Service composition of open-box web services

Chen, X.

Award date:
2008

Eindhoven, April 2008

Service Composition of Open-Box Web Services

by
Chen, XingTong

BSc: Material Management 1992
Student identity number 0602525

in partial fulfilment of the requirements for the degree of

Master of Science
in Operations Management and Logistics

Supervisors:
dr.ir. H. Eshuis, TU/e, IS
dr. A.J.M.M. Weijters, TU/e, IS

TU/e Department Technology Management.

ARW 2008 OM & L

Subject headings: Process and service composition, Web services, inter-organizational business process management, formal models in business process management, business process outsourcing.

ABSTRACT

Nowadays, organizations organize their production activities around supply chains to attain competitive competencies. In fast changing markets, collaboration partnerships are established on-the-fly in an adaptive, fine-grained way. In the ICT domain, the establishments of this kind of partnerships mean that services offered by providers are connected to services of consumers; in this way, services are composed. Recent research studies of web service composition have begun to examine approaches to compose open-box web services. This paper discusses a Business Process Web Service (BP-WS, in short) based approach to compose open-box web services. A BP-WS is a web service which has an (internal) business process specification that can be accessed externally. We suggest two procedures: a semi-automatic procedure for composing a set of open-box services, with data flow dependencies, into a structured composite service; and an automatic procedure for synchronizing a composite service with local services using message exchanges. In both these procedures, the open-box services are represented as structured business processes. We also propose dedicated BP-WS architectures for the global process side respectively the local process side to support our synchronization procedure in the bilateral scenario. Finally, we use two case studies: one from the healthcare domain, and one from the car insurance domain to validate our approach.

MANAGEMENT SUMMARY

To attain competitive competencies, organizations now organize their production activities around supply chains. In fast changing markets, collaboration partnerships are established on-the-fly in an adaptive, fine-grained way. In the ICT domain, the establishment of this kind of partnerships means that services offered by service providers are connected and integrated to service consumers, thus services are composed. For the reason of wide industry supports and their model of loose coupling, web services have been seen as a promising solution towards integrating services offered by diverse, heterogeneous and autonomous organizations.

Recent research studies of web service composition have begun to examine approaches to compose open-box web services. The aim of this paper is to solve the problem how to compose services in the open-box model. In this paper, a service is referred to as a set of functionalities to fulfill some business goals, and is represented as a structured business process. To enact services, three basic roles are needed: (i) service provider. (ii) service consumer. (iii) service directory. The open-box model is a service model, in which "there can be arbitrary control flow relations between consumers and providers. The execution progresses of both parties depend on one another." (Grefen et al., 2006a). We propose a Business Process Web Service (BP-WS, in short) based approach to compose open-box services. A BP-WS is a web service that "has an internal business process specification that can be accessed externally through a number of dedicated web service interfaces (ports)." (Grefen et al., 2006a).

In this paper, the composition procedure has as inputs the services presented as structured business processes; and these services have already been selected for composition in accordance with some sophisticated rules. A structured business process is a business process where each "or–split" has a corresponding "or–join" and each "and–split" has a corresponding "and–join" (Kiepuszewski et al., 2000.) We show that services expressed as structured business processes can be composed together, generating a new structured composite service.

We propose a semi-automatic procedure for service composition, applicable in both the bilateral and the multilateral scenario. In the bilateral situation, one service provider provides services to one service consumer; in the multilateral situation, several service providers provide services to one service consumer (Norta, 2007). We also propose an automatic synchronization procedure between the global process of the composite service and the local processes in the bilateral scenario. The local process refers to either the service provider side or the service consumer side. Both the procedures exclusively focus on control flow aspects. We also describe what architecture can support our synchronization procedure in the bilateral scenario.

In Chapter 1, we first survey the problem area — service composition — on which this paper focuses. In this paper, we follow the business process outsourcing paradigm in the bilateral context. In the context of business process outsourcing, service composition implies that the business process from the service provider is embedded into that from the service consumer. In the multilateral situation, we assume the inter-operability among services is loosely coupled. Loosely coupled inter-operability implies that the participants of the collaborations are organized in a peer-to-peer topology. Next, we substantiate the relevance of service composition.

In Chapter 2, we lay the foundation of this paper by explaining and discussing the main concepts and models used in this paper. We start with Business Process Web Service (abbreviated to BP-WS) and the open-box model which are the underlying foundations for our approach. Next, we introduce workflow management technology. The combination of service-oriented computing (SOC, in short) and workflow management (WFM) technology has been proposed as a promising approach to compose open-box services. Our focus is on structured workflow models. We also briefly review work (Eshuis et al., 2006) in this chapter, as our work is an extension of work (Eshuis et al., 2006).

In Chapter 3, we formally introduce a semi-automatic procedure to compose the open-box services. We reuse the approach proposed by Eshuis et al. (2006). Our semi-automatic procedure is also based on dependency graphs; and the algorithm proposed by Eshuis et al. (2006) is reused to construct the structured composite services. In the composition procedure, the most abstract composite blocks which are involved in the inter-organizational dependencies are treated as groups; because the inter-organizational dependencies put direct/indirect influences on all the sequential blocks within the most abstract composite block.

The semi-automatic procedure consists of four steps. Firstly, the most abstract composite blocks which are involved in the inter-organizational dependencies, both from the service consumer and the service providers, are replaced by dummy activities —aggregates; and a dependency graph which contains aggregates is constructed. Secondly, we use the algorithm introduced by Eshuis et al. (2006) to construct a structured global process which contains aggregates. The third step is un-aggregate, i.e. the original composite blocks are restored in the global process. The last step is manually typing the branching types with AND or XOR control flow constructs in the global process.

To successfully use the algorithm proposed by Eshuis et al. (2006), dependency graphs must satisfy two constraints. In the above composition procedure, the dependency graph which contains aggregates must also satisfy two constraints:

Constraint 1: The dependency graph is acyclic.

Constraint 2: If there is an edge from activity1 (aggregate 1) to activity2 (aggregate 2), then there is no path with length greater than 1 from activity1 (aggregate 1) to activity2 (aggregate 2).

The relaxation of the first constraint is planned as future work; the violation of the second constraint can be easily repaired.

The above presented composition procedure is applicable in both the bilateral scenario and the multi-lateral scenario.

In Chapter 4, we propose an automatic synchronization procedure between the global process and the local processes in the bilateral scenario. The key part of this automatic procedure is constructing the list of control activities (LCA). A control activity is an activity at the local process side the start of which the global process needs to control. The LCAs are constructed by a component “Construct LCA” at the global side. The component “Monitor & Control” controls when the activities on the LCA can start by means of message exchanges from global process to the local processes. Note, there are two LCAs in the bilateral scenario at the global side. LCA(P) is for the provider side; LCA(C) is for the consumer side. When constructing the LCA, we still have to treat some composite blocks as groups, just as in the composition procedure. The execution of the local side is automatic; the execution of the activities not on the LCAs is controlled by the local side. The global process needs to know the start and the completion of every activity at the local sides.

Our synchronization approach relies on the dedicated architectures. We developed new dedicated architectures for the global process respectively for the local processes to fully support our approach. This part of work is inspired by Grefen et al. (2006a and 2007). For the global side, component "Monitor & Control" and component "Construct LCA" were introduced. The LCA is stored in the document LCA(P) or LCA(C). The interface "CTRL(G)" is used to facilitate the synchronization from the local side to the global side. A user uses interface "MON UI" to monitor and control global process during the enactment time. For the local side, the architecture is basically the same with that proposed by Grefen et al. (2006a).

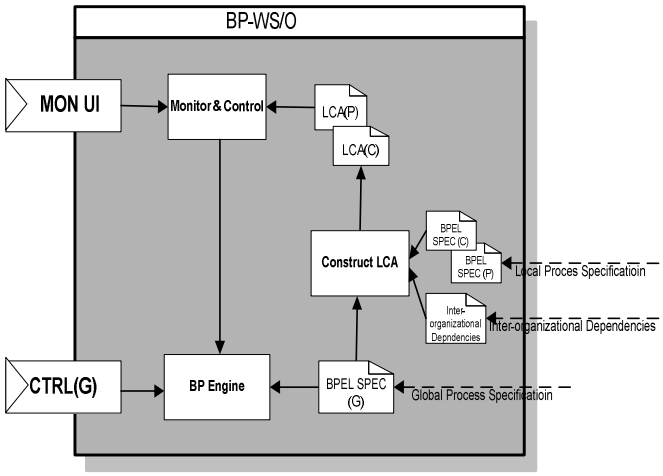

Figure 1: BP-WS/O architecture dedicated to the global process

In Chapter 5, the complete approach for composing services in the open-box model is put to the test by means of case studies. In the bilateral scenario, a case from the healthcare domain is used; in the multilateral scenario, a case from the car insurance domain is used.

In Chapter 6, a conclusion is drawn. We first summarize our work in this paper. Next, we present our reflections on the three research questions. We did not fully meet our aims. In terms of the composition procedure, our work is only applicable to structured process models; and the process models should contain no loops. With respect to the synchronization procedure, our work is only applicable if the processes do not contain any loops; and only applicable in the bilateral scenario. At last, we suggest some future works; for instance, how to extend some of our results for the multi-lateral scenario.

CONTENT

LIST OF FIGURE

Chapter 1

INTRODUCTION

ICT technologies, especially the Internet, have changed the way how business organizations operate in the modern world. IT and software applications play an important role in attaining (or retaining) the organizations' competencies. One challenge facing the ICT community is how to support the formation and the enactment of intensive inter-organizational collaborations among autonomous, heterogeneous business organizations in a time and cost efficient way.

Services Oriented Computing (SOC) is proposed as the most promising technology to support this kind of inter-organizational collaborations (Grefen, 2006). Web services provide a uniform framework for distributed application developments and resources sharing and are now seen as the primary solution towards composing inter-organizational business functionalities across the web (Manes, 2003). The concept of Business Process Web Service (BP-WS, in short) is an extension to the basic web service paradigm. A BP-WS has an (internal) business process specification that can be accessed externally by means of a number of dedicated web service ports (Grefen et al., 2006a).

The current state of the art of research works in service composition mainly focus on composing business functions and not so much on integrating business processes — in the open-box model as explained later on. Business functions are treated as black-box services, so atomic functionalities towards their invokers; business processes, however, have multiple functions (activities) (Grefen, 2006). To facilitate close collaboration, organizations are not satisfied with composition requirements which only address business functions. They want or need to monitor and control the progress of the instances of the services (Grefen, 2006; Grefen et al., 2006a). The main issues that this paper addresses are: "How can services expressed as explicitly visible business processes be composed together? ". "How can the composite service be synchronized with the local services?" and "What run-time architectures can support our approach?"

In this introduction, we first survey the problem area — service composition — on which this paper focuses. Then, we substantiate the relevance of service composition. At the end of this chapter, we explain the goal and structure of this paper.

1.1 Service Composition

Service composition is vital to support fruitful inter-organizational collaborations in the dynamic, specifically e-business contexts (Grefen, 2006). Constructing composite web services from simple and/or composite services is an interesting and challenging research problem.

1.1.1 Services and Service Composition

In the ICT domain, several works published in the literature are related to service composition. However, an agreed upon definition of service is still lacking (Berardi et al., 2005). In this paper, a service is referred to as a set of functionalities to fulfill some business goals, and is represented as a structured business process. To enact services, three basic roles are needed: (i) service provider, who provides services. (ii) service consumer, who is the client looking for services to fulfill its goals. (iii) service directory, which plays the role of providing a repository/registry of service descriptions. The combination of the service providers and the service consumers can be seen by a third party (for instance, a customer) as a single entity. Note that a service provider itself can also use other service providers in the implementation of its services.

Services are usually classified into two categories: elementary service and composite service. An elementary service is a service which provides access to applications that do not rely on other services to fulfill external requests (Grefen, 2006). A composite service defines inter-relationships among participating services. A composite service can be seen as an umbrella structure, which brings together other elementary and/or composite services that collaborate to implement a set of functions to achieve some business goals (Grefen, 2006). Service composition refers to a procedure that reaches from the selection of potential services to the definition of the final composite service.

