quotation inquiry.
2	Name: *QuotationInq_s* (Send quotation inquiry) Inputs/ Preconditions: *RInq* (*Retailer's* inquiry for quotation) Outputs/ Effects: *s_RInq* (*Retailer's* inquiry for quotation sent) Description: This process sends a quotation inquiry to the *Wholesaler*.
3	Name: *Quotation_r* (Receive quotation) Inputs/ Preconditions: s_RInq (*Retailer's* inquiry for quotation sent) *s_WQuotation* (Quotation sent by the *Wholesaler*) Outputs/ Effects: *r_WQuotation* (Quotation *receive*d from the *Wholesaler*) Description: This process receives the quotation sent by the *Wholesaler*.
4	Name: *QuotationEvaluation* (Evaluate the quotation) Inputs/ Preconditions: *r_WQuotation* (Quotation *receive*d from the *Wholesaler*) Outputs/ Effects: *EvalReport* (Evaluation report) Description: This process evaluates the quotation received from the *Wholesaler*.
5	Name: *CreatePO* (Create a purchase order) Inputs/ Preconditions: *EvalReport* (Evaluation report) Outputs/ Effects: *RPO* (*Retailer's* purchase order) Description: This process creates a purchase order.
6	Name: *PO_s* (Send the purchase order) Inputs/ Preconditions: *RPO* (*Retailer's* purchase order) Outputs/ Effects: *s_RPO* (*Retailer's* purchase order sent) Description: This process sends the purchase order to the *Wholesaler*.
7	Name: *POAcpt_r* (Accept the purchase order approval/acceptance) Inputs/ Preconditions: *s_RPO* (*Retailer's* purchase order sent) *s_POA* (Purchase order approval sent by *Wholesaler*) Outputs/ Effects: *r_POA* (Received purchase order approval from the *Wholesaler*) Description: This process receives the purchase order approval from the *Wholesaler*.

8	Name: *ComInv_r* (Receive commercial invoice) Inputs/ Preconditions: *s_WInv* (Commercial invoice sent by the *Wholesaler*) *r_POA* (Received purchase order approval from the *Wholesaler*) Outputs/ Effects: *r_WInv* (Received commercial invoice from the *Wholesaler*) Description: This process receives the commercial invoice from the *Wholesaler*.
9	Name: *TakeDelivery* (Take Delivery) Inputs/ Preconditions: *r_WInv* (Received commercial invoice from the *Wholesaler*) Outputs/ Effects: *WDelivery* (Goods delivered by the *Wholesaler*) Description: This process takes delivery of goods shipped by the *Wholesaler*.
10	Name: *ApprovePayment* (Approve payment) Inputs/ Preconditions: *WDelivery* (Goods delivered by the *Wholesaler*) Outputs/ Effects: *RInvPay* (*Retailer's* payment for invoice) Description: This process approves payment to the *Wholesaler*.
11	Name: *InvPayment_s* (Send payment for invoice) Inputs/ Preconditions: *RInvPay* (*Retailer's* payment for invoice) Outputs/ Effects: *s_RInvPay* (*Retailer's* payment for the invoice sent) Description: This process sends the *Retailer's* payment for the invoice to the *Wholesaler*.

Table 7.5 *Wholesaler's* OWLS Processes

S.No	**OWLS Process Details**
1	Name: *QuotationInq_r* (Receive quotation inquiry) Inputs/ Preconditions: *s_RInq* (Quotation inquiry sent by the *Retailer*) Outputs/ Effects: *r_RInq* (Quotation inquiry received from the *Retailer*) Description: This process receives the *Retailer's* inquiry for quotation.
2	Name: *QuotationPreparation*(Prepare quotation) Inputs/ Preconditions: *r_RInq* (Quotation inquiry received from the *Retailer*) Outputs/ Effects: *WQuotation* (*Wholesaler's* quotation) Description: This process prepares a quotation.

3	Name: *Quotation_s* (Send Quotation) Inputs/ Preconditions: WQuotation (*Wholesaler's* quotation) Outputs/ Effects: *s_WQuotation* (*Wholesaler's* quotation sent) Description: This process sends the *Wholesaler's* quotation to the *Retailer*.
4	Name: *PO_r* (Receive purchase order) Inputs/ Preconditions: *s_WQuotation* (*Wholesaler's* quotation sent) *s_RPO* (Purchase order sent by the *Retailer*) Outputs/ Effects: *r_RPO* (Purchase order received from the *Retailer*) Description: This process receives the purchase order sent by the *Retailer*.
5	Name: *POApproval* (Purchase order approval) Inputs/ Preconditions: *r_RPO* (Purchase order received from the *Retailer*) Outputs/ Effects: *POA* (Purchase order approval) Description: This process approves the purchase order received from the *Retailer*.
6	Name: *POAcpt_s* (Send the purchase order approval/acceptance) Inputs/ Preconditions: *POA* (Purchase order approval) Outputs/ Effects: *s_POA* (Purchase order approval sent) Description: This process sends the purchase order approval/acceptance to the *Retailer*.
7	Name: *CreateInquiry* (Create quotation inquiry) Inputs/ Preconditions: *s_POA* (Purchase order approval sent) Outputs/ Effects: *WInq* (*Wholesaler's* quotation inquiry) Description: This process creates a quotation inquiry to send to the *Manufacturer*.
8	Name: *QuotationInquiry_s* (Send the quotation inquiry) Inputs/ Preconditions: *WInq* (*Wholesaler's* quotation inquiry) Outputs/ Effects: *s_WInq* (*Wholesaler's* quotation inquiry sent) Description: This process sends the quotation inquiry to the *Manufacturer*.
9	Name: *Quotation_r* (Receive quotation) Inputs/ Preconditions: *s_WInq* (*Wholesaler's* quotation inquiry sent) *s_MQuotation* (Quotation sent by the *Manufacturer*) Outputs/ Effects: *r_MQuotation* (Quotation received from the *Manufacturer*)

	Description: This process receives the quotation sent by the *Manufacturer*.
10	Name: *ApproveQuotation* (Approve quotation) Inputs/ Preconditions: *r_MQuotation* (Quotation received from the *Manufacturer*) Outputs/ Effects: *QuotApp* (Quotation approval) Description: This process approves the quotation received from the *Manufacturer*.
11	Name: *QuotationApproval_s* (Send quotation approval) Inputs/ Preconditions: *QuotApp* (Quotation approval) Outputs/ Effects: *s_ QuotApp* (Quotation approval sent) Description: This process sends the quotation approval to the *Manufacturer*.
12	Name: *Invoice_r* (Receive commercial invoice) Inputs/ Preconditions: *s_QuotApp* (Quotation approval sent) *s_MInv* (Commercial invoice sent by the *Manufacturer*) Outputs/ Effects: *r_MInv* (Commercial invoice received from the *Manufacturer*) Description: This process receives the commercial invoice from the *Manufacturer*.
13	Name: *InsuCert_r* (Receive insurance certificate) Inputs/ Preconditions: *r_MInv* (Commercial invoice received from the *Manufacturer*) *s_InsuCert* (Insurance certificate sent by the *Manufacturer*) Outputs/ Effects: *r_InsuCert* (Insurance certificate received from the *Manufacturer*) Description: This process receives the insurance certificate from the *Manufacturer*.
14	Name: *CustomsDeclaration* (Customs Declaration) Inputs/ Preconditions: *r_InsuCert* (Insurance certificate received from the *Manufacturer*) Outputs/ Effects: *CDR* (Customs declaration report) Description: This process declares the delivered goods to the customs.

15	Name: *TakeDelivery* (Take delivery) Inputs/ Preconditions: *r_MInv* (Commercial invoice received from the *Manufacturer*) *CDR*(Customs declaration report) Outputs/ Effects: *MDelivery* (Delivery taken from the *Manufacturer*) Description: This process takes delivery of goods sent by the *Manufacturer*.
16	Name: *PaymentApproval* (Approve Payment) Inputs/ Preconditions: *MDelivery* (Delivery taken from the *Manufacturer*) Outputs/ Effects: *WInvPay* (Payment for invoice) Description: This process approves payment to the *Manufacturer*.
17	Name: *InvoicePayment_s* (Send payment for invoice) Inputs/ Preconditions: *WInvPay* (Payment for invoice) Outputs/ Effects: *s_ WInvPay* (payment for the invoice sent) Description: This process sends the payment for the invoice to the *Manufacturer*.
18	Name: *IssueComInv* (Issue commercial invoice) Inputs/ Preconditions: *s_ WInvPay* (payment for the invoice sent) Outputs/ Effects: *WInv* (*Wholesaler's* commercial invoice) Description: This process issues the *Wholesaler's* commercial invoice.
19	Name: *ComInv_s* (Send commercial invoice) Inputs/ Preconditions: *WInv* (*Wholesaler's* commercial invoice) Outputs/ Effects: *s_ WInv* (*Wholesaler's* commercial invoice sent) Description: This process sends the *Wholesaler's* commercial invoice to the *Retailer*.
20	Name: *ShipGoods* (Ship goods) Inputs/ Preconditions: *s_ WInv* (*Wholesaler's* commercial invoice sent) Outputs/ Effects: *WSR* (*Wholesaler's* shipment report) Description: This process ships the goods to the *Retailer*.
21	Name: *InvPayment_r* (Receive payment for invoice) Inputs/ Preconditions: *WSR* (*Wholesaler's* shipment report) *s_RInvPay* (Payment for the invoice sent by the *Retailer*)

	Outputs/ Effects: *r_RInvPay* (Payment for the invoice received from the *Retailer*) Description: This process receives the payment for the invoice from the *Retailer*.

Table 7.6 *Manufacturer's* OWLS Processes

S.No	**OWLS Process Details**
1	Name: *QuotationInquiry_r* (Receive quotation inquiry) Inputs/ Preconditions: *s_WInq* (Quotation Inquiry sent by the *Wholesaler*) Outputs/ Effects: *r_WInq* (The *Wholesaler's* quotation inquiry received) Description: This process receives the quotation inquiry from the *Wholesaler*.
2	Name: *PrepareQuotation* (Prepare Quotation) Inputs/ Preconditions: *r_WInq* (Quotation Inquiry received from the *Wholesaler*) Outputs/ Effects: *MQuotation* (*Manufacturer's* quotation) Description: This process creates the *Manufacturer's* quotation.
3	Name: *Quotation_s* (Send the *Manufacturer's* quotation) Inputs/ Preconditions: *MQuotation* (*Manufacturer's* quotation) Outputs/ Effects: s_*MQuotation* (*Manufacturer's* quotation sent) Description: This process sends the *Manufacturer's* quotation to the *Wholesaler*.
4	Name: *QuotationApproval_r* (Receive quotation approval) Inputs/ Preconditions: s*_MQuotation* (*Manufacturer's* quotation sent) *s_QuotApp* (Quotation approval sent by the *Wholesaler*) Outputs/ Effects: r_QuotApp (Quotation approval received from the *Wholesaler*) Description: This process receives the quotation approval from the *Wholesaler*.
5	Name: *PrepareInquiry* (Prepare quotation inquiry) Inputs/ Preconditions: *r_QuotApp* (Quotation approval received from the *Wholesaler*)

