A cross organisation compatible workflows generation and execution framework

A Cross Organisation Compatible Workflows Generation and Execution Framework

Mohammad Saleem[1], [2], Paul W.H. Chung[2], Shaheen Fatima[2], Wei Dai[3],

Abstract

With the development of the Internet, the demand for electronic and online commerce has increased. This has, in turn, increased the demand for business process automation. In this paper, we look at the use of workflows for business process automation. An automatically generated workflow can save time and resources needed for running online businesses. In general, due to the interdependencies between their activities, multiple business organisations will need to work together by collaborating and coordinating their activities with each other. This gives rise to the need for workflow collaboration across organisations. Current systems for workflow collaboration are only capable of reconciling existing workflows of the collaborating organisations. Automatic workflow generation systems only generate workflows for individual organisations and cannot handle the automatic generation of compatible workflows for multiple collaborating organisations. To overcome this problem, in this paper, we present a framework that is able to generate multiple sets of compatible workflows for multiple collaborating organisations. The proposed framework supports runtime enactment and runtime collaboration of the generated workflows. This framework enables users to save the time and resources that would otherwise be spent in modelling, reconciling and reengineering workflows.

1 Introduction

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*" (Workflow

[1] Computer Science Department, Loughborough University, Loughborough, LE11 3TU, UK
{P.W.H.Chung, S.S.Fatima}@lboro.ac.uk

[2] Department of Computer Science, CECOS University, Peshawar, Pakistan
saleem@cecos.edu.pk

[3] School of Management and Information Systems, Victoria University, Melbourne, Victoria, Australia
Wei.Dai@vu.edu.au

Management Coalition, 1999). It means that a business process is essential for the business goals of organisations. Workflow is the technology used to model automated business processes. According to Workflow Management Coalition'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*" (Workflow Management Coalition, 1999). A workflow has two main stages (Workflow Management Coalition, 1999):

- Build-time stage – this refers to the stage where workflow descriptions of the business process are defined or changed. This can be automatic or manual.
- Execution stage – this is where instances of the business process are created, executed and managed. This is the operational stage.

In the real world, organisations have to interact with other organisations to do business. For any two organisations to proceed in business, they need to have compatible workflows and compatible means that there should be an agreed sequence of activities exchanging collaborative messages and information (Chen and Chung, 2008). The point where exchange of collaborative messages and information takes place between two collaborating workflows is called an interface activity. An interface activity can be a sending activity or a receiving activity. The set of all interface activities in a workflow is called interface process (Chen and Chung, 2008). The proposed framework uses the idea of interface activities for collaboration among the interacting organisations. Interface activities decouple the collaborating workflows. A receiving activity only has to know the details of the corresponding sending activity and does not need the representational details of the entire collaborative workflow.

When two of more organisations do business together, the need for workflow collaboration across multiple organisations arises (Chen and Chung, 2007). Such collaboration is referred to as cross organisational workflow collaboration. Incompatible workflows should be reconciled before proceeding with business. Considerable amount of effort is needed to ensure that workflows are compatible (Chiu *et al.*, 2004; Schulz and Orlowska, 2004) and proceed into the execution stage.

[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 paper.

Recent research on workflow collaboration focuses on reconciling existing incompatible workflows (Krukkert, 2003; Chen, 2008). This is a bottom-up approach. In an alternative top-down approach, organisations meet, discuss and design collaborative processes and then implement them (Chen and Chung, 2007). Both of these approaches are time consuming, especially, if an organisation has many business partners to collaborate with. Every time an organisation has to collaborate with another organisation, both the organisations will have to invest a lot of time and resources to come up with compatible workflows. In case of any change to the workflow of an organisation, negotiations may have to be done all over again with all the collaborating organisations.

Another paradigm in the literature is automatic workflow generation, which is based on AI planning where a workflow is considered as a plan (Chen and Yang, 2005; Dong and Wild, 2008). 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 (Saleem, 2012). Web services are self-contained units of application logic (Srivastava and Koehler, 2003), which can be can be discovered, connected to and executed over the internet. Therefore workflow generation can be treated as a web services composition problem (Dong and Wild, 2008). In web services composition, a planner reasons about a pool of available services and the service that can bring about the desirable effect is added in the plan. Executing the plan will result in the goal state (Chen and Yang, 2005).

In web service composition, the planning system requires formal domain ontology for planning. Domain ontology refers to the formal representation of the environment where planning takes place (Chen and Yang, 2005). A web service composition environment is primarily a collection of web services, so the domain ontology is in the form of web services descriptions (Saleem, 2012).

Existing automatic workflow generation systems automatically generate workflows for single organisations only and cannot generate compatible workflows for multiple collaborating organisations (Sirin *et al.*, 2003; Sirin *et al.*, 2004; Okutan and Cicekli, 2010). The proposed framework automatically generates compatible workflows for multiple collaborating organisations to meet their high-level goals, without the organisations having to model their workflows beforehand.

As the proposed framework independently generates compatible workflows for each collaboration scenario among collaborating organisations, the organisations do not have to

worry about keeping their workflows compatible with the other organisations they interact with. Unlike the approach presented in this paper, in both manual and automatic workflow collaboration negotiations, the organisations have to change their workflows in such away that it becomes compatible with the workflow of the negotiating organisation and at the same time remains acceptable to the existing organisations it is interacting with. Thus the proposed framework eliminates the need for the time consuming negotiations that might otherwise have been necessary to reconcile incompatible workflows.

The framework uses SHOP2 for planning because SHOP2 supports complex domains, extended goals and non-deterministic actions (Peer, 2005). Furthermore, it is a highly efficient planning system and has a Web Ontology Language for Services (OWLS) type mechanism for representing atomic tasks and decomposing composite tasks into atomic tasks (Sirin *et al.*, 2004). The similar mechanism of OWLS and SHOP2 to hierarchically decompose complex tasks into sub tasks makes it straightforward to map OWLS definitions directly into SHOP2 domain (Sirin *et al.*, 2004; Wu *et al.*, 2003) and create workflows based on the translated domain.

SHOP2 is a hierarchical task network (HTN) planner. It requires domain knowledge for planning. The OWLS web services descriptions can be translated to create the SHOP2 domain. The SHOP2 domain consists of operators and methods. Operators are atomic tasks that can be executed directly. Methods are specifications to decompose complex tasks into atomic tasks. SHOP2 is a substantially expressive planner (Sirin and Parsia, 2004). The expressivity of SHOP2 is similar to Planning Domain Definition Language (PDDL) (Sirin *et al.*, 2004). In the context of semantic web services, PDDL is neither too restrictive nor too expressive and is considered as a viable compromise between expressivity and efficiency (Peer and Vokovic, 2005). It uses a restricted subset of first order logic to describe the semantics of operations. SHOP2 supports logical connectives such as conjunction, disjunction, implication, negation and universal quantification to combine logical atoms. SHOP2 supports the evaluation of arbitrary code at planning time through complex precondition reasoning. This makes it possible to integrate existing knowledge bases on the semantic web into SHOP2 domain (Sirin and Parsia, 2004).

Business Process Model and Notation (BPMN) language is the most widely used standard to represent workflows (Meng *et al.*, 2012). It defines the notation and semantics of business processes (OMG, 2011). It is a workflow modelling language (OMG, 2011) and lacks the

semantic precision required for automatic business process generation and execution (Ouyang *et al.*, 2008), and so we cannot use it as a notation for the proposed framework. Automatic workflow generation can be achieved by exploiting web services composition. OWLS is a language for describing web services (OWL Services Coalition, 2003). It is used to describe the functionality, access point, execution mechanism and compositional capabilities of web services. In OWLS, each service is modelled as a process (Sirin *et al.*, 2004). A process can be atomic, simple or composite. OWLS is a set of ontologies and OWLS 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 OWLS web services descriptions to SHOP2 domain (Sirin *et al.*, 2004).