1.1.2 Business Process Outsourcing Paradigm

Service composition implies inter-operability across organizational boundaries. In the BP-WS context, it implies inter-operability across workflows (business processes) from diverse organizations.

In this paper, we follow the business process outsourcing paradigm in the bilateral scenario; the inter-operability is loosely coupled in the multi-lateral scenario. In the bilateral situation, one service provider provides services to one service consumer (Norta, 2007). In the business process outsourcing paradigm, one part of the service consumer's process is outsourced (Grefen et al., 2006a; Khalaf and Leymann, 2006). However, the outsourced part of the business process is not seen as an atomic task in the consumer's process. In the contexts of business process outsourcing, service composition implies that the business process from the service provider is embedded into that from the service consumer (Grefen et al., 2006a). In the multilateral situation, several service providers provide services to one service consumer (Norta, 2007); the inter-operability is loosely coupled. Loosely coupled inter-operability implies that the participants of the collaborations are organized in a peer-to-peer topology; the business organizations collaborate at the same level, but are not organized hierarchically (Grefen et al., 2007). The loosely coupled inter-operability also implies that the definitions of the internal processes of the service providers are local. This form of collaboration means that each part of the collaboration operates independently, and synchronizes with the other parts at certain points (Aalst and Weske, 2001).

1.2 Relevance of Service Composition

Nowadays, product life cycles are shorter than ever before. This trend means that time used on the developments of new physical products (products, in short) and/or non-physical services (services, in short) are shorter than ever before. We can also observe that the complexities of products and services are increasing, e.g., more and more functionalities are integrated into new products and/or services (Grefen et al., 2007).

Organizations must now be competitive, efficient and flexible to utilize their resources. Business organizations are now focusing on their core business processes—on which they are best (have competitive competencies). They are now relying on other organizations to deliver "non-core" products and/or services to retain (or build) competitive advantages (Grefen et al., 2007). Thus, production activities are organized around supply chains.

The benefits for the participating organizations of supply chains are obvious. The main benefits can be observed from improved performances in delivering on time, and from a better responsiveness to customers' urgent and un-expected requests (Business link, 2007). However, organizing production activities around supply chains relies heavily on inter-organizational collaborations.

The agile business environments make the problem of the collaborations in supply chains very complex. The collaboration relationships must now be able to change in accordance with a specific service requested by the customers. And the formation of collaboration relationships is desired to be done on the fly. In ICT domain, inter-organizational collaboration means that service functionalities offered by diverse, autonomous and heterogeneous organizations are connected and integrated over the Internet, thus services are composed. To enable time and cost efficient service composition, automatic or semi-automatic electronic means are required for selecting, executing and monitoring services.

1.3 Goal and Structure of This Paper

In this paper, we focus on how to compose open-box services. We analyze the problem of service composition in the context of the BP-WS enabling technologies/specifications. We suggest two procedures. Firstly, we propose a semi-automatic procedure for composing a set of services with data flow dependencies into a structured composite service. This approach is an extension of work (Eshuis et al., 2006). Secondly, we propose an automatic procedure for synchronizing the composite service with the local services using message exchanges. We exclusively focus on asynchronous communication in this paper. This decision is based on an intention to build loosely coupled services; synchronous communication implies a tight coupling (Muth et al., 1998). In these two procedures, the services are modeled as structured business processes, and both the procedures exclusively focus on control flow aspects. We also discuss how our approach can be architecturally supported.

The structure of this paper is as follows. In Chapter 2, we discuss the concepts and models which are used in the sequel of this paper. In Chapter 3, we extend the approach proposed by Eshuis et al. (2006) to support the open-box model in service composition. In Chapter 4, the approach for synchronizing the composite service with the local services is introduced. We also discuss the architecture supporting our approach in this chapter. In Chapter 5, we put the developed procedures to the test by applying it to a case study from the healthcare domain for the bilateral context, and a case study from the car insurance domain for the multilateral context. We conclude the paper in Chapter 6, which presents the main observations from the work described. We also suggest future research directions in this last chapter.

Chapter 2

PRELIMINARIES

In this chapter, we lay the foundation of this paper by explaining the main concepts and models used in the sequel. In section 2.1, we discuss BP-WS and the Open-box model thoroughly. Next, we introduce workflow technology. Since our approach is an extension of work (Eshuis et al., 2006), we also briefly review work (Eshuis et al., 2006) in this chapter.

2.1 BP-WS and Open-box Model

In this section, we first discuss BP-WS. Next, we introduce the open-box model. In the last sub-section, we discuss the characteristics of the composite service in the open-box model.

2.1.1 BP-WS

BP-WS is an extension of the basic web service. A BP-WS is a web service that "has an internal business process specification that can be accessed externally through a number of dedicated web service interfaces (ports)" (Grefen et al., 2006a). A web service now has an internal process structure which is a reflection of the business process offered by the service provider. The state of the execution of the internal process can also be observed; and external parties can also control the execution of the internal process (Grefen et al., 2006a).

2.1.2 Black-box Model and Open-box Model

In addition to the process specifications, another factor that influences the interactions among the collaborating organizations is control flow interface. Four control flow interface classes are proposed by Grefen et al. (2006a). They are black-box, glass-box, half-open-box and open-box. This paper exclusively focuses on the open-box class. However, we next start with a brief introduction to the black-box class model to facilitate understanding.

2.1.2.1 Black-box Model

Following a black-box class model (black-box, in short), the behaviors of one organization are treated as a set of inputs and outputs. All details of the internal operations within the organizations are encapsulated (Grefen et al., 2006a).

2.1.2.2 Open-box Model

The open-box class model (open-box, in short) is defined as a service model, in which "there can be arbitrary control flow relations between consumers and providers. The execution progresses of both parties depend on one another." (Grefen et al., 2006a).The open-box model specifies the detailed behaviors within one organization as a process. And more importantly, this process specification is externally visible. All these visible activities form up the external level process of the organization. (For more information concerning the external level and the internal level process views, we refer to Appendix A.)

A diagram illustrating the open-box model is presented in Figure 2.1. As shown in Figure 2.1, the service provider exposes activities "A", "B", "C" and "D". These activities are collectively offered as a service. The service consumer exposes activities "X", "Y", "Z". The arcs represent the control dependencies. For example, the arc that goes from activity "X" to activity "A" represents that activity "A" depends on activity "X". As shown in Figure 2.1, there is an explicit control flow between intermediate steps of the local process both from service provider to consumer and vice versa.

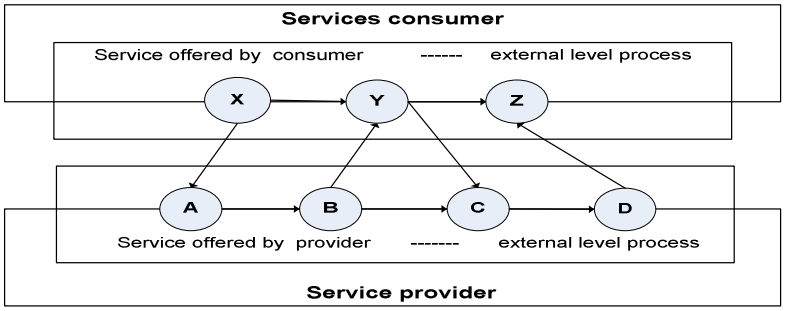

Figure 2.1: Open-box model

2.1.3 The Characteristics of the Composite Services in the Open-box Model

A composite service in the open-box model has following characteristics (Preuner and Schrefl, 2005):

- The overview of a composite service is a complex process, which restricts the potential sequences of activity invocations.
- The exposed functionalities (activities) of a service can be observable or invocable; or they do not need to be exposed.
- Interrelationships between services can be identified at the level of individual activity.

An observable activity is an activity "which is executed by the service provider and whose execution can be observed by the requester" (Preuner and Schrefl, 2005). An invocable activity is an activity "whose

invocation is at the requester's responsibility, whereas the service provider executes this activity but does not invoke them himself" (Preuner and Schrefl, 2005). An activity from a service is considered invocable if and only if its execution is constrained by the inter-relationships with the other services (Preuner and Schrefl, 2005).

2.2 Definition of Structured Process

The combination of SOC and workflow management (WFM) technology has been proposed as a promising approach to compose open-box services. "In this combination, SOC provides for dynamic inter-organizational interoperability and WFM technology provides for core business process management." (Grefen, 2006).

The inputs for our approach to compose open-box services are services represented as structured business processes (structured processes, in short). A structured process is "a process where each or-split has a corresponding or-join and each and-split has a corresponding and-join (Kiepuszewski et al, 2000)".

Structured processes are the most restrictive type of processes (Kiepuszewski et al., 2000). The reason why we chose structured processes as our inputs is three-fold. Firstly, most of the process enactment tools support structured process models, even though the un-structured process models are more expressive (Kiepuszewski et al., 2000; Aalst et al., 2003). Secondly, the verification and the implementation of structured processes are easier. Thirdly, two big flaws possible in unstructured processes — deadlock and multiple instances of the same activity active at the same time — can not appear (Eshuis et al., 2006).

2.3 Overview of Black-box Service Composition

Eshuis et al. (2006) proposed an approach to compose black-box services. Our work builds upon work (Eshuis et al., 2006), of which we now provide an overview.

2.3.1 General Overview of Work (Eshuis et al., 2006)

Eshuis et al. (2006) have developed a semi-automatic approach to compose black-box services. The inputs are a set of black-box services with dataflow dependency. The output is the composite service expressed as a structured business process. The approach consists of three steps. First step is the deriving of the dependency graphs. Two kinds of dependency graphs are derived: abstract dependency graph and concrete dependency graph. Concrete one specifies types for branching points. The second step is constructing the structured composite service which takes the abstract dependency graph as input. In the last step, the concrete dependency graph is used to type the structured process model.

The advantage of the approach proposed by Eshuis et al. (2006) is three-fold. Firstly, the semi-automatic approach proposed by Eshuis et al. (2006) itself requires manual inputs, however, a large part of it is fully automated. Secondly, the user inputs required on annotating the dependency graphs are much less than annotating services with formal pre- and post-conditions, which is required by most other comparable service composition approaches. Thirdly, the structured composite service, which makes use of only basic process patterns, can be encoded straightforwardly into any process languages (Eshuis et al., 2006). These advantages are also the reasons why this paper is based on work (Eshuis et al., 2006).

2.3.2 Blocks and Dependency Graphs

Eshuis et al. (2006) choose a hierarchical view of the formalization of the structured process models. In a hierarchical view, leaf nodes are individual activities and non-leaf nodes are blocks. Two kinds of blocks are considered: composite blocks of type COMP and sequential blocks of type SEQ. A COMP block can be of type AND or XOR. A COMP block has a set of children, whereas a SEQ block has a list of children. For example, COMP {SEQ [X, Y], SEQ [Z]} is a block in which activity X is done before activity Y, and both are executed in parallel with or exclusive to activity Z. In the graphical syntax, there is a split and a join node at the beginning and the end of each block (Eshuis et al., 2006).

Eshuis et al. (2006) derive graph-based compositions by analyzing input/output dependencies between black-box services. In graph-based compositions, services are coordinated by means of control flow constructs, such as AND-splits, AND-joins, XOR-splits and XOR joins (Eshuis et al., 2006). If treating each atomic activity as a black-box service, we can define Concrete Dependency Graphs among activities (CDG). The CDGs are used as inputs in our approach as explained later on. If activity *a* outputs a data item with a certain type and activity *b* needs as input a data item with the same type, then *b* depends on *a*. The dependencies between a set of individual activities are captured in a graph. To keep consistency with work (Eshuis et al., 2006), we reuse the notation from work (Eshuis et al., 2006) in this paper.

The CDG is a tuple (A, E, *join, fork*) with

— $A = \{a_1, a_2, \ldots, a_n\}$ a set of activities.