	Outputs/ Effects: *MInq* (*Manufacturer's* quotation inquiry) Description: This process creates a quotation inquiry.
6	Name: *QuotationInquiry_s* (Send the quotation inquiry) Inputs/ Preconditions: *MInq* (*Manufacturer's* quotation inquiry) Outputs/ Effects: s_*MInq* (*Manufacturer's* quotation inquiry sent) Description: This process sends the *Manufacturer's* quotation inquiry to the *Supplier*.
7	Name: *ReceiveQuotation_r* (Receive quotation) Inputs/ Preconditions: *s_MInq* (*Manufacturer's* quotation inquiry sent) *s_SQuotation* (*Supplier's* quotation sent) Outputs/ Effects: *r_SQuotation* (*Supplier's* quotation received) Description: This process receives the quotation from the *Supplier*.
8	Name: *QuotationApp* (Approve quotation) Inputs/ Preconditions: *r_SQuotation* (*Supplier's* quotation received) Outputs/ Effects: *QApp* (Quotation approval) Description: This process approves the quotation received from the *Supplier*.
9	Name: *QuotationApp _s* (Send quotation approval) Inputs/ Preconditions: *QApp* (Quotation approval) Outputs/ Effects: s_*QApp* (Quotation approval sent) Description: This process sends the quotation approval to the *Supplier*.
10	Name: *CommercialInvoice_r* (Receive commercial invoice) Inputs/ Preconditions: *s_SInvoice* (*Supplier's* commercial invoice sent) *s_QApp* (Quotation approval sent) Outputs/ Effects: *r_SInvoice* (*Supplier's* commercial invoice received) Description: This process receives commercial invoice sent by the *Supplier*.
11	Name: *InsuranceCertificate_r* (Receive Insurance Certificate) Inputs/ Preconditions: *r_SInvoice* (Commercial invoice received from the *Supplier*) *s_InsuranceCert* (Insurance certificate sent by the *Supplier*) Outputs/ Effects: *r_InsuranceCert* (Insurance certificate received from the *Supplier*)

	Description: This process receives the insurance certificate sent by the *Supplier*.
12	Name: *DeclareToCustoms* (Declare goods to customs) Inputs/ Preconditions: *r_SInvoice* (Commercial invoice received from the *Supplier*) *r_InsuranceCert* (Insurance certificate received from the *Supplier*) Outputs/ Effects: *DeclarationReport* (Goods declaration report) Description: This process declares goods *to* customs.
13	Name: *TakeRawDelivery* (Take delivery of raw material) Inputs/ Preconditions: *r_SInvoice* (Commercial invoice received from the *Supplier*) *DeclarationReport* (Goods declaration report) Outputs/ Effects: *SDelivery* (Delivery taken from the *Supplier*) Description: This process takes delivery of raw material shipped by the *Supplier*.
14	Name: *ApprovePaymentInvoice* (Approves payment for the invoice) Inputs/ Preconditions: *SDelivery* (Delivery taken from the *Supplier*) Outputs/ Effects: *MInvPay* (*Manufacturer's* payment for the invoice) Description: This process approves payment for the invoice to the supplier.
15	Name: *PaymentInvoice_s* (Send payment for invoice) Inputs/ Preconditions: *MInvPay* (*Manufacturer's* payment for invoice) Outputs/ Effects: *s_ MInvPay* (*Manufacturer's* payment for invoice sent) Description: This process sends the payment for the invoice to the *Supplier*.
16	Name: *GoodsManufacturing* (Manufacture goods) Inputs/ Preconditions: s_*MInvPay* (*Manufacturer's* payment for the invoice sent) Outputs/ Effects: *Goods* (Manufactured goods) Description: This process manufactures goods.
17	Name: *CreateInvoice* (Create commercial invoice) Inputs/ Preconditions: *Goods* (Manufactured goods) Outputs/ Effects: *MInv* (*Manufacturer's* commercial invoice)

	Description: This process creates the *Manufacturer's* commercial invoice.
18	Name: *Invoice_s* (Send commercial invoice) Inputs/ Preconditions: *MInv* (*Manufacturer's* commercial invoice) Outputs/ Effects: *s_ MInv* (*Manufacturer's* commercial invoice sent) Description: This process sends the *Manufacturer's* commercial invoice to the *Wholesaler*.
19	Name: *ArrangeShipment* (Arrange shipment of goods) Inputs/ Preconditions: *s_ MInv* (Commercial invoice sent) Outputs/ Effects: *MSR* (*Manufacturer's* shipment report) Description: This process arranges shipment of goods to the *Wholesaler*.
20	Name: *ArrangeInsurance* (Arrange insurance of goods) Inputs/ Preconditions: *MSR* (*Manufacturer's* shipment report) Outputs/ Effects: *InsuCert* (Insurance certificate) Description: This process arranges insurance of the shipped goods.
21	Name: *InsuCert_s* (Send Insurance certificate) Inputs/ Preconditions: *InsuCert* (Insurance certificate) Outputs/ Effects: *s_ InsuCert* (Insurance certificate sent) Description: This process sends the insurance certificate to the *Wholesaler*.
22	Name: *InvoicePayment_r* (Receive payment for invoice) Inputs/ Preconditions: *s_InsuCert* (Insurance certificate sent) *s_WInvPay* (Payment for the invoice sent by the *Wholesaler*) Outputs/ Effects: *r_WInvPay* (Payment for the invoice received from the *Wholesaler*) Description: This process receives the payment for the invoice from the *Wholesaler*.

Table 7.7 *Supplier's* OWLS Processes

S.No	**OWLS Process Details**
1	Name: *QuotationInquiry_r* (Receive quotation inquiry) Inputs/ Preconditions: *s_MInq* (Quotation inquiry sent by the *Manufacturer*) Outputs/ Effects: *r_MInq* (Quotation inquiry received from the *Manufacturer*)

	Description: This process receives the quotation inquiry from the *Manufacturer*.
2	Name: *QuotationPrep* (Prepare quotation) Inputs/ Preconditions: *r_MInq* (Quotation inquiry received from the *Manufacturer*) Outputs/ Effects: *SQuotation* (*Supplier's* quotation) Description: This process creates a *Supplier's* quotation.
3	Name: *SendQuotation_s* (Send the *Supplier's* quotation) Inputs/ Preconditions: SQuotation (*Supplier's* quotation) Outputs/ Effects: *s_SQuotation* (*Supplier's* quotation sent) Description: This process sends the *Supplier's* quotation to the *Manufacturer*.
4	Name: *QuotationApp_r* (Receive quotation approval) Inputs/ Preconditions: *s_SQuotation* (*Supplier's* quotation sent) *s_QApp* (Quotation approval sent by the *Manufacturer*) Outputs/ Effects: *r_QApp* (Quotation approval received from the *Manufacturer*) Description: This process receives the quotation approval from the *Manufacturer*.
5	Name: *IssueInv* (Issue commercial invoice) Inputs/ Preconditions: *r_QApp* (Quotation approval received from the *Manufacturer*) Outputs/ Effects: *SInvoice* (*Supplier's* commercial invoice) Description: This process issues a commercial invoice.
6	Name: *CommercialInvoice_s* (Send the commercial invoice) Inputs/ Preconditions: *SInvoice* (*Supplier's* commercial invoice) Outputs/ Effects: *s_ SInvoice* (*Supplier's* commercial invoice sent) Description: This process sends the *Supplier's* commercial invoice to the *Manufacturer*.
7	Name: *AssembleGoods* (Assemble raw material components) Inputs/ Preconditions: *s_ SInvoice* (*Supplier's* commercial invoice sent) Outputs/ Effects: *RawComps* (Raw material components assembled) Description: This process assembles different components of raw material.

8	Name: *InsureRaw* (Insure the raw material) Inputs/ Preconditions: *s_ SInvoice* (*Supplier's* commercial invoice sent) Outputs/ Effects: *InsuranceCert* (Insurance certificate) Description: This process insures the raw material.
9	Name: *InsuranceCertificate_s* (Send insurance certificate) Inputs/ Preconditions: *InsuranceCert* (Insurance certificate) Outputs/ Effects: *s_ InsuranceCert* (Insurance certificate sent) Description: This process sends the insurance certificate to the *Manufacturer*.
10	Name: *ShipRaw* (Ship raw material) Inputs/ Preconditions: *RawComps* (Assembled raw material components) Outputs/ Effects: *SSR* (*Supplier's* shipment report) Description: This process ships the raw material to the *Manufacturer*.
11	Name: *Documentation* (Do the necessary documentation) Inputs/ Preconditions: *SSR* (*Supplier's* shipment report) Outputs/ Effects: *Doc* (Necessary book keeping documentation done) Description: This process does the necessary book keeping documentation after the shipment and insurance has been done.
12	Name: *UpdateRecords* (Update records) Inputs/ Preconditions: *SInvoice* (*Supplier's* commercial invoice) *Doc* (Documentation done) *s_InsuranceCert* (Insurance certificate sent) Outputs/ Effects: *RecUpd* (Records updated) Description: This process updates the database records after the necessary documentation has been done.
13	Name: *PaymentInvoice_r* (Receive payment for invoice) Inputs/ Preconditions: *s_MInvPay* (Payment for the invoice sent by the *Manufacturer*) *RecUpd* (Records updated) Outputs/ Effects: *r_MInvPay* (*Manufacturer's* payment for the invoice Received) Description: This process receives the payment for the invoice sent by the *Manufacturer*.

7.3.2 Results

The OWLS process definitions as given in Table 7.4, 7.5, 7.6 and Table 7.7 were passed to the system and it generated 10 sets of compatible workflows for the four organisations. Figure 7.8 and Figure 7.9 show two of the sets. The remaining sets are given in Appendix G. The workflows generated are accurate and compatible. The system takes 9832 milliseconds to generate the 10 sets of compatible workflows.

Control dependencies are represented by solid lines and data dependencies are represented by dotted lines. The data dependencies in both sets of compatible workflows are the same and the control dependencies are different. In the *Supplier's* workflow in Figure 7.8, *InsureRaw* has a control dependency on *AssembleGoods and ShipRaw* has control dependency on *InsuraceCertificate_s*. In the *Supplier's* workflow in Figure 7.9, *InsureRaw* has control dependency on *Documentation,* and *ShipRaw* has a control dependency on *AssembleGoods*.

The workflow generation process starts when *goodsreq* holds, which means that the *Retailer* needs goods. The final goals for the *Retailer, Wholesaler, Manufacturer* and *Supplier* are *s_RInvPay, r_RInvPay, r_WInvPay* and *r_MInvPay* respectively. The goals mean that the *Retailer* sends a payment for the invoice to *Wholesaler, Wholesaler* receives a payment for the invoice from the *Retailer, Manufacturer* receives a payment for the invoice from the *Wholesaler* and the *Supplier* receives a payment for the invoice from the *Manufacturer.*

Figure 7.10 shows the interface process for the workflows in Figure 7.8. The corresponding interface activities have been labelled with the same alphabets. It can be observed that for every receiving activity there is a corresponding sending activity. Since there is no deadlock in the interface activities, the workflows are compatible. Compatibility can be checked in the same way for all the other sets of generated workflows.