In the rest of the paper, section 2 outlines the related work. Section 3 discusses the assumptions made for the proposed framework. Section 4 gives the general architecture of the framework. Section 5 presents the functionality of the framework and explains the major algorithms involved. Section 6 discusses implementation details. Section 7 describes application examples and Section 8 discusses the paper and highlights future work.

2 Related Work

Most of the existing work on cross organisational workflow collaboration in the literature deals with build time collaboration. Van-der-Aalst and Weske (2001) 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 between the interacting organisations; and in the third step, the organisations create their private workflows autonomously. This approach requires manual negotiations to reach an agreement, which can be very time consuming especially if there are many partners.

Krukkert (2003) 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. 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, which is a limitation of the system. In such cases, manual

changes need to be done to the activity diagrams in order to introduce a common sequence path. Alternatively, a third party collaboration system is required to bring about collaboration. Both cases undermine the benefit of using Krukkert's soloution.

Chen (2008) presented an approach for reconciling existing workflows to bring about compatibility. 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 counter-offer. The partner then either accepts or rejects the offer. The process of offer generation, counter-offer generation, acceptance and rejection goes on recursively till the negotiation is terminated or reconciliation is achieved.

Since workflows need to be executed, there needs to be runtime collaboration so that the transfer of files and information happens smoothly among the in-house and cross organisational activities. Chen and Chung (2006) presented a bottom-up cross organisational workflow enactment approach. The approach is workflow management system (WfMS) independent and the enactment is done via progressive linking enabled by runtime agents. Each interaction point in the collaborating workflows 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 form represents the progress of interoperation and can be used for historical record.

With the increase in demand for reusability and interoperability, research has considered composing web services into composite services to automatically generate business processes. Sirin *et al.* (2003) created a semi-automatic web services composition system, which allows users to select from a list of web services at each step of composition. The user starts the composition process by selecting one of the services registered with the system. The system then checks for web services that can satisfy the selected service, presents them to the user and the user selects one to add in the plan. The system then checks for web services that satisfy the next requirement. The process continues until the composition completes.

Later, Sirin *et al.* (2004) extended their semi-automatic web service composition system to a fully automatic system. They implemented an OWLStoSHOP2 translator to translate collections of OWLS process definitions into SHOP2 domain. 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 generated SHOP2 plan is converted to OWLS format by a plan converter called SHOP2toOWL, and executed by the Execution System. A

limitation with this system is that it plans for a single organisation only and does not take collaboration among multiple business organisations into account.

Okutan and Cicekli (2010) proposed an event calculus based web service composition and execution (WSCE) system. The system has two phases, namely composition phase and execution phase. In the composition phase, the OWLS process definitions are translated to axioms in event calculus domain. Web services are encoded as actions, web service inputs and outputs as action's knowledge preconditions and knowledge effects, and web service preconditions and effects as action preconditions and effects. The user inputs are substituted as initial condition axioms and the outputs as goals. Based on the domain knowledge, the Abductive Event Calculus Planner generates plans to reach the given goal state. The plans are presented to the user in the form of visual graphs, which can be sorted according to user's quality of service parameter among *execution duration*, *price*, *reliability* and *availability*. In the execution phase, the selected graph is transformed to OWLS descriptions and passed to the execution engine. The user enters the actual input values, and the actual web services modelled by the OWLS processes are invoked.

WSCE is a good effort to use event calculus for web service composition. The main benefit of this system is that it supports concurrent plans and so it is better suited for solving real world business scenarios. The main issue with this system is that it can compose workflows for a single organsisation only and does not take the generation of collaborative workflows for multiple organizations into account. The work, if extended for solving multi-organisation scenarios, can be a good addition to research.

Recently, there has been some work on composing workflows for multiple organisations. Chen *et al.* (2011) suggested a Pi-Calculus based approach to compose web services into cross organisational business processes. A cross organisational business processes is modelled as a set of concurrent local processes, which has a global start and a global end activity. The activities in the local processes can receive external start messages. A cross organisational controller controls the flow of control and data in the cross organisational process. The limitation with this work is that it uses a manual modelling approach and the web services composition is not automatic.

Correˆa da Silva *et al.* (2013) presented a lightweight, flexible and user-friendly platform for cross organisational workflow interactions. The platform is named JamSession and it can be considered as a meeting point for already existing software components to form new and

innovative service systems. JamSession is a user-friendly and light-weight platform, providing an appropriate framework to specify and implement cross organisational workflow interactions. It uses knowledge-based interaction protocols and predicates to models cross organisational workflows and activities in such away that the workflow definitions are local to the respective workflow management systems, and only the interaction protocols are made public. It makes the workflows highly decoupled. While the paper claims that the interaction protocols can be used to specify and execute cross organisational workflows, it only shows examples for the execution of cross organisational workflows. So, it is not possible to deduce whether the interaction protocols can be used for bringing about collaboration among cross organisational workflows at build time.

Problem Solving Methods (PSMs) (Crubezy, 2003) is another area that has a conceptual resemblance to web service composition, due to its focus on reusable domain-independent reasoning about ontologies (Elenius, 2004). In PSMs, the properties of a method can be specified as a *method* ontology. With the help of *mapping ontologies*, the inputs and outputs of the PSM can be connected to the entities in the ontologies of different domains. The use of the idea of PSM and *mapping ontologies* can be interesting in web services composition domain.

3 Assumptions

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

1. It is assumed that the collaborating organisations follow OWLS ontology for services, as OWLS is the most widely used standard specification for adding semantics to web services (Dong and Wild, 2008). OWLS provides a standard set of ontologies to the collaborating organisations for describing and composing web services. Apart from the service ontology, the collaborating organisations only need to follow the same domain ontology for the inputs/outputs/preconditions/effects that are not local to a single organisation and are used by multiple collaborating organisations.

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

3. It is assumed that during workflow generation and execution, the world does not change as a result of the actions of another agent and the initial state contains all the necessary information of the domain for the planning to be done. It is assumed that the services are readily available for execution and are always executable.

4. The collaborating organisations are required to know the input preconditions, outputs and effects of each other's corresponding interface activities so that compatible workflows could be generated and collaboration could be carried out at runtime among the sending and receiving processes. An interface activity can be a sending activity or a receiving activity. In this paper, an activity name followed by "*_s*" or *"_r*" means it is a sending or receiving activity respectively.

5. To ensure maximum usability of the 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.

4 Architecture

Figure 1 shows the general architecture of the proposed cross organisation compatible workflows generation and execution framework. Although there can be more than two collaborating organisations, for clarity the figure only depicts two. The framework requires OWLS process definitions and high-level goals from collaborating organisations as input.

As shown in Figure 1, the collaborating organisations pass their OWLS process definitions and high-level goals to the Collaboration and Workflow Generation Manager (CWGM). The CWGM loads the processes and passes the 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 the problem is done so that operators and workflows of the 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.