— $E \subseteq A \times A$ a set of dependencies.

and functions *join* and *fork* label respectively the incoming and outgoing dependencies of an activity with the branching type:

$$join, fork: A \rightarrow \{AND, XOR\}$$

One example concrete dependency graph is shown in Figure 2.2.

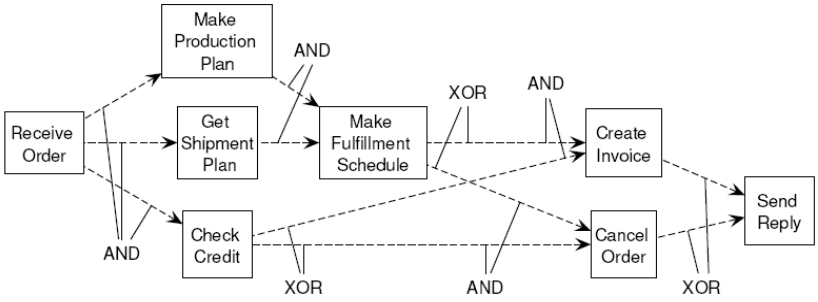

Figure 2.2: Example concrete dependency graph (taken from (Eshuis et al., 2006))

In Figure 2.2, the activities are modeled as rectangles. The arrows represent the dependency relationships. For example, activity “Make Production Plan” depends on activity “Receive Order”, then the arrow goes from “Receive Order” to “Make Production Plan”. The incoming and outgoing dependencies of

a service are typed with AND or XOR. Only one type, either AND or XOR, can be assigned to incoming resp. outgoing dependencies.

2.3.3 Definition of a LSPM (Language of Structured Process Models)

We next give a formal definition of a LSPM (Language of Structured Process Models) below, where $A \neq \emptyset$ is a set of activities:

1. if $x \in A$, then $x \in$ LSPM.
2. if $x_1, \ldots, x_n$ ($n \geq 1$) and $x_i \in$ LSPM (if $i > 0$ and $i \leq n$), then SEQ $[x_1, \ldots, x_n] \in$ LSPM.
3. if $x_1, \ldots, x_m$ ($m \geq 2$) and $x_i \in$ LSPM (if $i > 0$ and $i \leq m$), then COMP $[x_1, \ldots, x_m] \in$ LSPM.
4. nothing more.

Note, in the above definition a COMP block can be of type AND or XOR. We have one function *type: COMP* → {AND,XOR} which is used to assign the exact type to each COMP block (Eshuis et al., 2006).

To facilitate understanding, we can transfer structured process models expressed in LSPM into graphical process models, and the other way round. To illustrate this, consider one artificial service, service A. This service is represented as a structured business process. Service A can be expressed as SEQ[SEQ[A1, A2, A3], COMP {SEQ[A4], SEQ[A5]}, SEQ[A6]]. We next transfer this structured process model into a graphical model. First, all elements in a SEQ block form up a sequential chain. This means that the SEQ[A1, A2, A3] is executed first, next COMP{SEQ[A4], SEQ[A5]}, the last SEQ[A6]. Within one COMP block the SEQ blocks are executed in parallel with or exclusive to the other. In our case, SEQ[A4] and SEQ[A5] are done in parallel. In other words, this COMP block is typed with AND. We show a tree model for service A in Figure 2.3. The equivalent graphical process model for this service is specified in Figure 2.4. In Figure 2.4, the activities are modeled as rectangles. Circles indicate splits (more than one outgoing edge) or joins (more than one incoming edge). A in a circle indicates a split or a join of type AND. In the graphical process model, a SEQ block is represented as an activity chain; and a COMP block is represented as a set of activities connected by the same split and join pair. For instance, SEQ[A1, A2, A3] is represented as a chain consisting of three activities—A1, A2, and A3; COMP {SEQ[A4], SEQ[A5]} is represented as two activities—A4 and A5 connected by the same split and join pair. Note, to properly represent a COMP block, the split must be before join. By means of arrows, we show that some activities are executed after other activities in a SEQ block. For example, the arrow going from activity "A1" to activity "A2" indicates "A2" is executed after "A1" is executed.

Transferring a graphically expressed structured process to a structured process model expressed in LPSM is more straightforward, so we do not present it here.

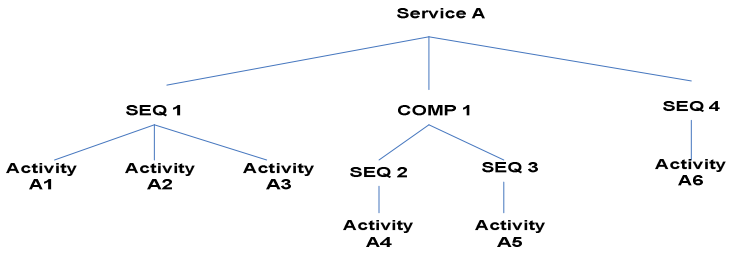

Figure 2.3: Tree model for service A

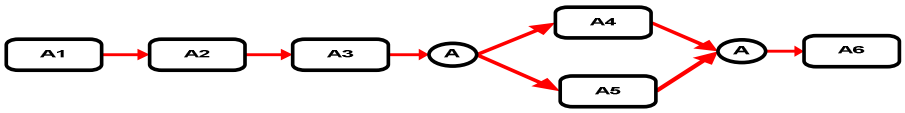

Figure 2.4: Example process for service A

2.3.4 Algorithm to construct structured composite service

Eshuis et al. (2006) proposed one algorithm which takes as input a dependency graph and returns a structured composition satisfying the input dependencies. This construction algorithm is listed below in Figure 2.5. In the main phase of the algorithm, the structured composition is iteratively constructed by processing black-box services in *S*. The set of *Initial(S, E)* contains services which do not depend on any other service. Function *ConstructBlock(X)* composes a given set X of services into either a single service (if X is singleton), or otherwise into a composite block consisting of a set of sequential blocks, each containing one service from X. Function *next* returns the services to be processed next whose input data can be delivered by previously processed services. Services in the set *toprocess* cannot be processed one by one. One service s_1 can directly influence the other service s_2, if the pre-conditions of the services are overlap; one service s_1 can also in-directly influence the other service s_2, if there is another service *s* that directly influences both two services, s_1 *and* s_2 . The set *Input I* of services in processed is defined as services on which services in I depend. Finally, concrete dependency graphs are used to type composite COMP nodes in the structured composition returned by the algorithm. Definition 2 introduced in the last sub-section specifies how to type the composite COMP nodes.

The structured composite service shown in Figure 2.6 is derived using the algorithm from Figure 2.2. In this composite service, O in a circle indicates a split or a join of type XOR. To construct this composite service, firstly, the initial set contains activity "Receive Order". This activity is put in SEQ[Receive Order]. Next, "Make Production Plan", "Get Shipment Order" and "Check credit" are put in the *toprocess*. All these three activities directly influence each other, they are tried as a group and put in a COMP block {SEQ[Make Production Plan], SEQ[Get Shipment Order], SEQ[Check credit]}. This COMP block is attached to the SEQ[Receive Order]. Next, activity "Make Fulfillment Schedule" is processed. The other activities can also be processed following the same vein. The last step is typing the COMP blocks in according with the concrete dependency graph.

```
1: procedure StructuredComposition((S, E))
2:     C := SEQ[constructBlock(Initial(S, E))]
3:     processed := Initial(S, E)
4:     while processed ≠ S do do
5:         toprocess := next(processed)
6:         for each maximal influencing subset I of toprocess do
7:             BlockI := constructBlock(I)
8:             InputI := {s ∈ processed|post(s) ∈ I}
9:             N := the most nested block in C containing all services in InputI.
10:            if N is composite then
11:                NotPreI := {c ∈ children(N)|InputI ∩ services(c) = ∅}
12:                if NotPreI ≠ ∅ then
13:                    PreI := COMP{c ∈ children(N)|InputI ∩ services(c) ≠ ∅}
14:                    N' := COMP({SEQ[PreI, BlockI]} ∪ NotPreI)
15:                    replace N by N' in C
16:                else
17:                    parent(N).append(BlockI)
18:                end if
19:            else
20:                parent(N).append(BlockI)
21:            end if
22:            processed := processed ∪ I.
23:        end for
24:    end while
25:    return C
26: end procedure
```

Figure 2.5: Algorithm for constructing structured compositions (taken from (Eshuis et al., 2006))

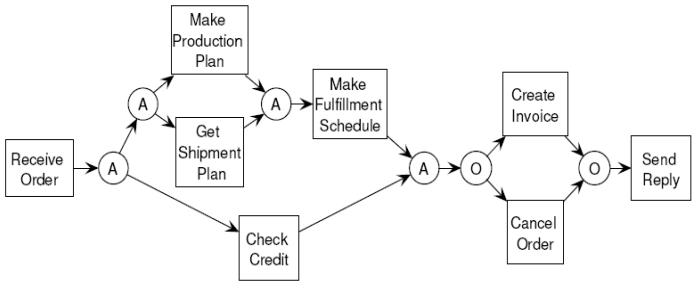

Figure 2.6: Structured composite service for Figure 2.2 (taken from (Eshuis et al., 2006))

To employ the algorithm, each dependency graph must satisfy two constraints. The dependency graph is acyclic; and if there is an edge from activity a_1 to a_2, then there is no path with length greater than 1 from a_1 to a_2 (Eshuis et al., 2006). The relaxation of the first constraint is planned as future work. The second constraint is needed for the algorithm; however, the violation of the second constraint can be easily repaired.

Chapter 3

PROCEDURE FOR COMPOSING OPEN-BOX SERVICES

In this chapter, we propose our approach to compose open-box services. We first consider the bilateral scenario; in the last section, we extend our approach to the multi-lateral scenario.

In this paper, the composition procedure starts from the services presented as structured business processes; and the services have already been selected for composition in accordance with some sophisticated rules, such as (Eshuis and Grefen, 2007a). Eshuis and Grefen, (2007a) propose a proposal on a procedure for finding BPEL services using matching.

3.1 Motivation for a New Approach

To compose open-box services, the most obvious way is probably putting two processes in parallel. The parallel processes form up the global process of the composite services. However, our desired composite services should be structured processes. Simply putting two structured processes coming from two different organizations in parallel does not ensure that the global process is structured. The inter-organizational dependencies may interrupt the properly nested pairings of splits and joins (characteristic of structured processes), resulting in an un-structured composite service.

To illustrate this, consider two artificial services from two different organizations. These services are represented as structured business processes. Service A can be expressed as SEQ[SEQ[A1, A2, A3], COMP {SEQ[A4], SEQ[A5]}, SEQ[A6]]. The CDG and the structured process model for service A is specified in Figure 3.1 respectively Figure 3.2. Service B is expressed as SEQ[SEQ[B1, B2, B3], COMP {SEQ[B4], SEQ[B5]}, SEQ[B6]]. The CDG and the structured process model for service B in Figure 3.3 respectively Figure 3.4. The structured process models can be derived by the algorithm introduced by Eshuis et al. (2006) from the CDGs.