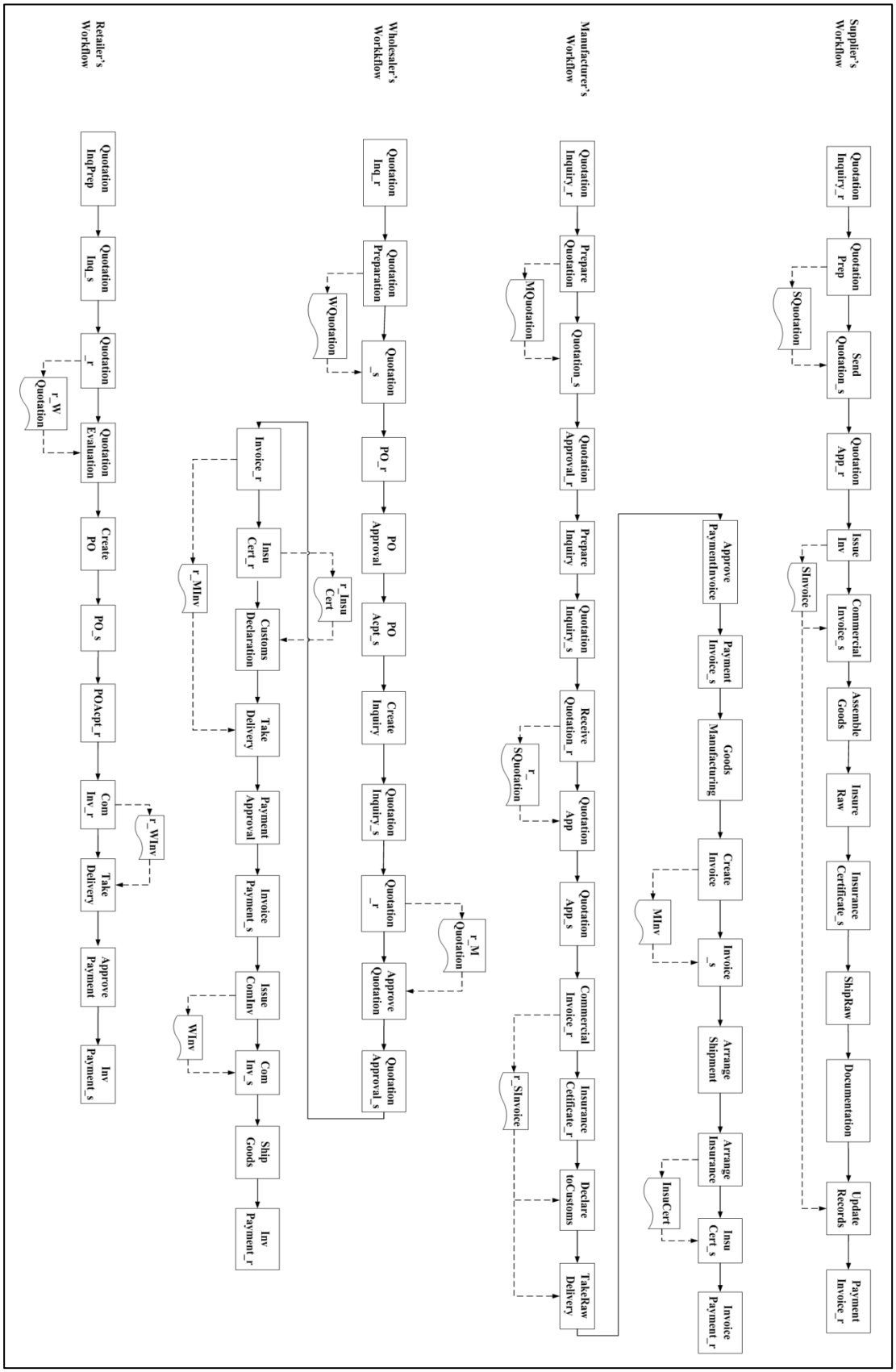

Figure 7.8 A Set of Compatible Workflows for *Retailer*, *Wholesaler*, *Manufacturer* and *Supplier*

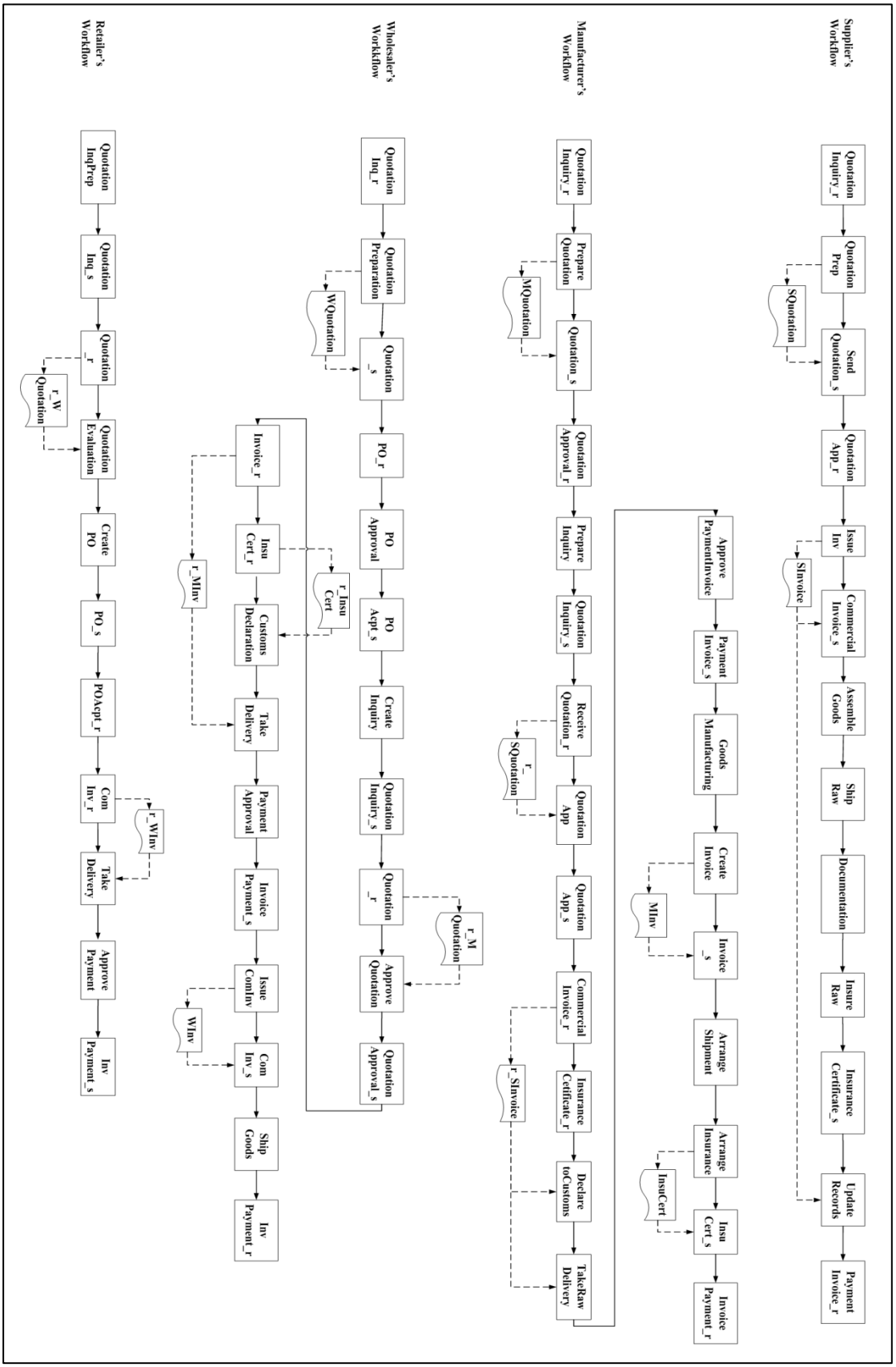

Figure 7.9 Another Set of Compatible Workflows for *Retailer*, *Wholesaler*, *Manufacturer* and *Supplier*

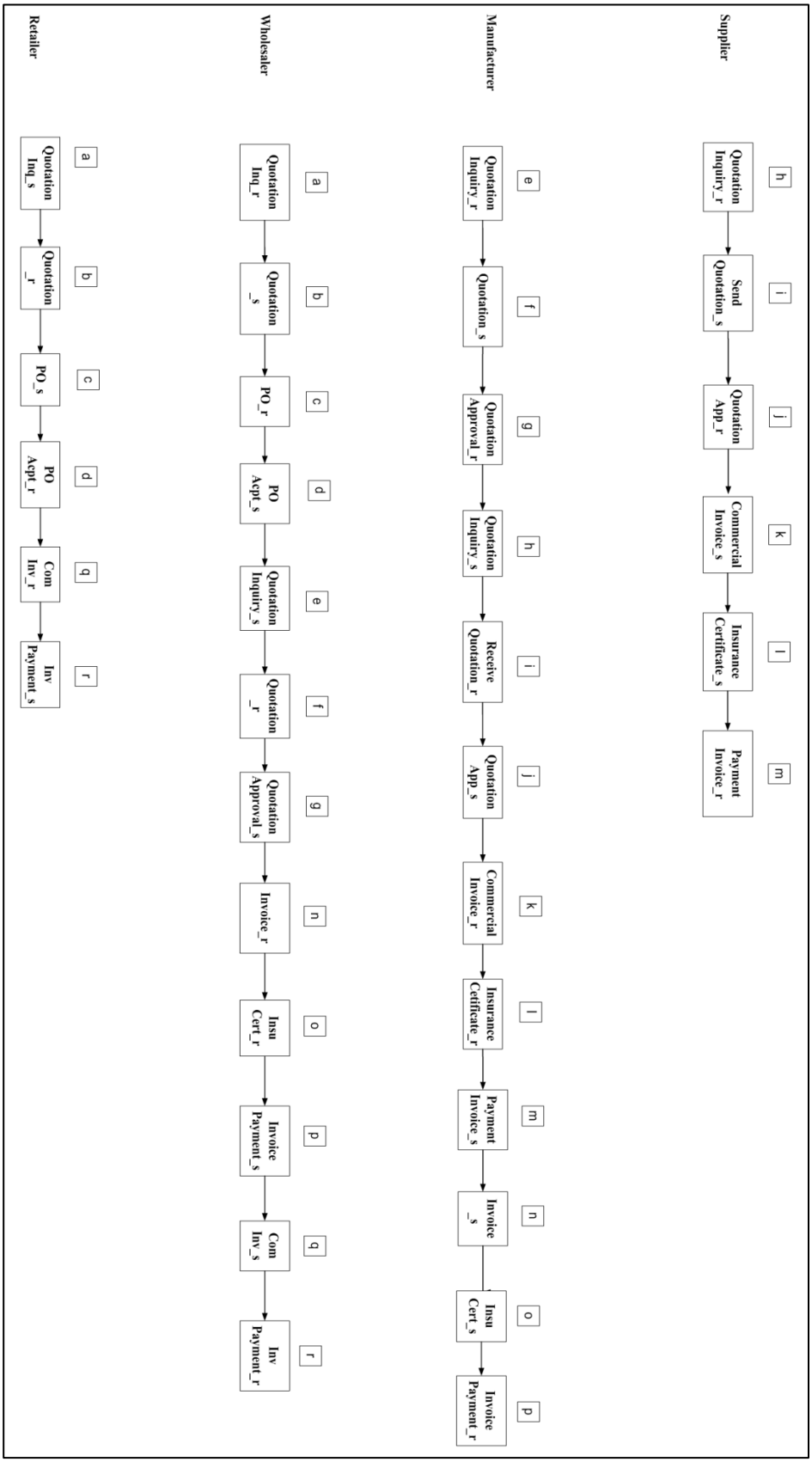

Figure 7.10 Interface Processes for the *Retailer*/*Wholesaler*/*Manufacturer*/*Supplier* Workflows given in Figure 7.8

At runtime, the set of compatible workflows with the least number of OWLS processes will be highlighted to the users for execution. In this particular scenario all the workflows are of the same length and so the first plan generated is highlighted to the users for execution. The users will enter the quantity of *goodsreq* to create a quotation inquiry. The *QuotationInqPrep* activity dependent on *goodsreq* will be executed to start the execution of the workflows. The in-house and cross organisational control and data dependencies will be followed, to make sure that all collaborating workflows in the selected set are enacted to the end. The execution of the compatible workflows to the end achieves the desired goals.