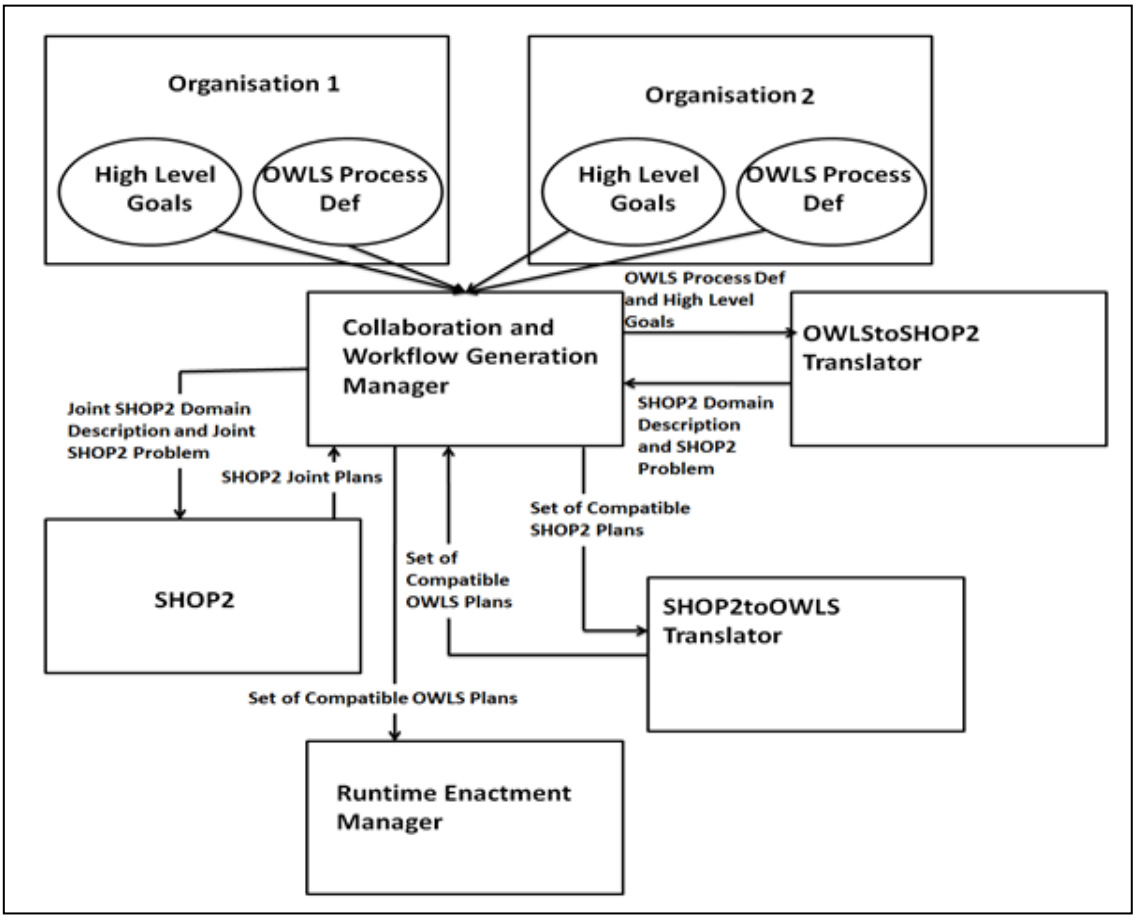

Figure 1 Architecture of the Proposed Framework

The CWGM can be present on a central system or one of the collaborating organisations. It is assumed that all the collaborating organisations agree to provide the path to their process definitions to CWGM. The workflow generation process needs to be repeated for every set of collaborating organisations. This is necessary because the atomic processes of the collaborating organisations and the services modelled by the atomic processes can be outside the boundaries of the collaborating organisations and their availability can change anytime. So, the generated workflows are always based on the available atomic processes that can be actually enacted. As the workflow generation process is purely automatic and based on process definitions, it will not create an explosion of interaction modalities among the collaborating organisations. Once the process descriptions are specified, the workflow

generation process is extremely quick as compared to workflow negotiation process; so this must not be a concern for the collaborating organisations.

In order to avoid planning against a huge number of irrelevant services, 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 relevance of web services for workflow generation is decided on the basis of their outputs and effects and it is achieved by a recursive checking algorithm.

The CWGM collapses SHOP2 domain descriptions of all interacting organisations into a single joint SHOP2 domain. The 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; it achieves their combined goals from their combined initial states, based on their combined domain descriptions. Each joint plan is subdivided to create a set of collaborating plans, one plan for each organisation. These plans are generated so that they are compatible with each other.

The set of compatible plans with the least number of activities is highlighted to the collaborating organisations for execution. The collaborating organisations select the highlighted set of compatible plans or any other set of compatible 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. The OWLS workflows are further passed to the Runtime Enactment Manager, which executes the actual Web Service Definition Language (WSDL) services modelled by the activities (OWLS processes) in the OWLS workflows and makes sure that the transfer of information and data among the collaborating organisations takes place smoothly.

5 Functionality

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 framework presented in the paper is closely related to the system

proposed by Sirin *et al.* (2004). The work presented in this paper extends the application of AI planning to workflow generation as well as workflow collaboration. Below are some of the major extensions and improvements the proposed framework makes to the approach taken by Sirin *et al.* 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 proposed 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.

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 paper does not have this limitation.

3. Similarly, their system executes information-providing services (services with only outputs) at planning time to produce the required output. The developed framework does not execute web services at planning time. It 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. If atomic processes and goals of the collaborating organisations are fed up as single unit to Sirin's system, it will fail to generate any plan. To enable it to generate a plan, we will need to group the atomic processes in the form of a composite process. The framework presented in this paper 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.

The following sub-sections discuss the detailed functionality and present the algorithms involved at each step.

5.1 Translating OWLS Process Definitions to SHOP2 Domain Descriptions

Collaborating organisations can load their OWLS process definitions to the CWGM using an interactive GUI. The collection of OWLS process definitions of an organisation are loaded in the form of an OWL file or a single composite process importing the atomic, simple and composite processes of the organisation.

The OWLSReader module of the CWGM reads the OWLS process definitions included in the OWL file loaded through GUI. The initial states and goal states of the collaborating organisations can be selected from GUI. All processes are loaded from the OWLS process definitions. The loaded processes and their inputs, outputs, preconditions and effects are 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*. The inputs, outputs, preconditions and effects of interface activities are not prefixed. This is because the outputs/effects of interface activities are used by the corresponding interface activities of other collaborating organisations.

The OWLStoSHOP2 Translator module translates OWLS process definitions into SHOP2 domain descriptions. OWLStoSHOP2 Translator also translates initial states and high-level goals selected from GUI into a SHOP2 problem. In order to translate OWLS processes into SHOP2 format, we propose the following algorithm.

The first step in the algorithm is to translate atomic processes 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. We present algorithms to carry out these tasks. The purpose of planning is to create an execution path for this automatically generated top-level composite process.

The Translate-Atomic-Process(*Q*) algorithm translates OWLS atomic processes into SHOP2 operators. It extends the translation algorithm put forward in (Sirin *et al.*, 2004), to translate atomic processes with both outputs and post-conditions. 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

The Translate-Atomic-Process(*Q*) algorithm 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 of the atomic process into negative post-conditions of the SHOP2 operator.

Unlike the translation algorithm described in Sirin *et al.* (2004) which translates only the preconditions of atomic processes into the preconditions of SHOP2 operators, Translate-Atomic-Process(*Q*) translates both the preconditions and inputs of atomic processes into the preconditions of SHOP2 operators. This enables the developed framework to use web services that have both inputs and preconditions in workflow generation. Similarly, unlike the translation algorithm described in Sirin *et al.* (2004) which translates only the effects of atomic processes into the post-conditions of SHOP2 operators, Translate-Atomic-Process(*Q*) translates both the effects and outputs of atomic processes into the post-conditions of SHOP2 operators. This enables the presented framework to use web services that have both outputs and effects in workflow generation.