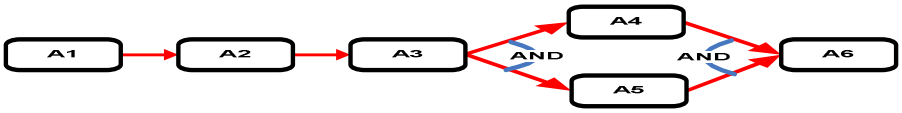

Figure 3.1: Example CDG for service A

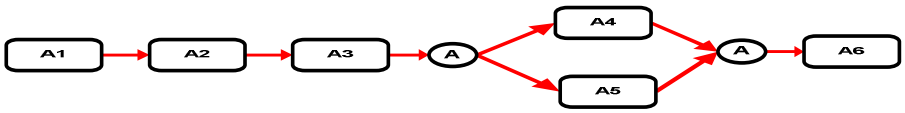

Figure 3.2: Example Process for service A

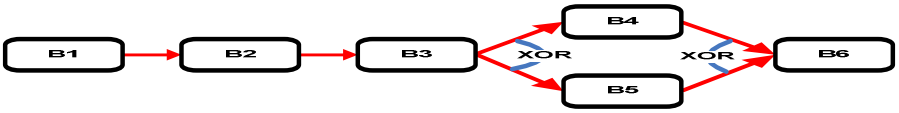

Figure 3.3: Example CDG for service B

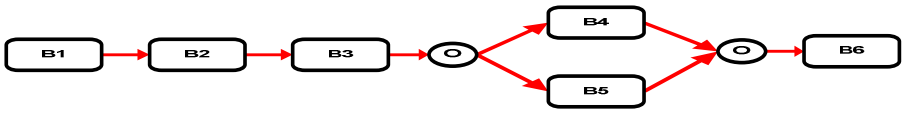

Figure 3.4: Example Process for service B

By adding inter-organizational dependencies between CDGs for service A and for service B, a new concrete dependencies graph which contains both the CDGs for service A and for service B is derived. Figure 3.5 shows a new concrete dependency graph. In Figure 3.5, the inter-organizational dependencies are modeled as dotted arrow. For instance, activity "A2" in service A depends on activity "B2" in service B; activity "B4" in service B depends on activity "A5" in service A.

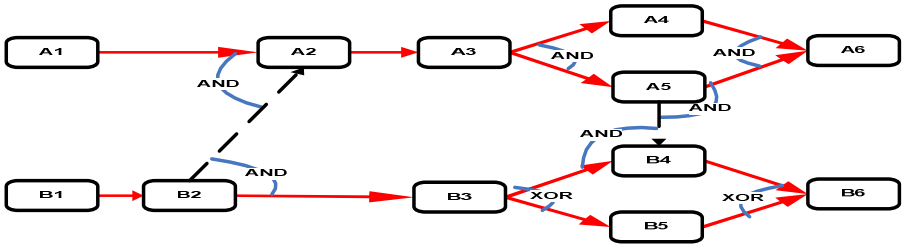

Figure 3.5: The CDG between service A and service B

Figure 3.6 shows an example composite service in which two structured services are simply put in parallel. In Figure 3.6 the frame can be seen as a general parallel construct. The upper side is service A, the bottom side service B. The boundary between these two services is modeled as dotted line. "C" inside a circle, which can be either AND or XOR, is the generic notion of a split or a join. From this figure, we can see, the nested paring of split and join of the COMP {SEQ[A4], SEQ[A5]} in service A (Fig 3.2) has been interrupted by an inserted split "C4"; the COMP {SEQ[B4], SEQ[B5]}in service B (Fig 3.4) has been interrupted by an inserted join "C3". The global process of the composite service is un-structured. One possible remedy to this approach is transferring the unstructured process to structured process. However, transferring the unstructured process to structured process has been proved not to be easy; since, process models can consist of parallelism while programs are sequential (Eshuis et al., 2006).

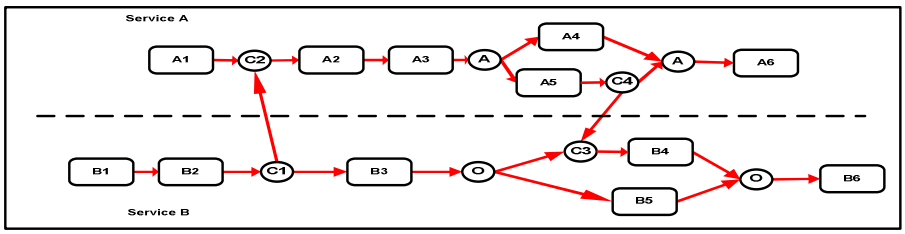

Figure 3.6: Example composite service of service A and B — simply putting two structured services in parallel

Since this simple method is not suitable in our context, we must find a more sophisticated approach to compose open-box services.

Even though the approach introduced by Eshuis et al. (2006) does not directly support the open-box model, it gives us some important inspirations. Firstly, the approach proposed by Eshuis et al. (2006) can convert dependency graphs into structured processes. Secondly, the algorithm works well in the black-box contexts. If we can fragment one process into multiple segments, each of which is treated as a black-box, then we can still use the algorithm proposed by Eshuis et al. (2006) to construct structured composite services.

Our problem now becomes how to fragment the structured processes. In the spectrum of this fragmenting, treating the whole service as a single black-box is at one end; treating each individual activity as a black-box at the other end. In the open-box model, we can not treat the whole service as one single black-box, because that would result in the black-box model. Next, we argue why treating each individual activity as one single black-box is also not a good choice.

Treating each individual activity as a black-box, we now use the algorithm proposed by Eshuis et al. (2006) to compose the above two services, A and B. The output is shown in Figure 3.7. One possible desired composite service complying with the inputs in Figure 3.1 through Figure 3.5 is shown in Figure 3.8. We can now compare these two composite services. In the desired composite service, activity "B4" and "B5" are in an XOR construct; either "B4" or "B5" is executed, but not both. However, in Figure 3.7, both "B4" and "B5" are executed. For this reason, the composite service which treats each individual activity as a black-box service is not desired.

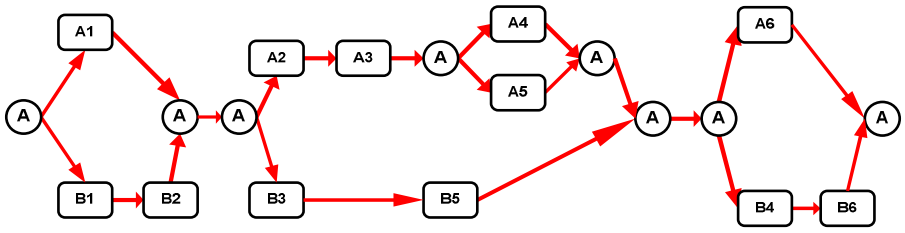

Figure 3.7: Composite service for A and B — treating each individual activity as a black-box

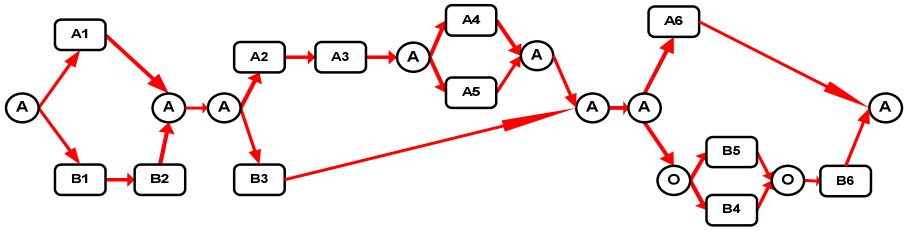

Figure 3.8: One possible desired composite service for A and B

The reason for this kind of undesired composite services is that we did not treat the COMP blocks which are influenced by the inter-organizational dependencies as groups. More specifically, we did not treat the COMP{SEQ[B4], SEQ[B5]} as a group in the example. To see why, we need to reexamine the example. "B4" and "B5" are in a COMP block of type XOR in the original process of service B. This relationship between "B4" and "B5" must be respected in the desired composite service; if "A5" can trigger "B4", "B5" must be impeded. We can also image another case in which "B4" and "B5" are in a COMP block of type AND. In this new case, when "A5" triggers "B4", "B5" must be triggered at the same time. This is to say even though no direct dependencies between "A5" and "B5" are observed in the dependency graph, "A5" has influence on "B5". To transfer this kind of direct and indirect influences into control flow, we need to treat the COMP blocks which are influenced by the inter-organizational dependencies as groups in the composition procedure.

The basic idea of our approach to compose open-box services is treating the COMP blocks involved in the inter-organizational dependencies as groups (aggregates) in the composition procedure. Next, treating each group (aggregate) and activity as a black-box service, we reuse the algorithm proposed by Eshuis et al. (2006) to construct the structured composite services. The last step is manually typing the branches of the composite services.

3.2 Construct Dependency Graphs Which Fulfill the Requirements of the Algorithm in Work (Eshuis et al., 2006)

In this section, we first discuss how to properly treat the COMP blocks which are involved in the inter-organizational dependencies as groups in the composition procedure. Next we present how to construct new dependency graphs which fulfill the requirements of the algorithm in work (Eshuis et al., 2006).

3.2.1 The Most Abstract COMP Block

We now introduce one auxiliary function — most abstract COMP block. This function is next used to aggregate the COMP blocks which are involved in the inter-organizational dependencies in the composition procedure.

The most abstract COMP block. For a set A of activities, the most abstract COMP block (*mac*) of A, denoted *mac*(A), is the COMP block *c* such that *c* is ancestor of each activity in A, and every other COMP block *x* that is ancestor of each activity in A, is child of *c*.

For example, a segment of a structured process is shown in Figure 3.9. There are two COMP blocks, COMP 1- COMP {SEQ[SEQ[A2],COMP{SEQ[A3], SEQ[A4]}, SEQ[A5]], SEQ[A7]} and COMP 2 –

COMP{SEQ[A3], SEQ[A4]}. We now assume activity "A4" is involved in the inter-organizational dependencies. The *mac({A4})* is COMP 1. Even though COMP 2 is an ancestor of "A4", it is a descendant of COMP 1.

We now argue why we must treat the most abstract COMP block — COMP 1 as a group in the composition procedure, but not the COMP block which is the immediate ancestor of an activity — COMP 2, in our example. Still use the example in Figure 3.9, activity "A4" is involved in one inter-organizational dependency; this implies that the activity (activities) in the other services has a direct influence on "A4". However, some indirect influences can be observed on the other activities in COMP 1. For instance, if "A4" is triggered, "A7" must be triggered as well. If we aggregate COMP 2, then only the indirect influence on "A3" is included, but not on "A7". A proper way to include all these direct and indirect influences is that we treat the most abstract COMP block, COMP 1 in our example, as a group in the composition procedure.

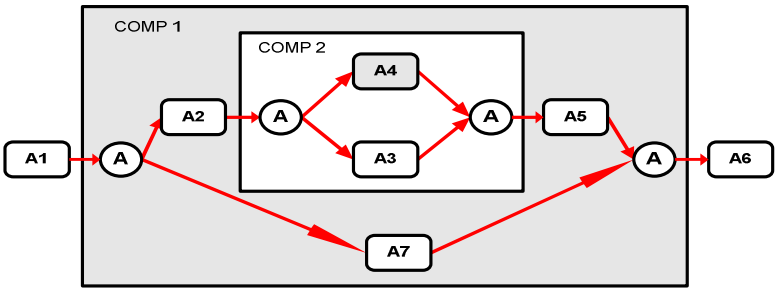

Figure 3.9: A segment of a structured process

3.2.2 Construct a New Dependency Graph Containing aggregates

To fully take advantage of the algorithm proposed by Eshuis et al. (2006), we need to construct a new concrete dependency graph which treats the activities of some COMP blocks in the structured process models as groups in the composition procedure. Which COMP blocks are needed to be treated as groups will be explained next. The inputs for constructing such a dependency graph are the CDGs (provider side and consumer side) from which the structured process models of services can be derived using the algorithm in (Eshuis et al., 2006), the structure process models (provider side and consumer side) and the inter-organizational dependencies between these services. Each inter-organizational dependency is represented as a pair of activities in different services and the direction of this dependency. The output of this procedure is a new concrete dependency graph. In this new concrete dependency graph, all the activities within a COMP block in the structured process models which need to be treated as a group in the composition procedure are replaced with a dummy activity — aggregate. We call this new dependency graph DGA (Dependency Graph containing Aggregates). Besides including aggregates, one important difference between DGA and normal CDG which contains aggregates is that a DGA involves inter-organizational dependencies. A formal definition and the notion of the DGA, which is similar to the definition and the notion of the CDG, can be found in Appendix B.

We divide the constructing procedure into two phases. In phase I, the first step is deciding on which COMP blocks within one structured process model (say, Process A) needs to be treated as groups; the second step is replacing all the activities within the COMP blocks, which have been decided to treat as groups in step 1, with dummy activities — aggregates in the CDG from which Process A can be derived; the third step is constructing a new CDG which now contains aggregates and activities, and all these aggregates

and activities belong to one single service. Step 1 through step 3 can be repeated until all the services which we want to compose have been processed. In phase II, inter-organizational dependencies are added between the CDGs which are the outputs of phase I, resulting in a DGA. Next we illustrate these two phases in detail.