7.4 Evaluation

The reported collaboration examples illustrate that the prototype creates multiple sets of correct compatible workflows for the collaborating organisations. The workflows are generated by composing OWLS processes of the collaborating organisations into alternative composition orders, while retaining the compatibility between the workflows. The developed framework, therefore, assumes that the collaborating organisations either have local OWLS process definitions or have links to remote OWLS processes definitions. The framework focuses on the compositional capabilities of OWLS processes and automatic discovery of OWLS processes from the web is not targeted in this framework.

SHOP2 scales well to complex domains, so it is easy for the developed framework to generate workflows in complex domains. Since SHOP2 is a very efficient system, the planning time is highly efficient as shown in Section 7.2.2 and 7.3.2. The OWLS processes that are not relevant to the workflow generation process are discarded, to enhance the efficiency of the planning process.

The implemented framework can handle collaboration at build time and runtime among arbitrary number of organisations. This is one of the most powerful qualities of the proposed system. Most of the existing collaboration systems handle either build time or runtime collaboration. Very few collaboration systems handle both for more than two organisations.

Since the workflow generation is based on web services composition, the implemented framework supports interoperability. Web services from highly diverse sources can be composed in a workflow, and invoked to achieve a desired goal. The standard light-weight XML based messaging protocols and WSDL access descriptions of the web services architecture make it possible for the implemented framework to support interoperability across diverse platforms, applications and databases.

The implemented framework encourages cohesiveness and modularity. Since, the implemented framework plans to create workflows to achieve the goals of multiple collaborating organisations, the workflows of the collaborating organisations are highly related. The SHOP2 and OWLS decomposition mechanism of complex tasks into atomic tasks makes the framework highly modular. It can compose highly diverse web services into workflows to achieve situation-specific goals.

The implemented framework encourages reusability of existing resources. Existing units of functionalities represented in the form of web services are modelled using OWLS processes and composed in workflows to achieve situation-specific desired goals. If the web services provider allows, a web service can be composed in the workflows of multiple requesting organisations and can be executed multiple times. So the already developed functionalities do not need to be redeveloped and can be reused to save time and resources.

The developed framework makes the workflow collaboration processes highly automated. The collaborating organisations do not have to bother to model or adapt their workflows. They also do not have to reconcile their workflows with the workflows of collaborating organisations. Similarly, if changes occur to the workflows, the organisations simply have to add in the new functionality in the form of OWLS processes or change the existing OWLS processes and their respective web services without having to worry about their impact over the collaborating workflows. This loosely coupled nature of the developed framework is enabled by the use of web services architecture.

The developed framework also has certain limitations. The process definitions passed to the system and workflows generated are in the OWLS format. This limits execution of the generated workflows to the systems that can execute OWLS workflows. OWLS

workflows execution systems are not common. The translation of OWLS workflows into Business Process Model and Notation (BPMN) language would make the workflows executable for most major WfMSs [124]. Moving from OWLS to BPMN will also enable the system to take BPMN definitions of activities as input. This will enable a much higher number of organisations to use the developed framework, since BPMN is one of the most widely used languages used for workflow modelling.

Since SHOP2 does not support the generation of parallel plans, the developed framework does not generate parallel workflows. In parallel workflows two or more activities can occur concurrently. The developed framework generates multiple sequential workflows, when it encounters activities that can be executed in parallel. This will force an activity to wait till the preceding activity is executed, although they could be executed in parallel. This is a limitation that should be addressed in the future since in the real world workflows often have activities that can be executed in parallel.

The developed framework also does not support iterative workflows, which may be important in some applications. OWLS repeat-while and repeat-until composite processes can be used to support iterative workflows. The translation of OWLS repeat-while and repeat-until composite processes directly into recursive SHOP2 methods can be used to support iterative workflows.

The developed framework composes atomic OWLS processes into compatible workflows for collaborating organisations. The composite processes are decomposed till they only contain atomic processes, and the atomic processes are then grouped as an *if-then-else* method. Although the primary aim of the framework is to compose the atomic processes of the collaborating organisations into compatible workflows, the translation of composite processes directly into SHOP2 methods will enable the users to give hints to the planner about how to proceed with the composition process, which may be important in some scenarios.

The implemented framework uses input, output, precondition and post-condition variables of OWLS processes for workflow generation. The support for additional process variables will enable the framework to support advanced processes.

If the required preconditions do not hold in a certain state of the world then the workflow generation fails. In such cases it is advisable to inform the collaborating

organisations about the reasons of failure and suggest possible solutions. Right now, the implemented framework lacks this functionality.

Each receiving OWLS process of the collaborating organisations is expected to know the outputs of the corresponding sending OWLS process. Similarly, OWLS processes within the organisations are expected to use unique names for unique inputs, preconditions, outputs and post conditions. This might not be possible in some cases. The matching mechanism should be extended to make it more flexible. The use of OWL reasoners to do the matching can be helpful in this regard.

The implemented framework merges the domains of the collaborating organisations into a single domain to create joint workflows. An alternative approach is to use a separate instance of SHOP2 planner for each organisation, and use intelligent agents to collaborate at the time of workflow generation to ensure that the compatibility is intact. If the addition of an activity in the workflow makes it incompatible with the workflows of the collaborating organisations then the step must be backtracked. The querying and backtracking mechanism presented by Au *et al.* [125] can be followed for this purpose.

7.5 Conclusion

This chapter presented two business collaboration scenarios and used them to illustrate the automatic generation and execution of workflows. It is shown that the developed framework is able to generate compatible workflows that achieve the high level goals of multiple collaborating organisations.

The first example is a very popular business collaboration scenario. Based on the OWLS process definitions of the hypothetical *Vendor* and *Customer*, 20 alternate composition paths were identified. The identification of multiple composition paths led to the generation of 20 sets of composite workflows. It was shown that the generated workflows only contained OWLS processes necessary to achieve the goal states from the initial states.

The second example is based on the business collaboration among a hypothetical *Retailer*, *Wholesaler*, *Manufacturer* and *Supplier*. Based on the OWLS process definitions of the hypothetical *Retailer, Wholesaler, Manufacturer* and *Supplier*, 10 sets

of composite workflows were generated. The developed framework also supports the runtime execution and runtime collaboration of the generated workflows.

It is identified that the developed framework supports modularity, inter-operability, cohesion and re-usability. Since the framework is based on an extended version of SHOP2 which is a very efficient planning system, the planning time is highly efficient. The proposed framework lacks support for parallel and iterative workflows which requires attention in the future.

Chapter 8: Conclusion and Future Work

8.1 Introduction

The main objectives of this thesis have been as follows:

1. to propose a framework for automatically generating compatible workflows for multiple collaborating organisations from the OWLS process definitions and high level goals of the collaborating organisations, and

2. to provide runtime enactment and collaboration support for the generated workflows.

To achieve these aims, the thesis identified the requirements for workflow generation and collaboration, and exploited web services architecture and AI planning to fulfil the requirements.

This chapter provides a discussion of the main conclusions of this research and a summary of the main contributions of this thesis. Section 8.2 gives the summary and conclusions of the thesis, Section 8.3 summarises the main contributions of the thesis and Section 8.4 gives the directions for future work.

8.2 Thesis Summary and Conclusions

Chapter 2 and 3 discussed workflow, automatic workflow generation and workflow collaboration in detail. Due to the increase in electronic commerce, the demand for business process automation has increased. Workflow technology is used for business process automation. Since an increasing number of organisations are moving to automated business processes, the demand for automatic workflow collaboration with other workflows has also increased. Although research has been done on automatic workflow collaboration, the collaboration systems are not fully automated and they involve users in making decisions at every step of the collaboration.

To reuse the existing components of work done and avoid the continuous reengineering of workflows, the focus in the research on workflow has recently shifted to automatic workflow generation. Existing automatic workflow generation frameworks are able to generate workflows automatically for a single organisation only. The generation of a

workflow for a single organisation still leaves the organisation to reconcile its workflow with the business partners and adapt accordingly. Hence, a gap in the literature about the integration of automatic workflow generation and workflow collaboration both at build-time and runtime stages was identified.

Workflow generation was identified as an AI planning problem [9].The efficiency and accuracy of the workflow generation depends on the planner and the formal domain used for planning. Chapter 4 reviewed the major planning paradigms and their representative planners. It was identified that domain knowledge is the key to successful planning for workflow generation. SHOP2 was selected as the most suitable planner for the workflow generation problem.

The thesis presented a novel algorithm to translate OWLS web services descriptions into SHOP2 domain descriptions. The translation algorithm enables SHOP2 to identify alternate composition paths to generate multiple workflows. Since the web services domain is a complex domain, sometimes the domain generated can be inefficient and unclear and can force SHOP2 into infinite loops. SHOP2 was extended to make it more suitable for the automatic workflow generation problem by enabling it to avoid going into infinite loops.

An integrated approach based on the integration of automatic workflow generation, build time workflow collaboration, runtime enactment and runtime workflow collaboration was presented in Chapter 5. The approach applies AI planning to workflow generation and workflow collaboration.

A framework based on the integration approach was also presented in Chapter 6. The developed framework was able to generate compatible workflows for multiple collaborating organisations. It supports the runtime execution and collaboration of the generated workflows. It also supports the basic requirements of the workflow generation and workflow collaboration problem i.e. reusability, cohesion, modularity, efficiency, domain complexity, interoperability and loose coupling.

The developed framework was tested for two multi-organisation business collaboration examples in Chapter 7. It was demonstrated that the framework could handle more than two collaborating organisations. The planning time was efficient, since the framework filters out the irrelevant OWLS processes as shown in Section 7.2.2 and uses an

extended form of SHOP2, a very efficient planner [31]. The workflows generated were found to be compatible and accurate since their execution achieved the desired goals.

The developed framework has certain limitations as well. It does not create parallel workflows due to lack of support for concurrency by SHOP2, but it is capable of identifying alternate composition paths and creating multiple sequential workflows for each parallel workflow. It expects the collaborating organisations to follow the same naming conventions for the inputs and outputs of the interface activities, which might not always be possible. Similarly, if it fails to produce valid workflows, the collaboration organisations are not given any feedback about the reasons of failure and possible solutions. In future, we aim to look at these limitations and improve the system to enable it to deal with these limitations.

8.3 Contributions

Following are the major contributions of the thesis.

- The thesis presented an integration approach based on the integration of automatic workflow generation, workflow enactment and cross organisational build time and run-time workflow collaboration. The proposed approach extended the application of AI Planning to the integration of workflow generation and workflow collaboration.