The Translate-Composite-Process(*Q*) algorithm 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. It works as follows.

Translate-Composite-Process(*Q*)

Let *Q* be the definition of a composite process *C* and *L* be a set of SHOP2 operators.

($b_1, \ldots, b_n$) is the list of 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

The Translate-Composite-Process(Q) algorithm 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, it calls Translate-Composite-Process(q_i) or Translate-Simple-Process(q_i) to translate it into a set of SHOP2 operators.

The translation algorithm described in Sirin *et al.* (2004) translates composite processes directly into SHOP2 methods as it looks at web service composition as finding an execution path for already defined composite processes. The proposed framework looks at web service composition as automatically combining atomic processes to form a composite process, for which an execution path can be found. This enables the proposed framework to automatically generate compatible workflows from atomic processes of collaborating organisations and enable the organisations to avoid the time consuming task of creating composite processes on their own.

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

Translate-Simple-Process(*Q*)

Let *Q* be the definition of a simple process *S* and *L* be a set of SHOP2 operators

($b_1, \ldots, b_n$) is the list of 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

The Translate-Simple-Process(*Q*) algorithm 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 implemented 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 the discussed approaches (Sirin *et al.*, 2004; Wu *et al.*, 2003), the implemented 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. 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 collection of OWLS processes passed to CWGM is translated into a SHOP2 domain. The Translate-OWLStoSHOP2(*P*,*G*) algorithm that translates 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-OWLStoSHOP2 (*P, G*)

Let *P* be a collection of OWLS processes, *K* be the set of definitions of OWLS processes in *P*, *G* is the conjunct of all goal states as specified by the organisation, *M* be a SHOP2 method with the name *BP* (Business Process), *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

Let $O=\{O_1,O_2 \ldots O_m\}$ be the translated set of SHOP2 operators and Pre_i = (conjunct of preconditions of O_i)

$M = (BP() \; G \; Nil \; Pre_1 \; O_1 \; BP \; Pre_2 \; O_2 \; BP \; \ldots \; Pre_m \; O_m \; BP)$

Add *L* to *D*

Add *M* to *D*

Return *D*

End Translate-OWLStoSHOP2

The Translate-OWLStoSHOP2(*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. The recursive SHOP2 method named *BP* groups the operators in an *if-then-else* format. The method BP represents the top-level business process of the corresponding organisation. An operator is executed when its preconditions hold. If the planner achieves all of the goal states in G, *Nil* is called to quit the method *BP*. 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 it a recursive expression, which will be called by the planner recursively, until the goals states are achieved or the planners fails to find any valid plans. *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.

Unlike the proposed approach described above, the approach by Sirin *et al*. (2004) does not combine operators to form a method. This means that their system can generate workflows only if the user manually defines the composite processes and passes them to the system. The composite processes and atomic processes are passed to the system together, for translation into SHOP2 domain.

5.2 Combining the Translated SHOP2 Domains into a Joint Domain

To carry out cross organisational workflow collaboration at workflow generation time, we introduce the following algorithm that collapses the domain descriptions for all interacting organisations in a single joint domain. In this way, all the interacting organisations are considered sub organisations of a single parent organisation. The SHOP2 *BP* methods representing the top-level business processes of each collaborating organisation in an *if-then-else* format are joined together to create a single joint SHOP2 method named *JBP*. The generated SHOP2 method represents the high-level business process of the single parent organisational structure having cross organisational boundaries.

The Create-Joint-SHOP2-Domain (*D*) algorithm creates a joint SHOP2 domain by taking set D of SHOP2 domains as input.

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 } be 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 of goals of Org_i

add G_i into *G*

End for

Add *O* to *JD*

Add *M* to *JD*

Let *O* = {O_1,O_2,…O_m} be the set of operators in *JD*, Pre_o = {Pre_{o1},Pre_{o2},…Pre_{om}} be the set of conjuncts of preconditions of {O_1,O_2,…O_m} respectively, *M* = {M_1,M_2,…M_n}be the set of methods in *JD* and {Pre_{m1},Pre_{m2},…Pre_{mn}} be the set of conjuncts of preconditions of {M_1,M_2,…M_n} respectively

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*)

Add *JBP* into *JD*

return *JD*

End Create- Joint-SHOP2-Domain

The Create-Joint-SHOP2-Domain(*D*) algorithm combines the operators and methods of the collaborating domains and merges them into the joint domain. It then creates a recursive SHOP2 method *JBP*, which groups the operators and methods of all collaborating organisations in an *if-then-else* format. The planner executes an operator or decomposes a method when its preconditions hold. If all goal states in *G* are achieved, *Nil* is called to quit the method *JBP*. In the expression *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*), calling *JBP* after every Pre_{oi} O_i and every

Pre_{mi} M_i makes it a recursive expression and *JBP* will be called recursively by the planner until valid plans are found or the SHOP2 returns a failure.

As the system proposed by Sirin *et al*. (2004) targets the creation of workflows for a single organisation only, it has a single SHOP2 domain to begin with. Therefore, they do not present any algorithm for collapsing the domains of multiple collaborating organisations into a single domain.

5.3 Planning for All Possible Sets of Compatible Plans

To plan for all possible sets of compatible plans, the SHOP2 needs to be extended in order to enable it to handle data inputs. During planning, the preconditions especially the ones representing data inputs will remain true in the entire lifecycle of the planning process until explicitly made false by an operator. If the atomic processes do not explicitly make their preconditions/inputs false, SHOP2 will keep repeatedly adding the first task list whose preconditions are true in the workflow. This will create an infinite loop. 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. We extend the SHOP2 planning algorithm so that the same tasks are not repeatedly added to the workflow (plan) or selected for decomposition (see Figure 2).

The extended SHOP2 planner creates a set $P = (P_1\ P_2 \ldots P_n)$ of multiple valid plans where every plan P_i in P is a sequence of instantiated operators $(O_1, O_2, \ldots, O_m)$ that will achieve the desired goals from the given initial states, in the joint domain. 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 the least number of operators is highlighted to the users for execution. Considering each operator takes the same time, this is the least cost heuristic. The users can select the highlighted set or any other set of compatible plans for execution.