PHASE I:

Step 1: Decide what COMP blocks need to be treated as groups

The inputs of this step are structured process models and the inter-organizational dependencies; the output is a set of COMP blocks. We use rule 3.1 to decide on what COMP blocks needed to be treated as groups in the composition procedure.

Rule 3.1

In a structured process model, if one activity *y* is involved in the inter-organizational dependencies, and this activity *y* is a descendant of some COMP blocks in the structured process model, then the *mac({y})* needs to be treated as a group in the composition procedure.

Step 2: Replace all the activities within one COMP block which is the output of step 1 with a dummy activity, which aggregates the activities.

The inputs of this step are the CDG from which either the consumer or the provider process can be derived and the output of step 1. The output of this step is a set of activities. This set of activities includes some dummy activities — aggregates.

This step consists of two sub-steps. In sub-step 2.1, the dependencies among activities in the CDG are removed, resulting in a set of activities; in sub-step 2.2, taking the output of sub-step 1 as inputs, all the activities within one COMP block which is the output of step 1 are replaced with a dummy aggregate activity, resulting in a set of activities, including some dummy aggregate activities. Sub-step 2 can be repeated until all the COMP blocks which are the outputs of step 1 are processed.

Step 3: Construct the dependencies among activities and/or aggregates

Next, we use rule 3.2 through rule 3.5 to construct dependencies within the output of step 2. The inputs of this step are the original CDGs (for consumer side respectively provider side) which do not contain any aggregates and the output of step 1 and step 2. The output of this step are two new abstract dependency graphs (for consumer side respectively provider side) which contain dummy activities — aggregates; all the activities within one new abstract dependency graph belong to one single service, either service consumer or service provider.

Rule 3.2: an activity depends on an aggregate, if the activity depends on an activity which is replaced by the aggregate in the CDG. When one activity depends on more than one activity which are replaced by the same aggregate, all these dependencies are combined and represented as one arrow. The direction of this arrow is from the aggregate to the activity.

Rule 3.3: an aggregate depends on an activity, if one activity which is replaced by the aggregate depends on that activity in the CDG. If more than one activity which is replaced by the same aggregate depends on that activity, all these dependencies are combined and represented as one arrow. The direction of this arrow is from the activity to the aggregate.

The above two rules are used to set up dependencies between aggregates and activities. The direction of these dependencies must be clearly identified, so we state these similar rules separately.

Rule 3.4: dependencies between activities which involve no activity (activities) replaced by the aggregates are the same as in the CDG.

This rule is used to set up dependencies between activities which are not influenced by the aggregates.

Rule 3.5: an aggregate (say aggregate A) depends on an aggregate (say aggregate B), if one activity replaced by aggregate A depends on one activity replaced by aggregate B in the CDG. If more than one activity replaced by aggregate A depend on activity (activities) replaced by aggregate B, all these dependencies are combined and represented as one arrow. The direction of this arrow is from aggregate B to aggregate A.

This rule is used to set up dependencies between two aggregates. This step can be repeated, until all the services which we want to compose have been processed.

PHASE II:

The input of this phase is the output of phase I and the inter-organizational dependencies. The output is a DGA.

The rules used to properly set up inter-organizational dependencies among the outputs of phase I are basically the same with the rules used in step 3 phase I. For instance, if one activity (say activity A, in service A) depends on one activity in the other service (say activity B, in service B), and activity B is replaced by an aggregate (say aggregate B), then activity A depends on aggregate B, represented as an arrow from aggregate B to activity A. We do not repeat all these rules again.

As an example, the DGA complying with inputs from Figure 3.1 through Figure 3.5 is shown in Figure 3.10. In Figure 3.5, two inter-organizational dependencies, "B2" to "A2" and "A5" to "B4", are observed. Applying rule 3.1, two COMP blocks *mac({B4}) and mac({A5})* need to be treated as groups in the composition procedure. Next, *mac({B4}) and mac({A5})* are replaced with two aggregates, "aggregate A" and "aggregate B". Using rule 3.2 through 3.5, the dependencies among activities and aggregates of all the services are properly set up. For example, in Figure 3.5 "B4" depends on "A5", since "B4" is replaced by "aggregate B", "A5" is replaced by "aggregate A", "aggregate B" depends on "aggregate A".

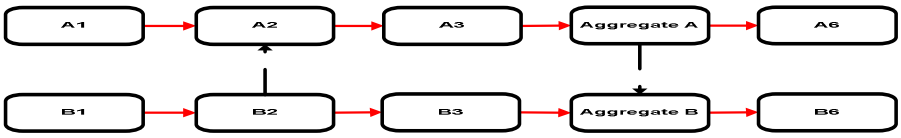

Figure 3.10: Example for constructing the DGA

3.2.3 Constraints and Resolutions

To successfully employ the algorithm in (Eshuis et al., 2006), the DGAs need to satisfy some constraints. In line with work (Eshuis et al., 2006), we define the constraints below.

C1: The dependency graph is acyclic (Both in the CDG and in the DGA).

C2: If there is an edge from activity1 (aggregate 1) to activity2 (aggregate 2), then there is no path with length greater than 1 from activity1 (aggregate 1) to activity2 (aggregate 2).

Next our discussion focuses on violation and resolution for the DGAs. We assume that there is no violation in the CDGs; some violations are only observed in the DGAs. With respect to discussions of violations in the CDGs we refer to (Eshuis et al., 2006). This is because each individual activity can be treated as a black-box service, and the resolutions introduced by Eshuis et al. (2006) are perfectly suitable in our context.

3.2.3.1 Violation of Constraint 1

We consider this violation in two cases. Case 1: two aggregates are involved in the violation. Case 2: only one aggregate is involved in the violation. We start with case 1.

In Figure 3.11, the CDG is acyclic. It is cyclic in the DGA, as shown in Figure 3.12.

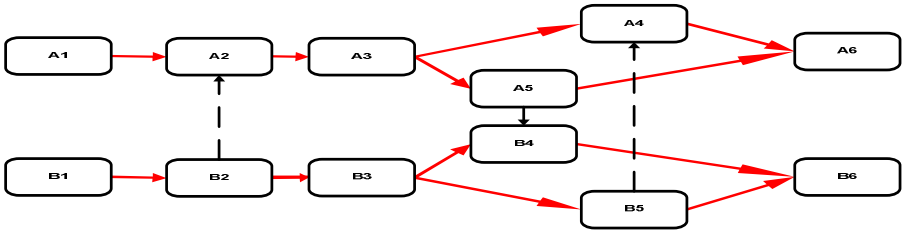

Figure 3.11: Acyclic CDG—case 1

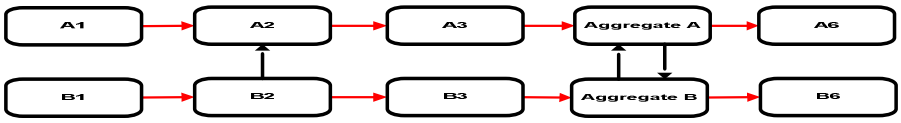

Figure 3.12: Cyclic DGA—case 1

In case 1, the solutions are classified into three categories. Category I deals with the violation if both the involved aggregated COMP blocks which cause the cyclic are of type AND; category II, if both the aggregated COMP blocks are of type XOR; Category III, if one aggregated COMP block is of type AND, and the other one of type XOR.

- Category I: Transform one of the aggregated COMP blocks which cause the DGA to be cyclic to multiple SEQ blocks; then reconstruct the DGA.

After transforming, these SEQ blocks are executed in a sequential order. This is a reasonable solution, as the activities in a parallel block (COMP) can be executed in an arbitrary order (Aalst et al., 2004). Sequential order is one possible order. In Figure 3.13, we present one solution to the violation shown in Figure 3.12. Note, we insert a dummy activity, or it will violate constraint 2 as explained next.

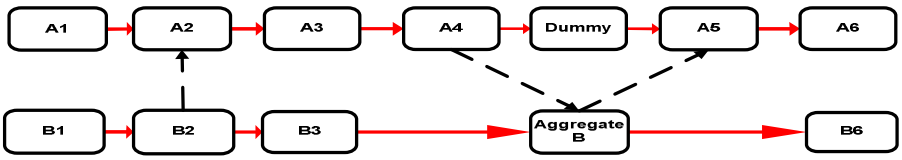

Figure 3.13: Solution of violation of constraint 1— COMP block of type AND (case 1)

- Category II: Transform one of the aggregated COMP blocks which cause the DGA to be cyclic to multiple new COMP blocks of type XOR; then reconstruct the DGA.

Each new COMP block contains one SEQ block of the original aggregated COMP block. The other branch of this new COMP block is a dummy activity. These new COMP blocks are executed in a sequential order. In Figure 3.14, we present one solution of category II to the violation shown in Figure 3.12. The internal structure of "aggregate A5" is shown in Figure 3.15, as an example.

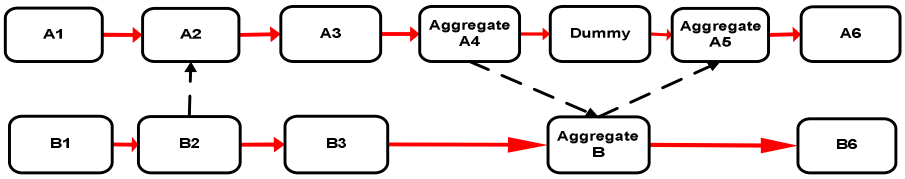

Figure 3.14: Solution of violation of constraint 1— COMP block of type XOR (case 1)

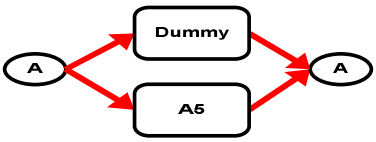

Figure 3.15: Internal structure of aggregate OA5

In the above presented approach, we get a solution to satisfy the constraint; and at the same time, the XOR structure of the aggregated COMP block is maintained. There is a potential problem, however, for this solution. In the original process, there are only two choices; either "A4" or "A5" is executed, but not both. In our solution, however, there are 4 choices. Two choices are not desired: I, neither "A4" nor "A5" is executed; II, both "A4" and "A5" are executed. In case neither "A4" nor "A5" is executed, the instance will be stuck in the original process. This is an exception to the original process. We rule it out of our consideration. The second case is the real problem. For instance, activity "A5" changes a negative value of a variable *V_O* to positive; activity "A4" changes a positive value of *V_O* to negative. We have now an instance with a negative value of variable *V_O*. Firstly, this instance goes through "A5", the value of *V_O* changes to positive; then it goes through "A4", the value of *V_O* changes back to negative. Such an effect is not desired. To avoid this kind of potential problem, the value of the variable used to decide on which branch to take, in our example the value of *V_O,* must be unchanged during this part of the process.

- Category III: Transform the aggregated COMP block of type AND which causes the DGA to be cyclic to multiple SEQ blocks; or transform the aggregated COMP block of type XOR

which causes the DGA cyclic to multiple new COMP blocks of type XOR; then reconstruct the DGA.

The solutions in Category III can be derived from that in Category I and Category II.

A composer now faces one question: "which aggregated COMP block should be decomposed?", in all the above three categories. Decomposing either the consumer's side or the provider's side has the same effect for the solutions in Category I and Category II. The answer depends on the negotiation between the consumer and the provider. A composer may decide to decompose the COMP block at the consumer side; it is under control of the composer, as a composer usually is the consumer. A composer can also decompose the COMP block at the provider side, if the consumer is powerful. In terms of the solutions in Category III, decomposing the aggregated COMP blocks of type AND is preferred. If decomposing the aggregated COMP block of type XOR, we have to re-aggregate the new COMP blocks; and the execution of the new COMP blocks may cause some un-desired effects as argued previously.