- The thesis presented a framework to create compatible workflows for multiple collaborating organisations. It is the only framework so far that creates compatible workflows for multiple collaborating organisations, without involving any reconciliation among collaborating workflows or any negotiations among the collaborating organisations.

- The developed framework handles both build-time and runtime workflow collaboration for arbitrary number of organisations. This is a powerful capability that is not common in literature.

- The thesis presented a run-time execution mechanism for the automatically generated compatible workflows. The runtime execution mechanism ensures the

smooth transfer of data and information among the in-house and cross organisational activities.

- The thesis introduced a novel technique to increase the efficiency of the workflow generation process. The system reasons about the usability of each atomic process in the workflow generation. The processes that do not make part of the workflow generation process are discarded. This way only the processes used in the workflow generation are translated into SHOP2 format and the planning engine only has to do planning based on the processes that are strictly used in the workflow generation.

- A novel algorithm for translating OWLS processes into SHOP2 domain descriptions has been developed. The developed translation algorithm creates an efficient and accurate SHOP2 domain to enable the planner to compose the atomic processes of the collaborating organisations into multiple sets of compatible workflows.

- Due to the very complex and diverse nature of web services, sometimes, it is not possible to translate the OWLS process definitions into a clear and efficient formal domain for SHOP2. Unclear and inefficient domains can force SHOP2 into infinite loops. SHOP2 has been extended to enable it to deal with such issues and make it more suitable for the workflow generation problem. Also the extended SHOP2 is capable of planning in terms of control as well as data dependencies for workflow generation.

- A novel algorithm has been developed to merge the domains of the collaborating organisations into a single joint domain. The joint domain can be considered as the domain of a single parent organisation, containing collaborating sub-organisations. Multiple joint plans can be created based on the joint domain. An algorithm to divide the joint plans into sub-plans is also implemented. Each sub-plan represents a workflow for a single collaborating organisation.

8.4 Future Work

The following are some of the directions for future work.

- If the developed framework fails to generate any valid workflows, it is advisable to inform the collaborating organisations about the reasons for failure and possible solutions. In future, the framework will be extended with the functionality to return the reasons for failure and suggestions to reach to the solution, in case it fails to produce any valid workflows.

- It is required that the collaborating organisations have access to their OWLS atomic processes that are to be composed and enacted. In future, the implemented framework will be extended with the functionality to discover web services from web services registries found locally or remotely on the web.

- The developed framework does not generate parallel workflows. ConGolog is a high-level programming language, based on situation calculus. A ConGolog interpreter that is able to support concurrency has been exploited for web services composition in [10, 11]. One of the future directions of work is to try the ConGolog interpreter for cross organisational workflows generation and exploit the parallel planning capabilities of ConGolog. SHOP2 can also be extended for parallel planning.

- The developed framework plans for workflows on the basis of functional attributes such as preconditions, inputs, outputs and post-conditions of the OWLS processes. The functionality of the framework can be extended to include additional functional attributes in planning. Similarly, workflows could be highlighted to the users for execution on the basis of the weight assigned to the workflows on the basis of certain non-functional attributes of the web services in the workflows e.g. cost, service quality, security etc.

- We also aim to investigate the use of OWL reasoners in SHOP2 planner to exploit the inferencing capabilities of OWL reasoners. The introduction of an OWL reasoner in SHOP2 will make it possible to handle very huge web domains. It will also ease the requirement of following the same naming convention for input, output, preconditions and post conditions of interface activities, by trying to do the matching and inferencing on the basis of OWL classes.

- We have developed a translation mechanism from OWLS to XML rules, for a rule based system for workflow enactment, developed by Phoenix [126]. We aim to improve the mechanism and work in collaboration with Phoenix on a rule based system for workflow generation and workflow collaboration, and compare it to the developed framework for accuracy and efficiency.

- Last but not least, the translation of OWLS into BPMN will be investigated and a translation algorithm will be developed. BPMN is one of the most widely used languages for modelling workflows. This will enable a high number of organisations to use the developed framework. This will also enable many independent WfMSs to enact the workflows generated by the developed framework.

References

1. *Workflow Management Coalition: Terminology & Glossary.* 1999, Technical Report WFMC-TC-1011.

2. *OWL Services Coalition: OWL-S: Semantic markup for web services*. 2003; Available from: http://www.ai.sri.com/daml/services/owl-s.

3. Yang, J. and Papazoglou, M.P., *Interoperation support for electronic business.* Communications of the ACM, 2000, **43**(6), p. 39–47.

4. Chen, X. and Chung, P.W.H., *Facilitating B2B E-business by IT-supported business process negotiation services*. in Proceedings of IEEE International Conference on Service Operations and Logistics, and Informatics, 2008.

5. Schulz, K.A. and Orlowska, M.E., *Facilitating cross-organisational workflows with a workflow view approach.* Data and Knowledge Engineering, 2004, **51**(1), p. 109–147.

6. Chiu, D.K.W., Cheung, S.C., Karlapalem, K., Li, Q., Till, S. and Kafeza, E., *Workflow View Driven Cross-Organizational Interoperability in a Web-Services Environment.* Information Technology and Management, 2004, **5**, p. 221–250.

7. Chen, X. and Chung, P.W.H., *A simulation-based difference detection technique for bottom-up process reconciliation*. in Proceedings of the 9th International Conference on Enterprise Information Systems, 2007.

8. Chen, L. and Yang, X., *Applying AI Planning to Semantic Web Services for Workflow Generation*. in Proceedings of First International Conference on Semantics, Knowledge and Grid, 2005, IEEE Computer Society, Washington DC, USA.

9. Dong, X. and Wild, D., *An Automatic Drug Discovery Workflow Generation Tool Using Semantic Web Technologies*. in Proceedings of the Fourth IEEE International Conference on eScience, 2008, IEEE Computer Society.

10. McIlraith, S. and Son, T., *Adapting Golog for composition of semantic Web services*. in Proceedings of Eighth International Conference on Knowledge Representation and Reasoning (KR2002), 2002, Toulouse, France.

11. McIlraith, S. and Son, T., *Adapting golog for programming in the semantic web.* 2001.

12. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V. and Shan, M.-C., *Adaptive and dynamic service composition in EFlow*. in Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE), 2000, Stockholm, Sweden: Springer Verlag.

13. Sirin, E., Parsia, B., Wu, D., Hendler, J. and Nau, D., *HTN planning for web service composition using SHOP2.* Journal of Web Semantics, 2004, **1**(4), p. 377–396.

14. Sirin, E., Hendler, J. and Parsia, B., *Semi-automatic composition of Web services using semantic descriptions*. in Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS, 2003.

15. Narayanan, S. and McIlraith, S.A., *Simulation, verification and automated composition of web services*. in Proceedings of the 11th international conference on World Wide Web, 2002, Honolulu, Hawaii, USA: ACM.

16. Ponnekanti, S. and Fox, A., *SWORD: A developer toolkit for web service composition*. in Proceedings of the 11th International WWW Conference (WWW2002), 2002.

17. *Transplan*. Available from: http://sourceforge.net/projects/transplan/.

18. Smith, H., *A Systems Integrator's Perspective on Business Process Management, Workflow and EAI.* 2002.

19. Srivastava, B. and Koehler, J., *Web Service Composition - Current Solutions and Open Problems*. in ICAPS 2003 Workshop on Planning for Web Services, 2003.

20. Georgakopoulos, D., Hornick, M. and Sheth, A., *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.* Distributed and Parallel Databases, 1995, **3**(2), p. 119–153.

21. Medina-Mora, R., Winograd, T. and Flores, R., *ActionWorkflow as the Enterprise Integration Technology*. in Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, 1993.

22. Guillaume, F., *Trying to Unify Entity-based and Activity-based Workflows*. Available from: http://wiki.zope.org/zope3/TryingToUnifiyWorkflowConcepts.

23. Shapiro, R., *A Comparison of XPDL, BPML and BPEL4WS*. 2001; Available from: http://www.ebpml.org/A_Comparison_of_XPDL_and_BPML_BPEL.doc.

24. Chen, X., *IT supported business process negotiation, reconciliation and execution for cross-organisational e-business collaboration*. Thesis at the Faculty of Computer Science 2008, Loughborough University.

25. Aalst van der, W.M.P., *Workflow Patterns.* Distributed and Parallel Databases, 2003, p. 5–51.

26. *Workflow Management Coalition: The Workflow Reference Model.* 1995, Technical Report WFMC-TC-1003.

27. Ader, M., *Seven Workflow Engines Reviewed.* Document World, 1997, **2**(3).

28. Russel, S. and Norvig, P., *Artificial Intelligence: A Modern Approach.* 1995, Prentice-Hall Inc.

29. Wooldridge, M. and Jennings, N.R., *Intelligent agents: theory and practice.* Knowledge Engineering Review, 1995, **10**(2), p. 115–152.

30. Rao, J. and Su, X., *A Survey of Automated Web Service Composition Methods*. in Proceedings of First International Workshop on Semantic Web Services and Web Process Composition SWSWPC2004, LNCS, 2004, San Diego, USA: Springer.

31. Wu, D., Sirin, E., Hendler, J., Nau, D. and Parsia, B., *Automatic Web Services Composition Using SHOP2*. in Workshop on Planning for Web Services, 2003.

32. Wu, D., Parsia, B., Sirin, E., Hendler, J. and Nau, D., *Automating DAML-S web services composition using SHOP2*. in Proceedings of 2nd International Semantic Web Conference (ISWC2003), 2003.

33. Wang, S.Y., Shen, W.M. and Hao, Q., *An agent-based Web service workflow model for inter-enterprise collaboration.* Expert System with Applications, 2006, p. 787–799.

34. Jennings, N.R., *An agent-based approach for building complex software systems.* Communications of the ACM, 2001, **44**(4), p. 35-41.

35. Casati, F., Sayal, M. and Shan, M.-C., *Developing e-services for composing eservices*. in Proceedings of 13th International Conference on Advanced Information Systems Engineering(CAiSE), 2001, Interlaken, Switzerland: Springer Verlag.

36. Saleem, M., Chung, P.W.H., Fatima, S. and Dai, W., *Intelligent Business Transaction Agents for Cross-Organizational Workflow Definition and Execution*. in Intelligent Information Processing V, 2010.

37. Orriëns, B. and Yang, J., *Establishing and maintaining compatibility in service oriented business collaboration*. in Proceedings of the 7th international conference on Electronic commerce, 2005, Xi'an, China: ACM.

38. Boivie, C.A., *Cross-organizational collaboration: from dating to tying the knot*. 2007 [cited 2011 18th October]; Available from: http://www.backbonemag.com/Magazine/CIO_View_11080701.asp.

39. Chen, X. and Chung, P.W.H., *A Framework for Cross-Organizational Workflow Collaboration*. in 13th Cross-Strait Academic Conference On Information Management Development & Relevant Strategy, 2007.

40. Chung, P.W.H. and Chen, X., *Reconciling and Enacting Cross-Organisational Workflow for B2B E-Commerce*. in BPM and Workflow Handbook Digital Edition v2, 2008, p. 347–360.

41. Tagg, R., *Workflow in different styles of Virtual Enterprise*. in Proceedings of Workshop on Information Technology for Virtual Enterprises, 2001.

42. Wombacher, A., *Decentralized establishment of consistent, multi-lateral collaborations*. Thesis at the Faculty of Informatics, 2005, Technical University Darmstadt.