If '*s*' is the current state of the world, '*T*' is the task list and '*D*' is the domain, the algorithm for the extended SHOP2 planner is as follows:

```
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 preconditions
   are 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 preconditions are 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 2 Extended SHOP2 Algorithm for Workflow Generation

The compatibility of the plans generated by the division of a joint plan is intuitive. In the joint plan, the compatible plans for each organisation are arranged together in a particular order that 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 (Yang and Papazoglou, 2000).

5.4 Runtime Execution and Collaboration

The developed framework provides runtime support for the generated sets of compatible workflows. The developed runtime execution mechanism is the only execution mechanism so far that enables the execution of multiple collaborating compatible OWLS based workflows.

The existing execution mechanisms from literature enact automatically generated workflows for single organisations only, however they can handle adhoc processes that are outside the boundaries of the organisation in the workflows (Chen *et al.*, 2011).

The selected set of compatible plans is passed to 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 OWLS API. The enactment of an atomic process is a call to the corresponding web accessible program with its inputs instantiated. The generated outputs are kept in form of a name-value pair, so that they can be passed as inputs to the corresponding processes downstream.

In real life, workflows generally run locally in the respective organisations. Although, the developed runtime execution system provides a centralised system for the execution of the workflows of collaborating organisations, the actual web services are enacted locally at the collaborating organisations. It has a similar effect as that of executing workflows locally. To execute the workflows locally at the respective organisations, the generated plans may be distributed to the collaborating organisations and the execution system may be hosted at each collaborating organisation. It can be achieved by extending the system to the client-server architecture. It would be helpful in cases where the organisations are not comfortable with sharing their execution data with an external organisation hosting the execution system.

Since the implemented runtime execution mechanism 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 activities (Chen and Chung, 2008). 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 if an organisation has to send huge documents 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. It also decouples the collaborating organisations from each other

completely at runtime, which is a desired quality (Van-der-Aalst, 1999; Van-der-Aalst and Weske, 2001). The execution mechanism will wait on a receiving activity until the respective sending activity has been executed.

Unlike the presented enactment mechanism, the mechanism proposed by Sirin *et al*. (2004) targets the enactment of a single workflow only, and it is not able to perform runtime collaboration among the workflows of multiple collaborating organisations.

6 Implementation

A proof-of-concept prototype has been implemented for the proposed framework. The GUI is developed using Swing and AWT classes of Java. Figure 3 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. Initial states and goal states can be selected at GUI. At runtime, the workflow to execute can also be selected from GUI.

The OWLSReader, CWGM, OWLStoSHOP2 Translator, SHOP2toOWLS Translator and Runtime Execution Manager are also developed using Java. The OWLSReader and Runtime Execution Manager are based on OWLS API, which is a Java based API for programmatic access to read, execute and write OWLS service descriptions. The planning is done using a modified version of JSHOP2 planner. JSHOP2 is Java implementation of the SHOP2 planner.

OWLS process definitions can be created manually or automatically using OWLS editor of Protégé. Protégé can load WSDL files and generate a skeleton OWLS process. It further provides graphical control constructs such as sequence, split, join, and choice to create composite processes from atomic processes. WSDL2OWLS tool can also be used for automatic generation of OWLS process definitions from WSDL descriptions. Appendix C shows an atomic OWLS process *IssueInspCert* from the workflow collaboration example in Section 8. It has been created automatically from its WSDL descriptions using WSDL2OWLS tool. WSDL descriptions of the web services are automatically generated from the Java code of the web services with the help of Apache Axis2. We use Jsch API to upload and download files over Secure File Transfer Protocol (SFTP). Jsch is a Java implementation of SSH2.

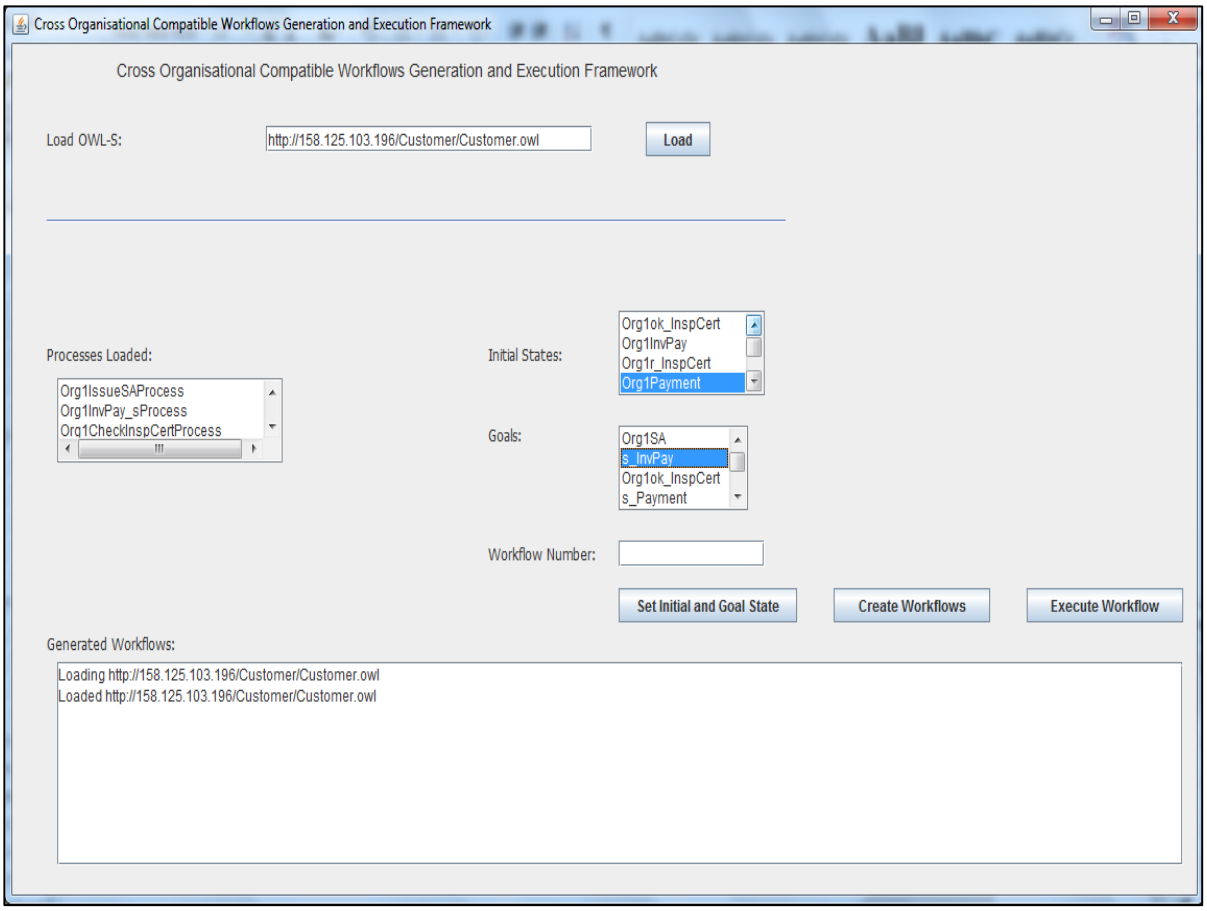

Figure 3 GUI of the Prototype

7 Workflow Collaboration Examples

7.1 Vendor/Customer Example

We will consider a Vendor/Customer example scenario. This is a modification of the example presented by Chen (2008). 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 it issues a commercial invoice represented as *Invoice*, carries out factory inspection as an in-house procedure, 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. It 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.

The OWLS process descriptions simulating the actual activities of *Vendor* and *Customer* are passed to the implemented framework. The *Vendor and Customer* can have any number of OWLS processes and the developed framework will filter out any that are not relevant to a given application scenario. Each activity is represented as an OWLS process, which is grounded in an actual WSDL service. The OWLS process descriptions for *Vendor* and *Customer* are given in Table 1 and 2 respectively in Appendix A.