We now consider case 2, in which only one aggregate is involved in the violation. In Figure 3.16, the CDG is acyclic. It is cyclic in the DGA, as shown in Figure 3.17.

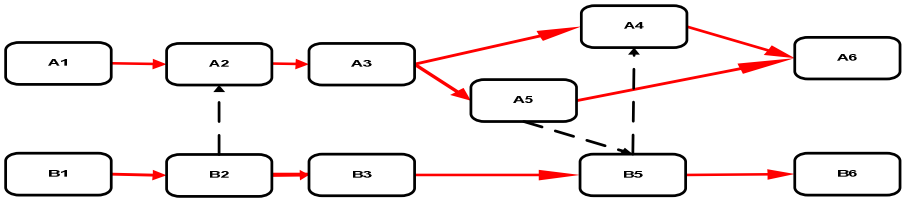

Figure 3.16: Acyclic CDG—case 2

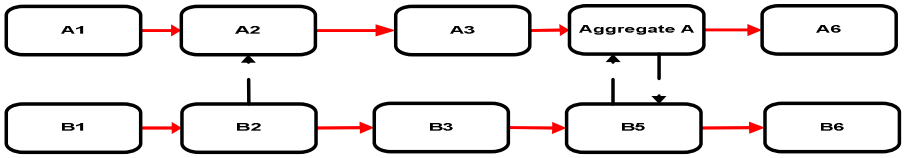

Figure 3.17: Cyclic DGA—case 2

The solution to this kind of violation can be derived from the solutions in Category I and Category II in case 1. If the aggregated COMP block is of type AND, it is decomposed to multiple SEQ blocks. If the aggregated COMP block is of type XOR, it is decomposed to multiple new COMP blocks of type XOR. Then the DGA is reconstructed.

3.2.3.2 Violation of Constraint 2

We consider this violation in two cases. Case 1: two aggregates are involved in the violation; Case 2: only one aggregate is involved in the violation. We start with case 1.

The CDG in Figure 3.18 does not violate constraint 2. However, a violation in the DGA is observed, as shown in Figure 3.19. In case 1, the solutions are classified into two categories. Category I deals with this

violation if the aggregated COMP block involved in both inter-organizational dependencies is of type AND; Category II, if the aggregated COMP block is of type XOR.

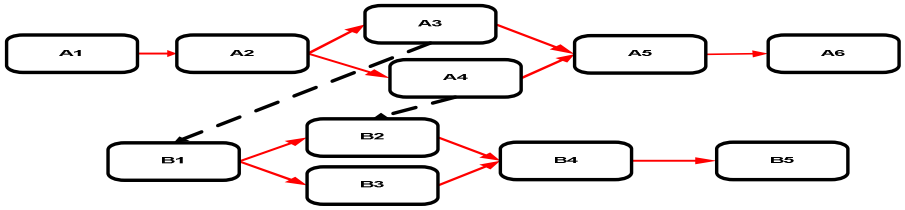

Figure 3.18: Example CDG: violation of constraint 2 — case 1

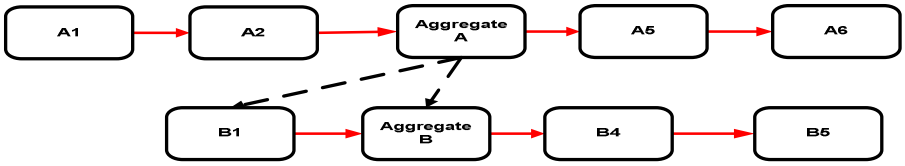

Figure 3.19: Example DGA: violation of constraint 2 — case 1

- Category I: Transform the aggregated COMP block involved in both the inter-organizational dependencies to multiple SEQ blocks; then reconstruct the DGA.

After transforming, the sequential order of the SEQ blocks must respect the dependency relationships at the counter part. In our example, SEQ block "A3" must be executed before "A4". Because B1 depends on "A3", "B2" depends on "A4", and "B1" is always executed before "B2". A solution is shown in Figure 3.20.

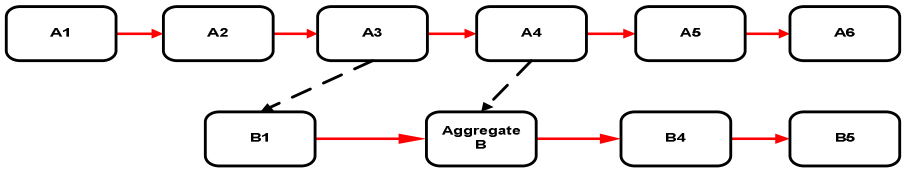

Figure 3.20: Solution of violation of constraint 2— COMP of type AND (case 1)

- Category II: Transform the aggregated COMP block which is involved in both the inter-organizational dependencies to multiple COMP blocks of type XOR; then reconstruct the DGA.

To maintain the XOR structure of the aggregated COMP block, we decompose the COMP block into multiple new COMP blocks of type XOR. All these new COMP blocks need to respect the dependency relationships at the counter part. A solution is shown in Figure 3.21.

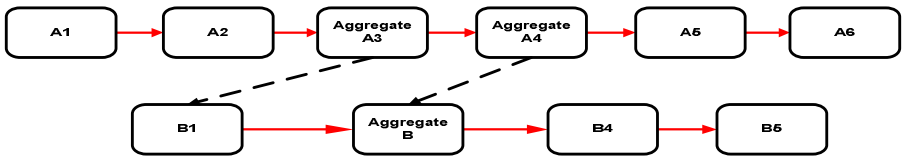

Figure 3.21: Solution of violation of constraint 2 — COMP of type XOR (case1)

In case 1, even though there are two aggregated COMP blocks involved in this violation, only one is involved in both the inter-organizational dependencies. It can be observed if both the aggregated COMP blocks are involved in both the inter-organizational dependencies, these two COMP blocks are cyclic.

Using Figure 3.18 and Figure 3.19, we now argue why we chose to decompose the aggregated COMP block involved in both the inter-organizational dependencies; it is the COMP block containing activity “A4” and “A5”. If we decompose the COMP block involved in only one inter-organizational dependency, in our example it is the COMP block containing activity “B2” and “B3”, we still get a DGA violating constraint II, as shown in Figure 3.22. For this reason, to successfully satisfy the constraint 2 in case 1, the aggregated COMP block involved in both the inter-organizational dependencies needs to be decomposed. In Figure 3.22, we suppose the COMP block containing activity “B3” and “B2” is of type AND. It has the same effect if the COMP block is of type XOR.

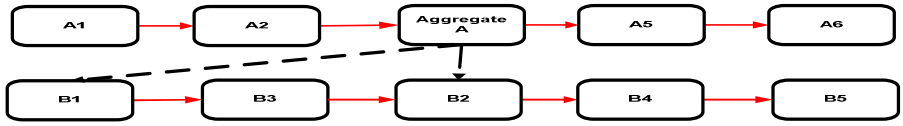

Figure 3.22: Example DGA: violation of constraint 2— case 1

We now consider case 2, in which only one aggregate is involved in the violation. In Figure 3.23, the CDG does not violate constraint 2. A violation of constraint 2 in the DGA is observed, as shown in 3.24. The solution to this kind of violation can be derived from the solutions in Category I and Category II. If the aggregated COMP block is of type AND, it is decomposed to multiple SEQ blocks. If the aggregated COMP block is of type XOR, it is decomposed to multiple new COMP blocks of type XOR.

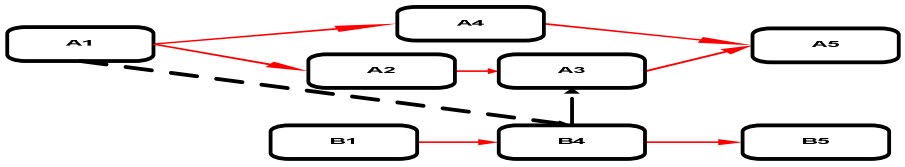

Figure 3.23: Example CDG: violation of constraint 2— case 2

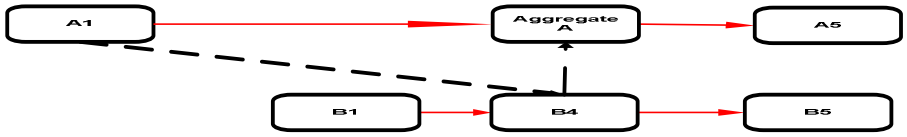

Figure 3.24: Example DGA: violation of constraint 2— case 2

A solution is shown in Figure 3.25 (the aggregated COMP block is of type XOR). Note, the new COMP block containing activity "A2" and "A3" must be executed after the one containing "A4", or it will still violate this constraint.

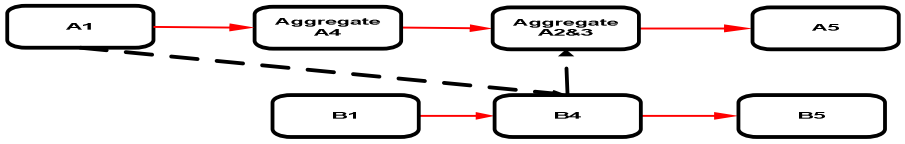

Figure 3.25: Solution of violation of constraint 2 — COMP of type XOR (case 2)

3.3 Compose Services in the Open-box Model into a Structured Composite Service

3.3.1 Example Scenario

In this sub-section we describe the running example for our approach.

A library frequently orders books from an online book trader. The online book trader ships the books by a standard carrier or by an express service (Preuner and Schrefl, 2005). The business process at the library side is described below (see Figure 3.27). Firstly, books are selected from some catalogues. Next, these selected books are registered in a system. And in parallel, an order is sent to the book trader—activity "Order processed". Next, the payment is processed. The book trader will give confirmation to the library when both the order and the payment are received. When both the books and the invoices have been received, the library will acknowledge receipt to the book trader. This is the last step of this process. The CDG from which the structured business process at the library can be derived is shown in Figure 3.26; the business process is shown in Figure 3.27. This process can be expressed as SEQ[SEQ[Select Books], COMP{SEQ[Registered], SEQ[Order Processed]}, SEQ[Payment Processed, Order Confirmed], COMP{SEQ[Book Received], SEQ[Invoice Received]}, SEQ[Acknowledge Receipt]].

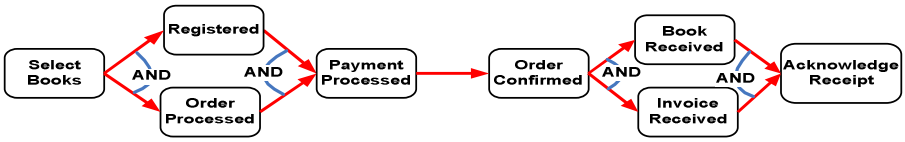

Figure 3.26: The CDG for the business process at the library

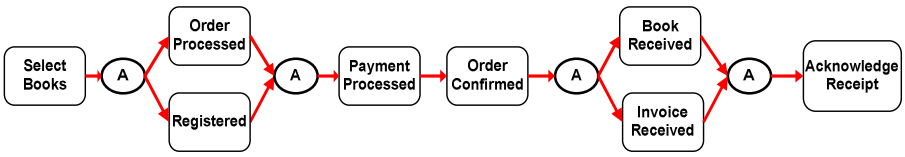

Figure 3.27: Business process at the library

The business process at the book trader is described below (see Figure 3.29). When an order is received, the book trader starts to prepare the shipment and the payment receiving. (So, it implies that the book trader always has the books ordered in stock). When the preparation has been finished, the book trader sends confirmation to the library—activity "Prepare Finished". The physical delivery of the books can be in two ways: the standard carrier and the express services. Which delivery method is chosen depends on some non-functional requirements; for example, the due date required by the library. When the delivery has been processed, an invoice is processed. Finally, an acknowledgement from the library is received. The CDG from which the structured business process at the book trader can be derived is shown in Figure 3.28; the business process is shown in Figure 3.29. The process can be expressed as SEQ[SEQ[Order Received], COMP{SEQ[Payment Received], SEQ[Shipment Prepared]}, SEQ[Prepare Finished], COMP{SEQ[Shipment Carrier], SEQ[Shipment Express]}, SEQ[Invoice Processed, Received Acknowledge From library]].