43. Zartman, I.W., *Negotiation as a Joint Decision-Making Process.* Journal of Conflict Resolution, 1977, **21**(4), p. 619–638.

44. Summers, D., *Longman dictionary of contemporary English.* 2002, Harlow, Essex, England: Pearson Education Limited.

45. Hamner, W.C. and Yukl, G.A., *The Effectiveness of Different Offer Strategies in Bargaining.* Social-Psychological Perspectives, ed. D. Druckman, 1977, California Sage Publication.

46. Gulliver, P.H., *Disputes and negotiations: A cross-cultural perspective.* 1979, Academic Press (New York).

47. Rosenschein, J.S. and Zlotkin, G., *Rules of Encounter*. 1994, MIT Press.

48. Pruitt, D.G. and Carnevale, P.J., *Negotiation in social conflict.* 1993, Buckingham: Open University Press.

49. Follett, M.P., Metcalf, H.C. and Urwick, L., *Dynamic administration: The collected papers of Mary Parker Follett.* 1942, New York: Harper & Brother Publishers.

50. Hiltrop, J. and Udall, S., *The essence of negotiation.* 1995, London and New York: Prentice Hall.

51. Krukkert, D., *Matchmaking of ebXML Business Processes* 2003, Technical Report IST-28584-OX_D2.3_v.2.0.

52. van-der-Aalst, W.M.P. and Weske, M., *The P2P Approach to Interorganizational Workflows*. in Proceedings of the 13th International Conference on Advanced Information Systems Engineering, 2001, Springer-Verlag.

53. Byde, A., Piccinelli, G. and Lamersdorf, W., *Automating negotiation over B2B processes*. in Proceedings of 13th International Workshop on Database and Expert Systems Applications, 2002.

54. *RosettaNet Business Dictionary.* 2002.

55. van-der-Aalst, W.M.P., *The Application of Petri Nets to Workflow Management.* The Journal of Circuits, Systems and Computers, 1998, **8**(1), p. 21–66.

56. Pratt, V., *Modeling concurrency with geometry*. in Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on principles of programming languages, 1991, Orlando, Florida, United States: ACM.

57. Van-Der-Aalst, W.M.P., *Process-oriented architectures for electronic commerce and interorganizational workflow*. Information Systems, 1999, **24**(8), p. 639–671.

58. Chen, Q. and Hsu, M., *Inter-enterprise collaborative business process management*. in Proceedings of 17th International Conference on Data Engineering, 2001.

59. Chen, X. and Chung, P., *Cross-Organisational Workflow Enactment Via Progressive Linking by Run-Time Agents*. in International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, 2006.

60. Biegus, L. and Branki, C., *InDiA: a framework for workflow interoperability support by means of multi-agent systems.* Engineering Applications of Artificial Intelligence, 2004, **17**(7), p. 825–839.

61. Peer, J., *Web Service Composition as AI Planning - a Survey.* 2005, University of St. Gallen, Switzerland.

62. Dustdar, S. and Schreiner, W., *A survey on web services composition.* International Journal of Web and Grid Services, 2005, **1**(1), p. 1–30.

63. Fikes, R.E. and Nilsson, N.J., *Strips: A new approach to the application of theorem proving to problem solving.* Artificial Intelligence, 1971, **2**(3-4), p. 189–208.

64. Bonet, B. and Geffner, H., *Planning as heuristic search.* Artificial Intelligence, 2001, **129**(1-2), p. 5–33.

65. Bonet, B. and Geffner, H., *HSP: Heuristic Search Planner, entry at the AIPS-98 Planning competition*. 1998, Pittsburgh.

66. Bonet, B. and Geffner, H., *Heuristic search planner 2.0.* AI Magazine, 2001, **22**(3), p. 77–80.

67. Pearl, J., *Heuristics: Intelligent search strategies for computer problem solving.* 1985, Reading (Massachusetts), USA: Addison-Wesley Publishing Company.

68. Korf, P., *Linear-space best-first search.* Artificial Intelligence, 1993, **62**(1), p. 41–78.

69. Hoffman, J., *Ff: The fast-forward planning system.* The AI Magazine, 2001.

70. Hoffmann, J., *Extending FF to numerical state variables*. in Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02), 2002.

71. Hoffmann, J., *The metric-FF planning system: translating "Ignoring delete lists" to numeric state variables.* Journal of Artificial Inteligence Research, 2003, **20**, p. 291–341.

72. Blum, A. and Furst, M., *Fast Planning Through Planning Graph Analysis*. in Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95), 1995.

73. Nebel, B., Dimopoulos, Y. and Koehler, J., *Ignoring irrelevant facts and operators in plan generation*. in Proceeding of ECP-97, 1997, Springer Berlin / Heidelberg.

74. Koehler, J., Nebel, B., Hoffmann, J. and Dimopoulos, Y., *Extending planning graphs to an ADL subset*. in Recent Advances in AI Planning, S. Steel and R. Alami, Editors. 1997, Springer Berlin / Heidelberg, p. 273–285.

75. Long, D. and Fox, M., *Effecient implementation of the plan graph in stan.* Journal of Artificial Intelligence Research, 1999, **10**, p. 87–115.

76. Weld, D.S., Anderson, C.R. and Smith, D.E., *Extending Graphplan to handle uncertainty and sensing actions*. in Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, 1998, Madison, Wisconsin, United States: American Association for Artificial Intelligence.

77. Gazen, B.C. and Knoblock, C.A., *Combining the Expressivity of UCPOP with the Efficiency of Graphplan*. in Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning, 1997, Springer-Verlag.

78. Chapman, D., *Planning for conjunctive goals.* Artificial Intelligence, 1987, **32**(3), p. 333–377.

79. McAllester, D. and Rosenblitt, D., *Systematic nonlinear planning*. in Proceedings of the ninth National conference on Artificial Intelligence, 1991, Anaheim, California: AAAI Press.

80. Penberthy, S. and Weld, D., *UCPOP: A Sound, Complete, Partial Order Planner for ADL*. in KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, B. Nebel, C. Rich, and W. Swartout, Editors. 1992, p. 103–114.

81. Nguyen, X. and Kambhampati, S., *Reviving partial order planning*. in Proceedings of the 17th international joint conference on Artificial Intelligence, 2001, Seattle, WA, USA: Morgan Kaufmann Publishers Inc.

82. Smith, D.E., Frank, J. and J'onsson, A.K., *Bridging the gap between planning and scheduling.* Knowledge Engineering Review, 2000, **15**(1), p. 47-83.

83. Younes, H.L.S. and Simmons, R.G., *On the Role of Ground Actions in Refinement Planning*. in Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling Systems, 2002.

84. Younes, H. and Simmons, R., *VHPOP: Versatile heuristic partial order planner.* Journal of Artificial Intelligence Research, 2003.

85. Kautz, H. and Selman, B., *Planning as satisfiability*. in Proceedings of the 10th European conference on Artificial intelligence, 1992, Vienna, Austria: John Wiley & Sons, Inc.

86. Gupta, N. and Nau, D.S., *On the complexity of blocks-world planning.* Artificial Intelligence, 1992, **56**(2-3), p. 223–254.

87. Kautz, H.A. and Selman, B., *Blackbox: A new approach to the application of theorem proving to problem solving.* 1998.

88. Gerevini, A. and Serina, I., *Lpg: A planner based on local search for planning graphs with action costs*. in AIPS, 2002.

89. Gerevini, A., Saetti, A. and Serina, I., *Planning in PDDL2.2 domains with LPG-TD*. in Proceedings of the International Planning Competition,14th International Conference on Automated Planning and Scheduling, 2004.

90. Kautz, H. and Selman, B., *Pushing the envelope: planning, propositional logic, and stochastic search*. in Proceedings of the Thirteenth National Conference on Articial Intelligence and the Eighth Innovative Applications of Articial Intelligence Conference, 1996, Portland, Oregon: AAAI Press.

91. Kautz, H. and Selman, B., *The role of domain-specific knowledge in the planning as satisfiability framework.* Artificial Intelligence Planning Systems, 1998, p. 181–189.

92. Huang, Y.-C., Selman, B. and Kautz, H., *Control knowledge in planning: benefits and tradeoffs*. in Proceedings of the sixteenth national conference on Artificial Intelligence and eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence, 1999, Orlando, Florida, United States: American Association for Artificial Intelligence.

93. Berardi, D., Calvanese, D., De-Giacomo, G., Lenzerini, M. and Mecella, M., *e-Service Composition by Description Logics Based Reasoning*. in Proceedings of the International Workshop on Description Logics (DL03), 2003, Rome, Italy.

94. Petri, C.A., *Kommunikation mit Automaten*. 1962, Institut für instrumentelle Mathematik.

95. Hamadi, R. and Benatallah, B., *A Petri net-based model for web service composition*. in Proceedings of the 14th Australasian database conference - Volume 17, 2003, Adelaide, Australia: Australian Computer Society, Inc.

96. Dimopoulos, Y., Nebel, B. and Koehler, J., *Encoding planning problems in nonmonotonic logic programs*. in Proceedings of the Fourth European Conference on Planning, 1997.

97. Sacerdoti, E.D., *Planning in a hierarchy of abstraction spaces.* Artificial Intelligence, 1974, **5**(2), p. 115–135.

98. Erol, K., Hendler, J. and Nau, D.S., *Semantics for HTN planning.* 1994, Technical Report CS-TR-3239.

99. Erol, K., Hendler, J. and Nau, D.S., *UMCP: A sound and complete procedure for hierarchical task-network planning.* Artificial Intelligence Planning Systems, 1994, p. 249–254.

100. Nau, D., Cao, Y., Lotem, A. and Munoz-Avila, H., *SHOP: simple hierarchical ordered planner*. in Proceedings of the 16th international joint conference on Artificial Intelligence, 1999, Stockholm, Sweden: Morgan Kaufmann Publishers Inc.

101. Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D. and Yaman, F., *SHOP2: an HTN planning system.* Journal of Artificial Intelligence Research, 2003, **20**, p. 379–404.

102. Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F. and Scherl, R.B., *Golog: a logic programming language for dynamic domains.* The Journal of Logic Programming, 1997, **31**(1-3), p. 59–83.

103. Giacomo, D.G., Lespérance, Y. and Levesque, H.J., *ConGolog, a concurrent programming language based on the situation calculus.* Artificial Intelligence, 2000, **121**(1-2), p. 109–169.

104. Cimatti, A., Giunchiglia, E., Giunchiglia, F. and Traverso, P., *Planning via model checking: A decision procedure for AR*. in Proceedings of the 4th European Conference on Planning, 1997, Berlin / Heidelberg: Springer.

105. Giunchiglia, F. and Traverso, P., *Planning as Model Checking*. in Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning, 2000, Springer Berlin / Heidelberg.

106. Daniele, M., Traverso, P. and Vardi, M., *Strong Cyclic Planning Revisited*. in Recent Advances in AI Planning, S. Biundo and M. Fox, Editors. 2000, Springer Berlin / Heidelberg, p. 35–48.

107. Cimatti, A., Roveri, M. and Traverso, P., *Automatic OBDD-based generation of universal plans in non-deterministic domains*. in Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, 1998, Madison, Wisconsin, United States: American Association for Artificial Intelligence.

108. Edelkamp, S. and Helmert, M., *The implementation of Mips.* 2000.

109. Fourman, M., *Propositional planning.* 2000.

110. Hölldobler, S. and Störr, H.-P., *Solving the Entailment Problem in the Fluent Calculus Using Binary Decision Diagrams*. in Proceedings of the First International Conference on Computational Logic, 2000, Springer-Verlag.