Based on the passed OWLS processes, the system generates 20 sets of compatible workflows in 6816 milliseconds. 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 concurrently. Figure 4 shows two of the generated workflows for *Vendor* and *Customer*. 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.

Figure 4 shows that the data dependencies are the same in both sets of the workflows but the control dependencies are different. In the *Vendor*'*s* workflow in Set 1, *ShippingArrangement* has a control dependency on *SA_r*, and *Inv_s* has control dependency on *InsuCert_s*. In the *Vendor*'*s* workflow in Set 2, *Inv_s* has a control dependency on *SA_r* and *ShippingArrangement* has control dependency on *Inv_s*.

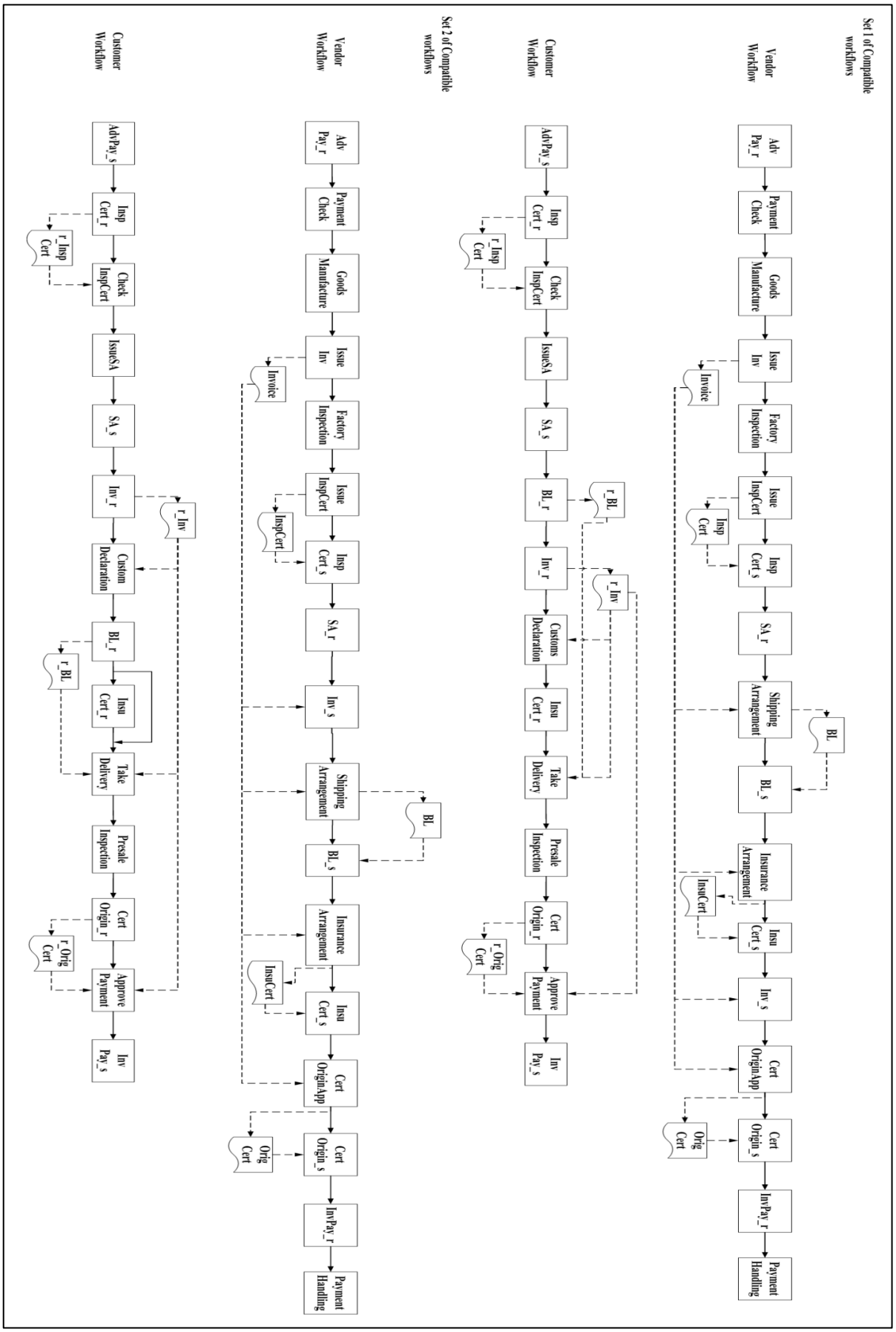

Figure 4 Sets of Compatible Workflows for *Vendor* and *Customer*

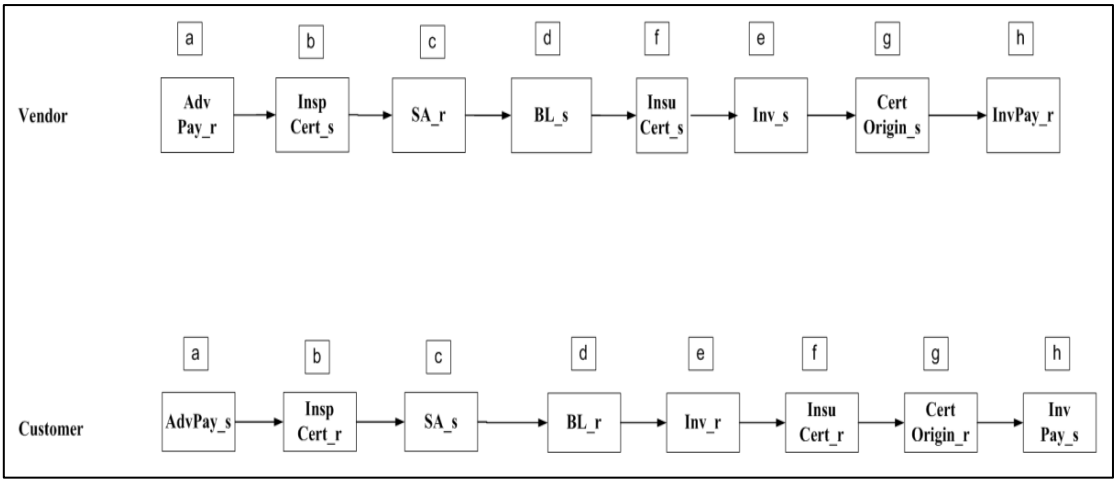

Figure 5 Interface Processes for Set 1 of *Vendor*/*Customer* Workflows in Figure 4

Similarly, in the *Customer*'*s* workflow in Set 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 Set 2, *Inv_r* has a control dependency on *SA_s* and *BL_r* has control dependency on *CustomsDeclaration.*

Both sets of workflows are accurate and compatible. The workflows, when executed, are able to achieve the desired goals of the collaborating organisations. Moreover, the workflows can be executed to the end in coordination with the collaborating workflows. The compatibility of the workflows can be verified by considering their respective interface processes. Figure 5 shows the interface processes for the Set 1 of compatible workflows in Figure 4. 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.

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 activities. For example, in Figure 4, *InspCert_s* is a

sending activity 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.

7.2 Retailer/Wholesaler/Manufacturer/Supplier Example

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 from the real world and therefore we have used it as an example to test the developed 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 Table 3, 4, 5 and 6 respectively in Appendix B. The OWLS process definitions as given in Table 3, 4, 5 and 6 were passed to the system and it generated 10 sets of compatible workflows for the four organisations in 9832 milliseconds. Figure 6 shows one of the generated sets. The workflows generated are accurate and compatible.

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 indicate 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*.

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 actual 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.

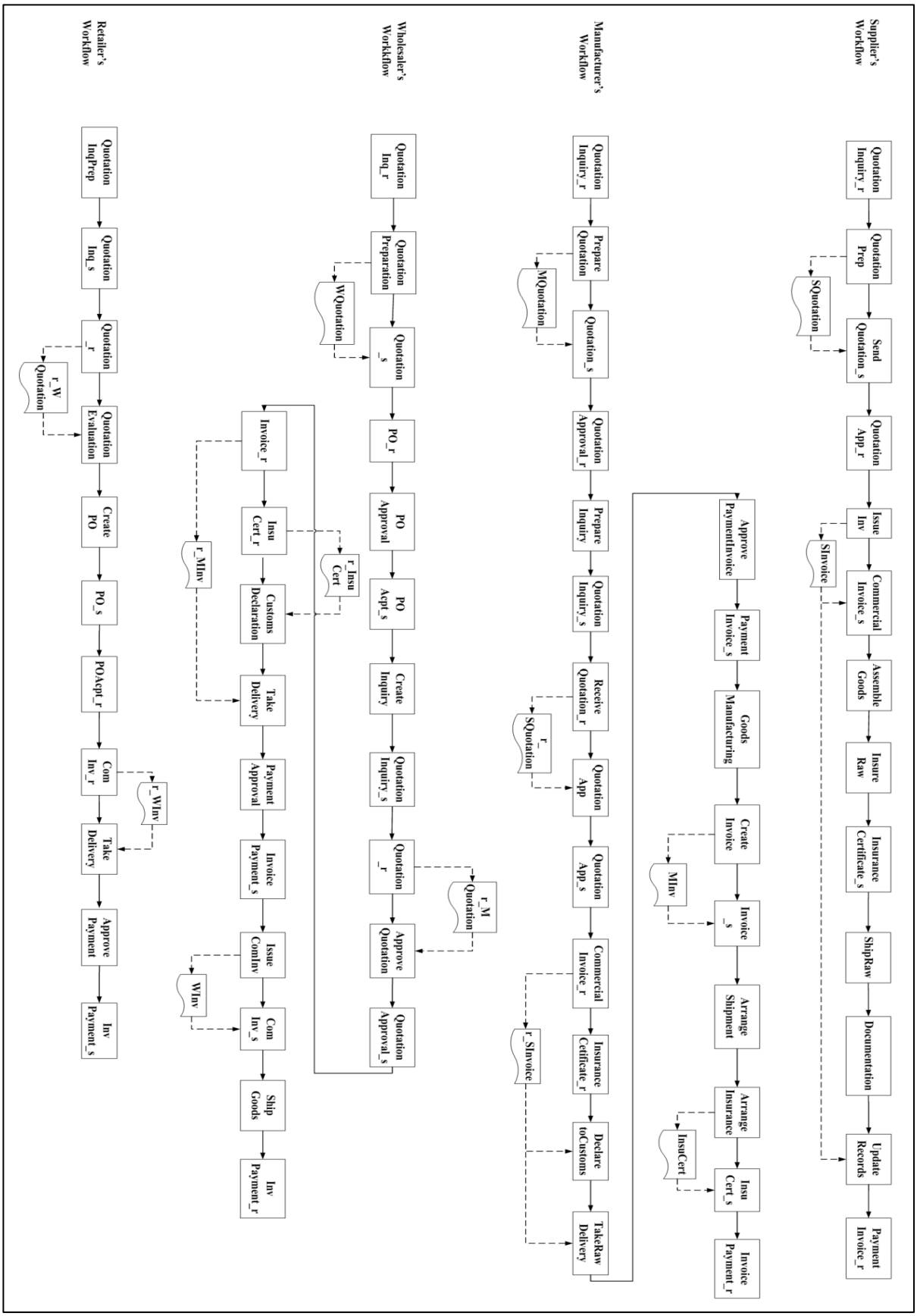

Figure 6 A Set of Compatible Workflows for *Retailer*, *Wholesaler*, *Manufacturer* and *Supplier*

The execution of the compatible workflows to the end achieves the desired goals. During the execution phase, the actual WSDL web services modelled by the OWLS atomic processes in the workflows of *Retailer*, *Manufacturer*, *Wholesaler* and *Supplier* are enacted with the help of Simple Object Access Protocol (SOAP).

7.3 Further Application Scenarios

The case studies above illustrated how collaboration between different partners in a supply chain in industry could be supported. The proposed framework and technologies can be used to support a wide range of cross organisation collaboration in different domains. For example, in the higher education sector it is common that a higher education institution would apply to different government funding bodies, charities or companies for research funding. Each of these organisations has their own workflows for application submission, review, and award notification and monitoring. These organisations typically work with many institutions and each of these institutions has its own workflows for grant preparation, grant expenditure, project monitoring and project reporting. It is clear that there is a huge potential and need for workflow support for cross organisation collaboration in this sector too. The proposed framework as illustrated can be modelled to support research grant management across different organisations and institutions.

Another application scenario is support for libraries. Each library normally has a workflow for lending out books. A library may be required to automatically obtain books from other libraries or buy it from bookstores like Amazon, if a required book is not available in the database of the library. Different libraries may have their own respective workflows for lending out books, and book stores like Amazon also have their workflows including activities like book searching, book selection based on certain criteria, card validation, payment and shipping etc. This scenario requires multi-organisation collaboration.

The proposed framework can also be used in the mortgage trading domain. A mortgage trading consultancy may be required to automatically obtain mortgage information from various banks, select the best available option based on the limitations and goals of the client and connect to the selected bank to begin the mortgage application process for the client.

For the above and other multi-organisation workflow collaboration scenarios, the proposed framework can be used to generate compatible workflows for the collaborating organisations, based on their web services descriptions and high level goals.

8 Discussion and Future Work