To attain a simple and illustrative process, we assume the standard carrier and the express service are offered by the book trader itself. It is straightforward, however, to extend the case to that the delivery services are offered by different companies.

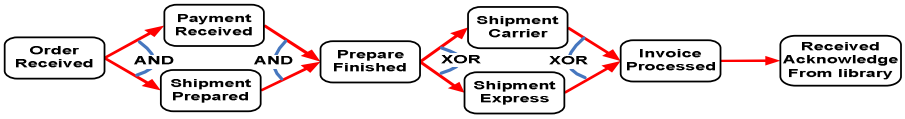

Figure 3.28: The CDG for the business process at the book trader

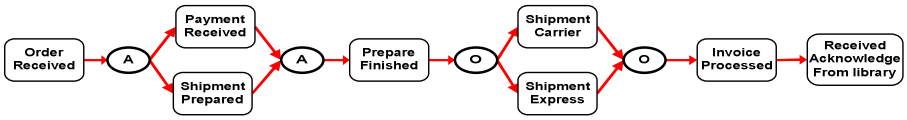

Figure 3.29: Business process at Book Trader

The inter-organizational dependencies are shown in Figure 3.30, which is a CDG including the above two services. Note, one dummy activity is insert in the book trader side after activity "Prepare Finished" to satisfy the requirement that "Only one type, either AND or XOR, can be assigned to incoming resp. outgoing dependencies" (Eshuis et al., 2006).

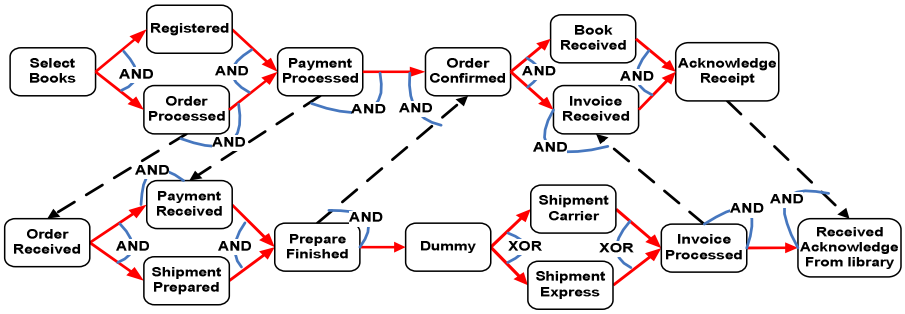

Figure 3.30: The CDG between library and book trader

3.3.2 Composition Procedure

Our composition procedure takes the CDGs (provider side and consumer side) from which the structured process models of services can be derived using the algorithm in (Eshuis et al., 2006), the structure process models of services (provider side and consumer side) and the inter-organizational dependencies between these services as inputs. We follow a semi-automatic approach. The output is a structured composite service. Our procedure starts from the constructing of the DGAs. In this section, we focus on the bilateral scenario. In next section we generate our approach to the multilateral scenario.

Our approach consists of four steps:

Step 1: Derive the DGA from the CDGs, structured processes and the inter-organizational dependencies.

For the detailed procedure of this step we refer to section 3.2. The properly constructed DGA complying with inputs from Figure 3.26 through Figure 3.30 is shown in Figure 3.31. To resolve the violation of constraint 2, two dummy tasks are added. This solution comes from (Eshuis et al., 2006). Because the violations are also observed in the CDG, our solutions introduced in section 3.2 are not applicable as explained previously.

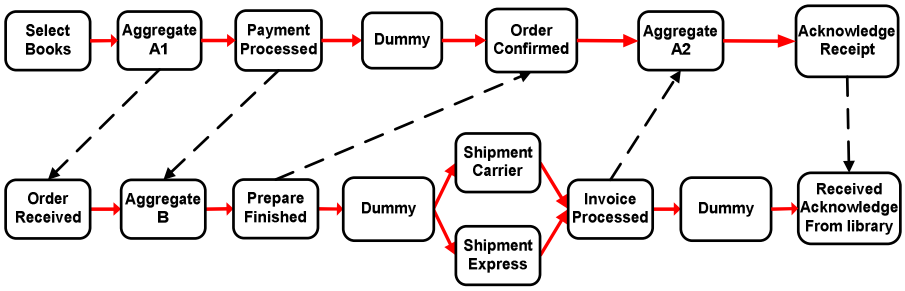

Figure 3.31: The DGA complying with inputs from running example

Step 2: Use the algorithm in (Eshuis et al., 2006) to construct a structured composite service.

Treating each activity and aggregate as black-box service, we can now directly employ the algorithm to construct a structured composite service which contains activities and aggregates. The

properly constructed structured composite service including aggregates is shown in Figure 3.32. C in a circle indicates a composite construct, either of type AND or type XOR.

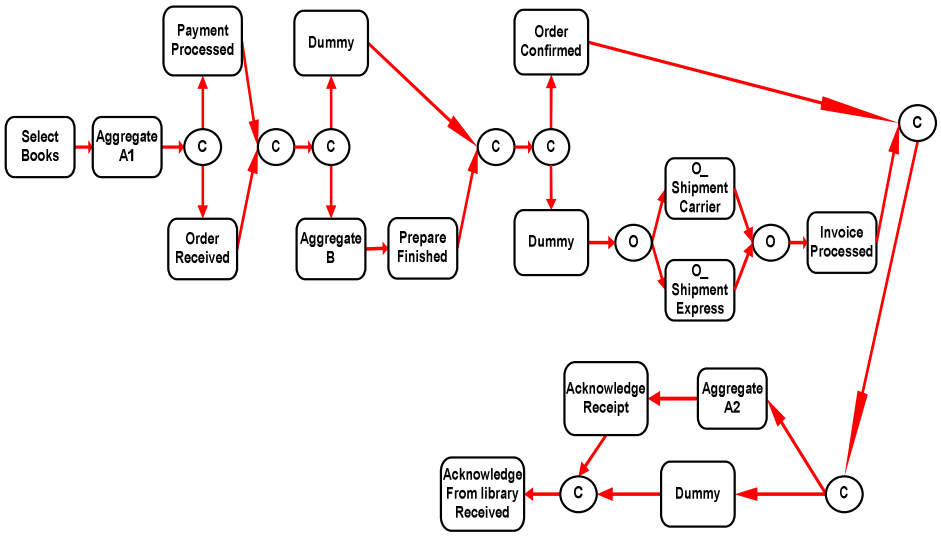

Figure 3.32: Structured composite service with aggregates

Step 3: Restore the COMP blocks, getting the most detailed structured composite service.

In this step, the aggregates are replaced, e.g., the original COMP blocks are used to replace the aggregates in the composite service in the above step. A new composite service, which complies with inputs from the running example, is shown in Figure 3.33.

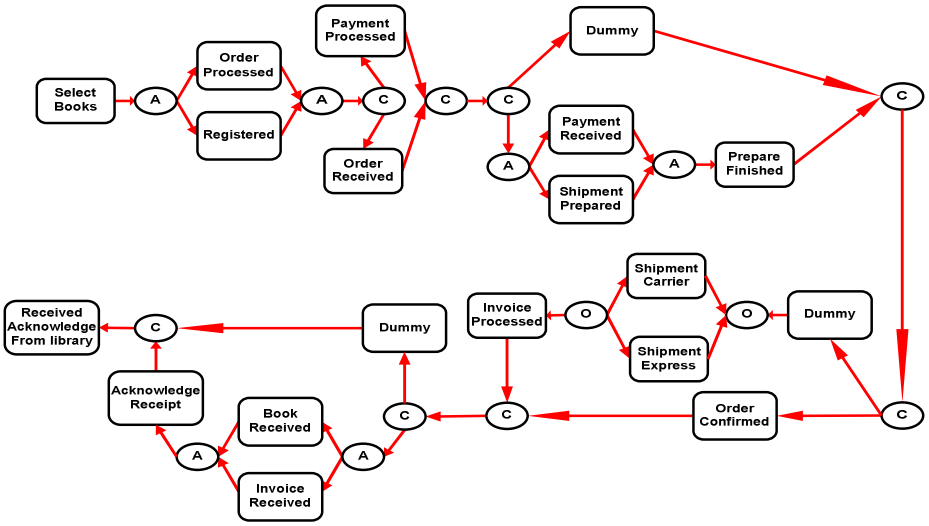

Figure 3.33: The un-aggregated structured composite service complying with inputs from the running example

Step 4: Manually type the branches using the AND or the XOR constructs.

In this step, the branches of the most detailed composite service are manually typed in accordance with the CDG. The result is shown in Figure 3.34.

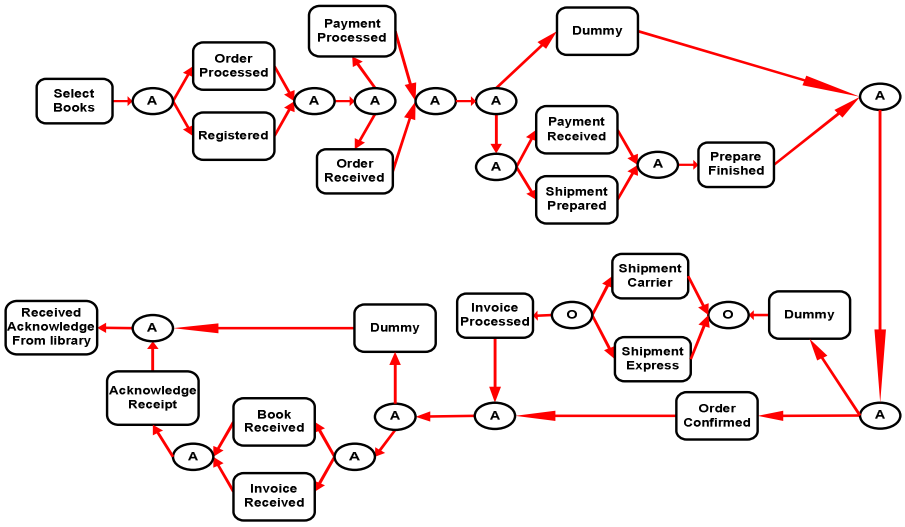

Figure 3.34: Structured composition complying with inputs from the running example

3.4 Generalization to Multiple Participants

In the previous section, we discussed our composition procedure for the bilateral scenario. In this section, we generalize the procedure to the multilateral scenario.

3.4.1 Negative Answer to a Stepwise Approach

In this sub-section, we argue why we can not iteratively use the bilateral approach in the multilateral scenario. In other words, we can not compose the providers' processes one by one to the consumer's in the multilateral scenario.

We now consider three artificial services. Their processes at the external level are shown in Figure 3.35. And the CDG is shown in Figure 3.36. The desired composite service is shown in Figure 3.37. In the desired composite service, activity "A2" is done by service A itself; service A also needs to outsource part of this process to one of its two service providers. Only one of the two providers is chosen. Following the bilateral approach, we first compose service A and B. A new composite service is shown in Figure 3.38.