111. Bertoli, P., Cimatti, A., Pistore, M., Roveri, M. and Traverso, P., *Mbp: a model based planner.* 2001.

112. Jensen, R.M. and Veloso, M.M., *OBDD-based universal planning for synchronized agents in non-deterministic domains.* Journal of Artificial Intelligence Research, 2000, **13**(1), p. 189–226.

113. Bacchus, F. and Kabanza, F., *Planning for temporally extended goals*. in Proceedings of the thirteenth national conference on Artificial intelligence, 1996, Portland, Oregon: AAAI Press.

114. Bacchus, F. and Ady, M., *Planning with resources and concurrency: a forward chaining approach*. in Proceedings of the 17th international joint conference on Artificial intelligence, 2001, Seattle, WA, USA: Morgan Kaufmann Publishers Inc.

115. Bacchus, F. and Kabanza, F., *Using temporal logic to control search in a forward chaining planner.* in Proccedings of Second International Workshop on Temporal Representation and Reasoning (TIME), 1995, Melbourne Beach, Florida.

116. Alur, R., Feder, T. and Henzinger, T.A., *The benefits of relaxing punctuality.* Journal of the ACM, 1996, **43**(1), p. 116–146.

117. Kvarnstrom, J. and Doherty, P., *TALplanner: A temporal logic based forward chaining planner.* Annals of Mathematics and Artificial Intelligence, 2001, **30**(1-4), p. 119–169.

118. Haigh, K.Z., *Situation Dependent Learning for Interleaved Planning and Robot Execution.* 1998.

119. Peer, J., *A PDDL based Tool for Automatic Web Service Composition*. in Proceedings of the Second Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2004) at the 20th International Conference on Logic Programming, 2004.

120. Kuter, U., *Planning under uncertainty: moving forward*. 2006, Thesis at the University of Maryland at College Park, USA.

121. Ko, R.K.L., Lee, S.G., Lee, E.W. and Jusuf, A., *Dynamic Collaborative Business Process Formulation via Ontologised Hierarchical Task Network (HTN) Planning.* Artificial Intelligence, 2009, **13**(15).

122. Papazoglou, M. and Yang, J., *Design Methodology for Web Services and Business Processes*. in Technologies for E-Services, A. Buchmann, L. Fiege, F. Casati, M.-C. Hsu, and M.-C. Shan, Editors. 2002, Springer Berlin / Heidelberg, p. 175–233.

123. Sirin, E., *OWL-S API*. 2004; Available from: http://www.mindswap.org/2004/owl-s/api/.

124. White, S., *XPDL and BPMN*. 2003.

125. Au, T.C., Nau, D. and Subrahmanian, V.S., *Utilizing Volatile External Information during Planning*. in Proceedings of the European Conference on Artificial Intelligence (ECAI), 2004.

126. *Phoenix*. Available from: http://www.staff.vu.edu.au/phoenix/phoenix/index1.htm.

Appendices

Appendix A: List of Abbreviations

B2B – Business to Business

BDD – Binary Decision Diagrams

BP – Business Process

BPMN – Business Process Model and Notation

CPM – Collaborative Process Manager

CSDL – Composite Service Definition Language

CTL – Computation Tree Logic

CWGM – Collaboration and Workflow Generation Manager

ER – Entity-Relation

FF – Fast Forward

HSP – Heuristic Search Planner

HTN – Hierarchical Task Network

JBP – Joint Business Process

JD – Joint Domain

JSHOP2 – Java Simple Hierarchical Ordered Planner 2

MBP – Model Based Planner

MITL – Metric Interval Temporal Logic

PBM – Planning by Model Checking

PDDL – Planning Domain Definition Language

PIPs – Partner Interface Processes

POCL – Partial Order Causal Planners

POP – Partial Ordered Planners

RIFO – Removing Irrelevant Operators and Initial Facts from Planning Problems

SFTP – Secure File Transfer Protocol

SGP – Sensory Graphlan

SHOP – Simple Hierarchical Ordered Planner

STAN – STate ANalysis

UMOP – Universal Multi-agent Obdd-based Planner

VHPOP – Versatile Heuristic Partial Order Planner

WfMC – Workflow Management Coalition

WfMS – Workflow Management System

WF-nets – Workflow Nets

WSDL – Web Service Definition Language

XPDL – XML Process Definition Language

Appendix B: WfMS Products

Currently there are more than 150 vendors providing services in workflow and business process management. Following are some of the most highly reputed and widely used WfMS products.

- WebSphere MQ Workflow is a workflow engine based on object oriented design and client server architecture. It uses Active X objects and java APIs for modelling activities. It uses forms and portlets to interact with users. It integrates business process analysis, simulation and development tool with a comprehensive monitoring environment. The ability of this product to create JSP files automatically from workflow definitions make it highly usable with the web technology.

- FileNet P8 BPM Suite is the J2EE transformation of FileNet. It is basically for P8 platforms and it powers the distributed architecture and EAI capabilities of P8 platforms. A Java/COM API has been developed in this product to power tailored development and integration. FileNet P8 BPM Suite uses a web browser based adhoc capable process definition tool. It has a production capable process model and the capability to cooperate with the native Content Manager and Web Content Manager.

- Staffware Process Suite (SPS) offers high productions and is used to automate processes that lie at the centre of administrative processes and production processes. SPS uses a form definition tool and a scripting language for interactive processes; it uses EAI adapters, SQL database accessibility and Tuxedo transactions for automatic processes. Its process monitoring tool takes care of both operational and management needs.

- TIBCO Inconcert enables user to easily build workflows and modify them on the fly. It is based on the integration of object oriented technology and document management. It uses TIBCO Integration Manager Orchestration engine and TIBCO Rendezvous messaging for application integration. Its BPM designer

supports Inconcert process definitions and Integration manager orchestration definitions.

- Enhydra Shark Workflow is the most widely known opensource WfMS. It is based on a WfMC's specifications compliant Java/XML workflow engine framework. It adopts XML Process Definition Language (XPDL) for modelling workflows. It has a Java based process editor JaWE which is compliant to XPDL. The latest version of Enhydra Shark has a graphical administration tool for workflow enactment management. Icube's Openflow, jBPM and JBoss are some of the other popular opensource workflow engines.

Appendix C: IssueInspCert (OWLS)

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:process="http://www.daml.org/services/owl-s/1.1/Process.owl#"
   xmlns:grounding="http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
   xmlns:service="http://www.daml.org/services/owl-s/1.1/Service.owl#"
   xmlns:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xml:base="http://158.125.103.196/OWLS%20processes/Vendor/IssueInspCert.owl ">

 <!-- Service description -->
 <service:Service rdf:ID="IssueInspCertService">
          <service:presents rdf:resource="#IssueInspCertProfile"/>
          <service:describedBy rdf:resource="#IssueInspCertProcessModel"/>
          <service:supports rdf:resource="#IssueInspCertGrounding"/>
 </service:Service>

 <!-- Profile description -->
 <profile:Profile rdf:ID="IssueInspCertProfile">
          <service:isPresentedBy rdf:resource="#IssueInspCertService"/>
          <profile:serviceName xml:lang="en">Issuing Inspection Certificate</profile:serviceName>
          <profile:textDescription xml:lang="en">This service issues inspection  certificate.
 </profile:textDescription>
          <profile:hasInput rdf:resource="#ok_Insp"/>
          <profile:hasOutput rdf:resource="#InspCert"/>
 </profile:Profile>

 <!-- Process Model description -->
 <process:ProcessModel rdf:ID="IssueInspCertProcessModel">
          <service:describes rdf:resource="#IssueInspCertService"/>
          <process:hasProcess rdf:resource="#IssueInspCertProcess"/>
 </process:ProcessModel>

 <process:AtomicProcess rdf:ID="IssueInspCertProcess">
          <process:hasInput rdf:resource="#ok_Insp"/>
          <process:hasOutput rdf:resource="#InspCert"/>
 </process:AtomicProcess>

 <process:Input rdf:ID="ok_Insp">
          <process:parameterType rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
          <rdfs:label>Presale Inspection Successful</rdfs:label>
 </process:Input>

 <process:Output rdf:ID="InspCert">
          <process:parameterType rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
          <rdfs:label>Inspection Certificate</rdfs:label>
 </process:Output>

 <!-- Grounding description -->
 <grounding:WsdlGrounding rdf:ID="IssueInspCertGrounding">
          <service:supportedBy rdf:resource="#IssueInspCertService"/>
          <grounding:hasAtomicProcessGrounding rdf:resource="#IssueInspCertProcessGrounding"/>
 </grounding:WsdlGrounding>

  <grounding:WsdlAtomicProcessGrounding rdf:ID="IssueInspCertProcessGrounding">
```