This paper presented a framework for the generation and execution of compatible workflows for multiple collaborating organisations. The presented framework is different from existing systems because the existing systems reconcile pre-modelled workflows. This is a time consuming technique, more so, if the organisation is collaborating with many partners. Automatic workflow generation is the solution to tackle this problem. Existing systems that can automatically generate workflows can do so for single organisations only and cannot handle the generation of compatible workflows for multiple organisations. This leaves the organisation to reconcile the workflows with the collaborating partners on their own if there is any incompatibility, which again requires time and resources. The presented framework solves this problem by integrating workflow generation and workflow collaboration. It generates compatible workflows for multiple collaborating organisations, so that the time and resources invested in modelling and reconciling collaborating workflows can be saved. It also has the capability to handle the execution of the generated collaborating workflows.

Since the workflow generation is based on web services composition, the implemented framework supports reusability and interoperability. Web services from highly diverse sources can be composed in a workflow, and invoked to achieve a desired goal. So the already developed functionalities do not need to be redeveloped and can be reused to save time and resources. The implemented framework encourages cohesiveness and modularity.

The scenarios given in Section 7 show that the developed framework can generate compatible workflows for two or more collaborating organisations and it can support the collaborative enactment of workflows of two or more interacting organisations.

As obvious from the time taken for the generation of workflows in Section 8, the developed system makes the generation of compatible workflows for collaborating organisations extremely efficient. While the time required by the system presented in this paper is in milliseconds, the usual time required for manual collaborations is in days. Practically, the maximum number of organisations or the number of processes that the developed system can handle is dependent on the available memory of the hardware system running it.

SHOP2 does not follow a specific model of time (Parkinson *et al.*, 2011). The time taken by the system to generate workflows is dependent on several factors, including: the number of collaborating organisations, the number of activities in the workflow of each collaborating

organisation and the quality of the domain knowledge used for planning (Saleem, 2012). The SHOP2 domain for workflow generation problem is in the form of web services descriptions and OWLS process definitions translated into SHOP2 format. The order in which the methods are specified in the domain can influence the efficiency of SHOP2 (Sohrabi and McIlraith, 2009; Shivashankar *et al.*, 2011). The order in which the if-else conditions are specified in the *JBP* method can also influence the efficiency of the system. So a linear increase in the planning time as a function of number of participating organisations cannot be concluded. Nonetheless, planning time for the scenario involving four organisations is also sufficiently fast for practical application.

The developed framework uses AI planning for workflow generation. AI planners are not usually designed for the web scale planning problems. A mismatch between SHOP2 and OWLS that exists is that the logic used for describing SHOP2 domain is differently expressive than OWL used for describing web services (Sirin and Parsia, 2004) i.e. while OWL assumes an open world, the SHOP2 has a closed world assumption. Similarly, SHOP2 assumes that the modelled domain must be correct which is not easy to ensure in the web domain (Sirin and Parsia, 2004). The data in the semantic web domain can be too huge for the relatively limited inferencing capabilities of AI planners. The integration of an OWL reasoner with SHOP2 will minimise these issues (Sirin and Parsia, 2004). The replacement of the theorem-prover of SHOP2 with a sound and complete OWL reasoner to exploit its inferencing capabilities, suitability to the semantic web and its usability for workflow generation will be investigated and implemented in future. The effect of the integration of OWL reasoner with SHOP2 on the efficiency of the developed framework also needs investigation.

The framework currently focuses on the compositional capabilities of OWLS processes and does not focus on the automatic discovery of OWLS processes from the web. The reasoning capability of OWL reasoners can be used for automatic web services discovery, which will be targeted in future. Similarly, the paper does not focus on the security aspects of web services invocation.

SHOP2 does not support concurrency (Sirin *et al.*, 2004) and hence it cannot create parallel workflows. SHOP2 can be extended to support concurrency, which will in turn enable the support for parallel workflows. ConGolog supports concurrency (Giacomo *et al.*, 2000) and therefore can be used to create parallel workflows. The extension of SHOP2 for concurrency

and the use of ConGolog interpreter for parallel cross organisational compatible workflows generation need further investigation.

Many small organisations carry out electronic commerce using online business platforms like eBay and Amazon. In such situations, the automatic workflow generation, collaboration and enactment is dependent on the permissions and functionalities provided by the host e-commerce platforms and the standards that they follow to provide point-to-point interaction. To investigate the effort required to migrate or adapt such platforms to provide flexible cross-organisation collaboration would be an interesting and challenging area for further research.

9 Acknowledgements

This work is funded by Engineering and Physical Sciences Research Council (EPSRC) through Innovative Manufacturing and Construction Research Centre (IMCRC).

References

Chen, F., Ren, C., Dong, J., Wang, Q., Li, J. and Shao, B., 2011. Modeling cross-organizational services composition with Pi-calculus, in Proceedings of the IEEE International Conference on Service Operations, Logistics, and Informatics, pp. 51–56.

Chen, L. and Yang, X., 2005. Applying AI Planning to Semantic Web Services for Workflow Generation, in Proceedings of First International Conference on Semantics, Knowledge and Grid, IEEE Computer Society, Washington DC, USA, pp. 65.