```
        <grounding:owlsProcess rdf:resource="#IssueInspCertProcess"/>
        <grounding:wsdlDocument>
                http://158.125.103.196/OWLS%20processes/Vendor/IssueInspCert.wsdl
        </grounding:wsdlDocument>
        <grounding:wsdlOperation>
      <grounding:wsdlOperationRef>
        <grounding:portType>
                http://158.125.103.196:8080/IssueInspCert/services/IssueInspCert/IssueInspCertHttpSoap11Endpoint
        </grounding:portType>
        <grounding:operation>
                http://158.125.103.196:8080/IssueInspCert/services/IssueInspCert/IssueInspCert
        </grounding:operation>
      </grounding:wsdlOperationRef>
    </grounding:wsdlOperation>
        <grounding:wsdlInputMessage>
              http://158.125.103.196:8080/IssueInspCert/services/IssueInspCert.IssueInspCertRequest
        </grounding:wsdlInputMessage>
    <grounding:wsdlInputMessageParts rdf:parseType="Collection">
      <grounding:wsdlMessageMap>
       <grounding:owlsParameter rdf:resource="#ok_Insp"/>
       <grounding:wsdlMessagePart>ok_Insp</grounding:wsdlMessagePart>
      </grounding:wsdlMessageMap>
    </grounding:wsdlInputMessageParts>
        <grounding:wsdlOutputMessage>
                http://158.125.103.196:8080/IssueInspCert/services/IssueInspCert.IssueInspCertResponse
        </grounding:wsdlOutputMessage>
        <grounding:wsdlOutputMessageParts rdf:parseType="Collection">
         <grounding:wsdlMessageMap>
          <grounding:owlsParameter rdf:resource="#InspCert"/>
          <grounding:wsdlMessagePart>InspCert</grounding:wsdlMessagePart>
         </grounding:wsdlMessageMap>
    </grounding:wsdlOutputMessageParts>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Appendix D: IssueInspCert (WSDL)

```
<wsdl:definitions xmlns:axis2="http://vendor" xmlns:ns1=http://org.apache.axis2/xsd
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:http=http://schemas.xmlsoap.org/wsdl/http/
xmlns:ns0="http://vendor/xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" targetNamespace="http://vendor">
<wsdl:documentation>This service issues an inspection certificate</wsdl:documentation>
<wsdl:types>
 <xs:schema xmlns:ns="http://vendor/xsd" attributeFormDefault="qualified"
  elementFormDefault="qualified"  targetNamespace="http://vendor/xsd">
  <xs:element name="IssueInspCert">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="ok_Insp" nillable="true" type="xs:string" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="IssueInspCertResponse">
   <xs:complexType>
    <xs:sequence>
      <xs:element name="InspCert" nillable="true" type=" xs:string " />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:schema>
</wsdl:types>
 <wsdl:message name="IssueInspCertMessage">
  <wsdl:part name="part1" element="ns0:IssueInspCert" />
 </wsdl:message>
 <wsdl:message name="IssueInspCertResponse">
  <wsdl:part name="part1" element="ns0:IssueInspCertResponse" />
 </wsdl:message>
 <wsdl:portType name="IssueInspCertPortType">
   <wsdl:operation name="IssueInspCert">
   <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
      message="axis2:IssueInspCertMessage" wsaw:Action="urn:IssueInspCert" />
   <wsdl:output message="axis2:IssueInspCertResponse" />
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="IssueInspCertSOAP11Binding" type="axis2:IssueInspCertPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
 <wsdl:operation name="IssueInspCert">
  <soap:operation soapAction="urn:IssueInspCert" style="document" />
   <wsdl:input>
    <soap:body use="literal" />
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal" />
   </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="IssueInspCertSOAP12Binding" type="axis2:IssueInspCertPortType">
 <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
```

```
  <wsdl:operation name="IssueInspCert">
   <soap12:operation soapAction="urn:IssueInspCert" style="document" />
   <wsdl:input>
    <soap12:body use="literal" />
   </wsdl:input>
   <wsdl:output>
    <soap12:body use="literal" />
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:binding name="IssueInspCertHttpBinding" type="axis2:IssueInspCertPortType">
  <http:binding verb="POST" />
  <wsdl:operation name="IssueInspCert">
    <http:operation location="IssueInspCert" />
   <wsdl:input>
    <mime:content type="text/xml" />
   </wsdl:input>
   <wsdl:output>
    <mime:content type="text/xml" />
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="IssueInspCert">
  <wsdl:port name="IssueInspCertSOAP11port_http" binding="axis2:IssueInspCertSOAP11Binding">
   <soap:address location="http://158.125.103.196:8080/Vendor/services/IssueInspCert" />
  </wsdl:port>
  <wsdl:port name="IssueInspCertSOAP12port_http" binding="axis2:IssueInspCertSOAP12Binding">
   <soap12:address location="http://158.125.103.196:8080/Vendor/services/IssueInspCert" />
  </wsdl:port>
  <wsdl:port name="IssueInspCertHttpport" binding="axis2:IssueInspCertHttpBinding">
   <http:address location="http://158.125.103.196:8080/Vendor/services/IssueInspCert" />
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Appendix E: IssueInspCert (Java)

```
package vendor;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

public class IssueInspCert {
        public String IssueInspCert ( String ok_Insp ) throws IOException
          {

                BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
                String file_name = "D:\\PHD\\Runtime Section\\Certificates\\InspCert.txt";
                File file = new File(file_name);
                file.createNewFile();
                FileWriter fstream = new FileWriter(file_name);
                BufferedWriter out = new BufferedWriter(fstream);
                out.write("This is a Sample Inspection Certificate created by Vendor" + “after “ +
                        ok_Insp + “signalled the successful completion of factory inspection” );
                out.close();
                System.out.println("Vendor: Issuing Inspection Certificate");
                return file.getPath();

          }
}
```

Appendix F: All Sets of Compatible Workflows for Vendor/Customer Business Collaboration Example

Initial States: [*Payment*]

Goal States: [*ok_PH, s_InvPay*]

Data Dependencies: Following are the data dependencies among the activities of the workflows. The data dependencies remain the same for all sets of compatible workflows.

Inv_s, *ShippingArrangement*, *InsuranceArrangement* and *CertOriginApp* have data dependency on *IssueInv*.

CertOrigin_s has data dependency on *CertOriginApp*

BL_s has data dependency on *ShippingArrangement*

InspCert_s has data dependency on *IssueInspCert*

InsuCert_s has data dependency on *InsuranceArrangement*

CheckInspCert has data dependency on *InspCert_r*

TakeDelivery has data dependency on *BL_r* and *Inv_r*

ApprovePayment has data dependency on *CertOrigin_r* and *Inv_r*

Control Dependencies: The sequential order of activities represented by a comma "," represents the control dependencies in the workflows. The execution starts from the customer workflow. The execution mechanism waits on the receiving activities until the respective sending activity has been executed.

Generated Sets of Compatible Workflows

Vendor Workflow 1:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, Inv_s, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 1:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, BL_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 2:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, Inv_s, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 2:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, BL_r, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 3:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, Inv_s, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 3:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, BL_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 4:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, Inv_s, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 4:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, BL_r, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 5:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, Inv_s, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 5:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, BL_r, Inv_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 6

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, Inv_s, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 6:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, BL_r, Inv_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 7:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, Inv_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 7:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, BL_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 8:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, Inv_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 8:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, BL_r, Inv_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 9:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, Inv_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 9:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, BL_r, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 10:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, Inv_s, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 10:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, BL_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 11:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, Inv_s, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 11:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, BL_r, Inv_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 12:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, Inv_s, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 12:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, BL_r, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 13:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, Inv_s, ShippingArrangement, BL_s,

InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 13:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, InsuCert_r, BL_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 14:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, Inv_s, BL_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 14:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, InsuCert_r, BL_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 15:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, Inv_s, InsuranceArrangement, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 15:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, InsuCert_r, BL_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 16:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, Inv_s, InsuCert_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 16:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, InsuCert_r, BL_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 17:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, Inv_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 17:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, BL_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 18:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, Inv_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 18:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, BL_r, Inv_r, CustomsDeclaration, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 19:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, Inv_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 19:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, BL_r, InsuCert_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Vendor Workflow 20:

[AdvPay_r, PaymentCheck, GoodsManufacture, IssueInv, FactoryInspection, IssueInspCert, InspCert_s, SA_r, ShippingArrangement, BL_s, InsuranceArrangement, InsuCert_s, Inv_s, CertOriginApp, CertOrigin_s, InvPay_r, PaymentHandling]

Customer Workflow 20:

[AdvPay_s, InspCert_r, CheckInspCert, IssueSA, SA_s, Inv_r, CustomsDeclaration, InsuCert_r, BL_r, TakeDelivery, PresaleInspection, CertOrigin_r, ApprovePayment, InvPay_s]

Appendix G: All Sets of Compatible Workflows for Retailer/Wholesaler/Manufacturer/Supplier Business Collaboration Example

InitialStates: [*goodsreq*]

GoalStates: [*s_RInvPay, r_RInvPay, r_WInvPay, r_MInvPay*]

Data Dependencies: Following are the data dependencies among the activities of the workflows. The data dependencies remain the same for all sets of compatible workflows.

SendQuotation_s has a data dependency on *QuotationPrep*

UpdateRecords and *CommercialInvoice_s* has a data dependency on *IssueInv*

Quotation_s has a data dependency on *PrepareQuotation*

QuotationApp has a data dependency on *ReceiveQuotation_r*

DeclaretoCustoms and *TakeRawDelivery* has a data dependency on *CommercialInvoice_r*

Invoice_s has a data dependency on *CreateInvoice*

InsuCert_s has a data dependency on *ArrangeInsurance*

Quotation_s has a data dependency on *QuotationPreparation*

ApproveQuotation has a data dependency on *Quotation_r*

CustomsDeclaration has a data dependency on *InsuCert_r*

TakeDelivery has a data dependency on *Invoice_r*

ComInv_s has a data dependency on *IssueComInv*

QuotationEvaluation has a data dependency on *Quotation_r*

TakeDelivery has a data dependency on *ComInv_r*

Control Dependencies: The sequential order of activities represented by a comma "," represents the control dependencies in the workflows. The execution starts from the *Retailer's* workflow. The execution mechanism will wait on the receiving activities until the respective sending activities have been executed.

Generated Sets of Compatible Workflows

Retailer Workflow 1:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 1:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 1:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 1:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, AssembleGoods, InsureRaw, InsuranceCertificate_s, ShipRaw, Documentation, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 2:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 2:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 2:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 2:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, InsureRaw, AssembleGoods, InsuranceCertificate_s, ShipRaw, Documentation, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 3:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 3:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 3:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 3:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, AssembleGoods, InsureRaw, ShipRaw, InsuranceCertificate_s, Documentation, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 4:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 4:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 4:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 4:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, AssembleGoods, ShipRaw, InsureRaw, InsuranceCertificate_s, Documentation, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 5:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 5:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 5:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 5:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, InsureRaw, AssembleGoods, ShipRaw, InsuranceCertificate_s, Documentation, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 6:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 6:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 6:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 6:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, InsureRaw, InsuranceCertificate_s, AssembleGoods, ShipRaw, Documentation, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 7:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 7:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 7:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 7:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, AssembleGoods, InsureRaw, ShipRaw, Documentation, InsuranceCertificate_s, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 8:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 8:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 8:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 8:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, AssembleGoods, ShipRaw, InsureRaw, Documentation, InsuranceCertificate_s, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 9:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 9:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 9:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 9:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, AssembleGoods, ShipRaw, Documentation, InsureRaw, InsuranceCertificate_s, UpdateRecords, PaymentInvoice_r]

Retailer Workflow 10:

[QuotationInqPrep, QuotationInq_s, Quotation_r, QuotationEvaluation, CreatePO, PO_s, POAcpt_r, ComInv_r, TakeDelivery, ApprovePayment, InvPayment_s]

Wholesaler Workflow 10:

[QuotationInq_r, QuotationPreparation, Quotation_s, PO_r, POApproval, POAcpt_s, CreateInquiry, QuotationInquiry_s, Quotation_r, ApproveQuotation, QuotationApproval_s, Invoice_r, InsuCert_r, CustomsDeclaration, TakeDelivery, PaymentApproval, InvoicePayment_s, IssueComInv, ComInv_s, ShipGoods, InvPayment_r]

Manufacturer Workflow 10:

[QuotationInquiry_r, PrepareQuotation, Quotation_s, QuotationApproval_r, PrepareInquiry, QuotationInquiry_s, ReceiveQuotation_r, QuotationApp, QuotationApp_s, CommercialInvoice_r, InsuranceCertificate_r, DeclaretoCustoms, TakeRawDelivery, ApprovePaymentInvoice, PaymentInvoice_s, GoodsManufacturing, CreateInvoice, Invoice_s, ArrangeShipment, ArrangeInsurance, InsuCert_s, InvoicePayment_r]

Supplier Workflow 10:

[QuotationInquiry_r, QuotationPrep, SendQuotation_s, QuotationApp_r, IssueInv, CommercialInvoice_s, InsureRaw, AssembleGoods, ShipRaw, Documentation, InsuranceCertificate_s, UpdateRecords, PaymentInvoice_r]

Appendix H: List of Published Papers

Saleem, M., Chung, P.W.H., Fatima, S. and Dai, W., *Intelligent Business Transaction Agents for Cross-Organizational Workflow Definition and Execution*, in Proceedings of Intelligent Information Processing V: 6th IFIP International Conference on Intelligent Information, 2010, p. 245–250.

Saleem, M., Chung, P.W.H., Fatima, S. and Dai, W., *Cross Organisational Compatible Plans Generation Framework*, in Proceedings of AI-2011 Thirty-first SGAI International Conference on Artificial Intelligence, Cambridge, England, 2011, p. 223–228**.**