Chen, X., 2008. IT supported business process negotiation, reconciliation and execution for cross-organisational e-business collaboration, Thesis at the Faculty of Computer Science, Loughborough University, UK; Available from: https://dspace.lboro.ac.uk/dspace-jspui/handle/2134/4873.

Chen, X. and Chung, P., 2006. Cross-Organisational Workflow Enactment Via Progressive Linking by Run-Time Agents, in Proceedings of International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, pp. 54–59.

Chen, X. and Chung, P.W.H., 2007. A simulation-based difference detection technique for bottom-up process reconciliation, in Proceedings of the 9th International Conference on Enterprise Information Systems, pp. 72–77.

Chen, X. and Chung, P.W.H., 2008. Facilitating B2B E-business by IT-supported business process negotiation services, in Proceedings of IEEE International Conference on Service Operations and Logistics, and Informatics, pp. 2800–2805.

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

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

Correˆa da Silva, F.S., Venero, M.L.F., David, D.M., Saleem, M. and Chung, P.W.H., 2013. Interaction Protocols for Cross-organisational Workflows, in Knowledge Based Systems, **37**, pp. 121–136.

Crubezy, M. and Musen, M., 2003. Ontologies in Support of Problem Solving, in Staab, S. and Studer, R., Editors, Handbook on Ontologies, pp. 321–342.

Dong, X. and Wild, D., 2008. An Automatic Drug Discovery Workflow Generation Tool Using Semantic Web Technologies, in Proceedings of the Fourth IEEE International Conference on eScience, IEEE Computer Society, pp. 652–657.

Elenius, D., 2004. Modelling Services with Protégé, in Seventh International Protégé Conference.

Giacomo, G.D., Lesperance, Y. and Levesque, H.J., 2000. Congolog, a Concurrent Programming Language Based on the Situation Calculus, in Artificial Intelligence, **121**(1-2), pp. 109–169.

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

Meng, L.X., He, F. and Sun, L.L., 2012. Research on Semantic Business Process Model in Logistics Distribution Field, in Applied Mechanics and Materials, **198-199**, pp. 899–904.

Okutan, C. and Cicekli, N.K., 2010. A monolithic approach to automatic composition of semantic web services with the Event Calculus, in Knowledge Based Systems, **23**(5), pp. 440–454.

OMG, 2011. Business Process Model and Notation (BPMN).

Ouyang, C., Marlon D. and van der Aalst, W.M., 2008. Pattern-based translation of BPMN process models to BPEL web services, in International Journal of Web Services Research (IJWSR), **5**(1), pp. 42–62.

OWL Services Coalition, 2003. OWLS: Semantic markup for web services; Available from: http://www.ai.sri.com/daml/services/OWLS.

Parkinson, S., Longstaff, A.P., Crampton, A. and Gregory, P., 2011. in Proceedings of the 22nd International Conference on Automated Planning and Scheduling, pp. 216–224.

Peer, J. and Vokovic, M., 2005. A proposal for a semantic web service description format, in Proceedings of the European Conference on Web Services, Springer-Verlag, pp. 285–299.

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

Saleem, M., 2012. Cross Organisational Compatible Workflows Generation and Execution, Thesis at the Faculty of Computer Science, Loughborough University, UK.

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

Shivashankar, V., Kuter, U. and Nau, D. S., 2011. Hierarchical Goal Network Planning: Initial Results, Technical Report CS-TR-4983 and UMIACS-TR-2011-0.

Sirin, E., Hendler, J. and Parsia, B., 2003. Semi-automatic composition of Web services using semantic descriptions, in Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS, pp. 17–24.

Sirin, E. and Parsia, B., 2004. Planning for semantic web services, in Semantic Web Services Workshop at 3rd International Semantic Web Conference.

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

Sohrabi, S. and Mcilraith, S. A., 2009. Optimizing web service composition while enforcing regulations, in The Semantic Web-ISWC 2009, Springer Berlin Heidelberg, pp. 601–617.

Srivastava, B. and Koehler, J., 2003. Web Service Composition - Current Solutions and Open Problems, in ICAPS 2003 Workshop on Planning for Web Services, pp. 28–35.

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

Van-der-Aalst, W.M.P. and Weske, M., 2001. The P2P Approach to Interorganizational Workflows, in Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag, pp 140–156.

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

Wu, D., Parsia, B., Sirin, E., Hendler, J. and Nau, D., 2003. Automating DAML-S web services composition using SHOP2, in Proceedings of 2nd International Semantic Web Conference (ISWC2003), pp. 195–210.

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

Appendix A. Process Details of *Vendor* and *Customer*

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 1 *Vendor'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*.

Table 2 *Customer's* OWLS Processes

Appendix B. Process Details of *Retailer, Wholesaler, Manufacturer* and *Supplier*

The OWLS processes for *Retailer*, *Wholesaler, Manufacturer* and *Supplier* are given in Table 3, 4, 5, and 6 respectively. Only the processes relevant to this collaboration scenario are given.

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 3 *Retailer'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 4 *Wholesaler'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 5 *Manufacturer'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*.

Table 6 *Supplier's* OWLS Processes

Appendix C. OWLS Definition for IssueInspCert

```
<?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/OWLS/1.1/Process.owl#"
   xmlns:grounding="http://www.daml.org/services/OWLS/1.1/Grounding.owl#"
   xmlns:service="http://www.daml.org/services/OWLS/1.1/Service.owl#"
   xmlns:profile="http://www.daml.org/services/OWLS/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>
```