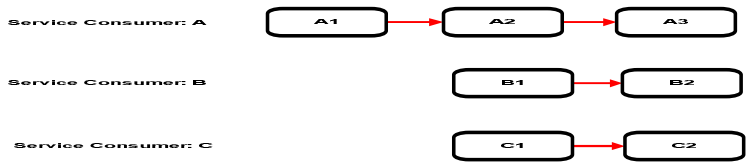

Figure 3.35: Business process for service A, B and C

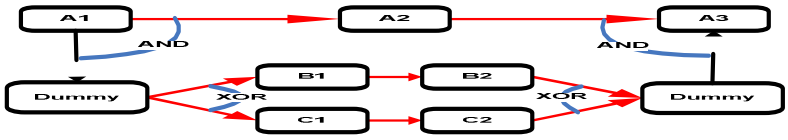

Figure 3.36: The CDG for service A, B and C

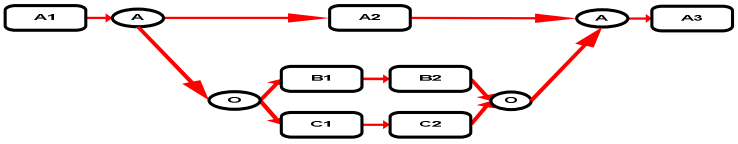

Figure 3.37: The desired composite service for service A, B and C

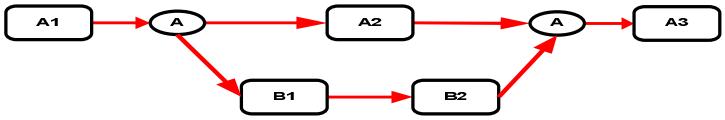

Figure 3.38: Composite service for service A and B

After composing services A and B, in this second step, we need to compose service C to the composite service of A and B. The resulting composite service is shown in Figure 3.39. In Figure 3.39, however, we can not properly type this composite service. If we type “C01” with XOR, activity “A2” can be by passed; if we type “C01” with AND, both service B and C are executed. Neither of these two typings can generate our desired composite service. For this reason, we can not use the bilateral approach iteratively in the multilateral scenario to guarantee a desired composite service.

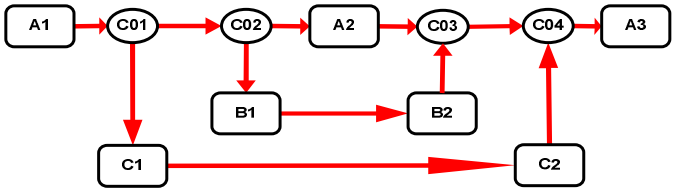

Figure 3.39: Composite service for service A, B and C (Stepwise approach)

3.4.2 Generalization of the Composition procedure to Multilateral Scenario

Our approach is applicable in the multilateral scenario. We can still follow the four-step framework. In step 1, the DGAs must now include all the services which we want to compose in the multilateral scenario. The last three steps are exactly the same in the multilateral scenario as in the bilateral scenario.

Still using the example shown in Figure 3.35 and 3.36, we briefly showcase how our approach can be applicable in the multi-lateral scenario. In step 1, the CDG shown in Figure 3.36 is used as inputs among others. Note, the CDG shown in Figure 3.36 contains all three services which we want to compose. Step 2 through step 4 is just the same as in the bi-lateral scenario. Finally, we get our desired structured composite service shown in Figure 3.37.

Chapter 4

ARCHITECTURE & ITS CONFIGURATION AND SYNCHRONIZATION PROCEDURE

In this chapter, we propose an automatic procedure to synchronize standard BPEL processes in the bilateral scenario. With respect to the multilateral scenario, we refer to the work proposed in the project Crosswork (Grefen et al. 2007; Crosswork, 2007).

To fully support our approach, we clearly distinguish the architecture for the global process of the composite service (global process, in short) from the process at the local service side (local process, in short); and new dedicated architectures for the global process and for the local process are developed. The architecture for the local process applies to both the service provider and the service consumer. These new architectures are inspired by Grefen et al. (2006a and 2007). Two rules are defined. These rules are used to decide on what activities at the local processes need to be controlled by the global process in the bilateral scenario.

In the synchronization procedure, a control activity is an activity at the local processes the start of which the global process needs to control (Grefen et al., 2006a). All the control activities are put on a list of control activities (LCA) as explained later on. The execution of the local process is automatic; the execution of the activities not on the "LCA" is controlled by the local process itself. Note, the local process refers to either the service provider or the service consumer in the bilateral scenario.

4.1 Architecture Supporting Our Approach

To fully support our approach, we developed new dedicated architecture for the local process respectively for the global process.

The architecture for the local processes is shown in Figure 4.1. The document "BPEL SPEC (L)" is the specification for the local processes at the external level at either the provider side or the consumer side; for example, the specification shown in Figure 3.2 and Figure 3.4 in last Chapter. This specification is created by the local process using some process modeling language (in our case, it is BPEL). A composer (normally, the consumer) can obtain this specification by the interface "SPEC". This specification is used to construct the composite services. Interface "MON" and "CTRL(L)" are used to monitor and control the execution of the local process from the global process to the local process. Control can be at the process level and/or the activity level. Two important operations at activity level are "lock activity" and "release activity". The "lock activity" informs the BP-WS at the local process not to start an activity until it is released through function "release activity". Together, these two functions provide an implementation of

control flow dependencies from the global process to the local process. Interface "ACT" is used to activate the local process and/or an instance at the local process (Grefen et al., 2006a).

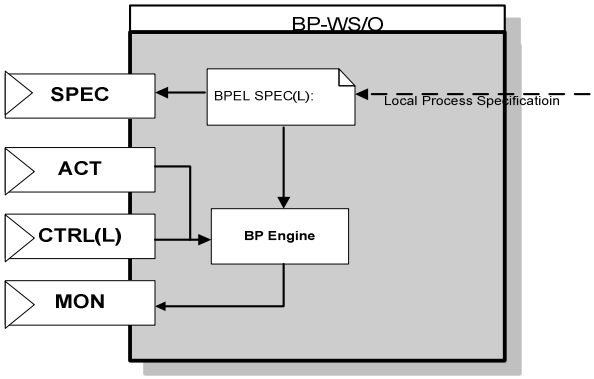

Figure 4.1: BP-WS/O architecture dedicated to the local process

The architecture for the global process side is shown in Figure 4.2.

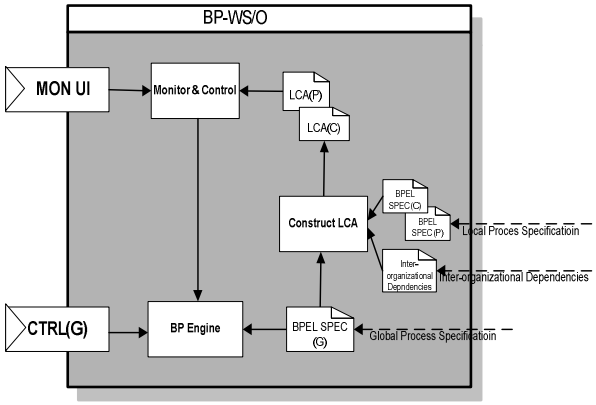

Figure 4.2: BP-WS/O architecture dedicated to the global process

Compared to the architecture shown in Figure 4.1, we do not need interface "SPEC" and "MON", because the composite service does not need to expose the global process to the local process; and the local process does not need to monitor the execution of the global process. We also do not need the interface "ACT", because the local process should not be able to activate the global process. The document "BPEL SPEC (G)" is the specification for the global process of the composite service; for instance, the composite service shown in Figure 3.34 in Chapter 3. This specification is constructed by a component "Process Composition" (not shown in our architecture). Our work in the previous chapter presents an approach which can be implemented in the component "Process Composition".

To support our approach, however, we must provide a BP-WS at the global process with the information required for control flow dependencies from the global process to the local process. The "LCA" is constructed by a component "Construct LCA". To construct the "LCA", the component "Construct LCA" needs two inputs: the local process specification (both the provider side and the consumer side), and

the inter-organizational dependencies. The local process specification, such as that in Figure 3.2 and Figure 3.4 in Chapter 3, can be attained through the interface "SPEC" in the local side; and stored in the documents "BPEL SPEC (C)/(P). The inter-organizational dependencies are store in the document "Inter-organizational Dependencies". This document is input from the other component not shown in our architecture. The "LCA" is stored in the document "LCA". In the next section, we use a sequence diagram to illustrate how the BP-WS at the global process uses the "LCA" to control the activities at the local process. Note, there are two "LCA"s in the bilateral scenario. "LCA(P)" is for the provider side; "LCA(C) is for the consumer side.

The interface "CTRL(G)" is used to facilitate the synchronization from the local process to the global process. Two important operations defined in this interface are "start activity" and "complete activity". Operation "start activity" informs the BP-WS at the global process that an activity at the local process has been started. Operation "complete activity" informs the BP-WS at the global process that an activity at the local process has been completed. Together, these two operations provide an implementation of control flow dependencies from the local processes to the global process. A user uses interface "MON UI" to monitor and control the execution of the global process during the enactment time.

The purpose of the component "Monitor & Control" is two-fold. Firstly, this component is responsible for communicating with all specification and control ports of all local and global services to be aware of the status of execution of the composite service. Secondly, this module controls the start of the execution the process, starting the process and/or an instance at the global process.

4.2 Construct the List of Control Activities (LCA)

Component "Construct LCA" uses Rule 4.1 and Rule 4.2 to construct the "LCA(P)/LCA(C)". The LCA(P)/LCA(C) are decided by means of static analysis. This static analysis can be done off-line. We use rule 4.1 and rule 4.2 to construct the LCA(P) respectively the LCA(C).

Rule 4.1: If an activity *y* at the service provider side depends on an activity (or activities) at the service consumer side with respect to control flow dependency, this activity *y* at the provider side needs to be put on the LCA(P); if this activity *y* is a descendant of a COMP block, all the descendants of the most abstract COMP block of *y — mac ({y}) —* are put on the LCA(P).

For instance, from Figure 3.30 in Chapter 3, we know activity "Payment Received" depends on an activity at the service consumer side, and it is a descendant of a COMP block. The descendants of *mac ({"Payment Received"})* includes two activities, "Payment Received" and "Shipment Prepared"; all these two activities are put on the LCA(P).

Rule 4.2: If an activity *x* at the service consumer side depends on an activity (or activities) at the service provider side with respect to control flow dependency, this activity *x* at the consumer side needs to be put on the LCA(C); if this activity *x* is a descendant of a COMP block, all the descendants of the most abstract COMP block of *x — mac ({x})* are put on the LCA(C).

For instance, from Figure 3.30, we know activity "Invoice Received" depends on an activity at the service provider side, and it is a descendant of a COMP block. The descendants of *mac ({"Invoice Received"})* includes two activities, "Invoice Received" and "Book Received"; all these two activities are put on the LCA(P).

Rule 4.2 is similar to rule 4.1; however, to clear distinguish the LCA(C) from the LCA(P), we stated it separately.

In the above two rules, even though two activities involved in an inter-organizational dependency, only the ending activity is controlled, but not the starting activity. For instance, in Figure 3.30, the starting activity “Order Processed” is not controlled by the global process; only the ending activity “Order Received” is controlled. This is because that one activity needs to be controlled by the global process, if and only if the execution of this activity is constrained by the other service(s). One starting activity does not need inputs from the other service, all its inputs can be satisfied within the local process, so it does not need to be controlled by the global process.

4.3 Implementation of Control Flow Dependency

4.3.1 Control Flow from the Global process to the Local process

In this sub-section, we intend to solve the problem how an activity at the local process can be controlled by the global process. We use a sequence diagram to illustrate the interaction between the BP-WS — global process and the BP-WS — local process. The local process refers to either service consumer or service provider in the bilateral scenario; however, for clear reference, we take the service provider as an example in the description below.

In Figure 4.3, the sequence diagram illustrates the synchronization procedure from the global process to the provider side, described below.

In step 1, a user invokes the component “Monitor & Control” through “MON UI”. In step 2, the component “Monitor & Control” sends feedback to “MON UI”. In step 3, the user activate a composite service. In step 4, the component “Monitor & Control” activates the global process for the composite service at the global process BP-Engine. In step 5, when one global process is activated, the global process BP-Engine sends a message to the provider side. This message informs the provider to activate the local process; this is done through the interface “ACT”, operation “activate (local process)” at the provider side. Step 6 through step 8 gives feedback. In step 9, the user starts an instance. In step 10, the component “Monitor & Control” activates an instance at the global process BP-Engine. In step 11, the global process BP-Engine activates an instance at the provider side; this is done through the interface “ACT”, operation “activate (instance)”. In step 12, the component “Monitor & Control” informs the global process BP-Engine to lock the activities on the LCA(P). In step 13, the global process BP-Engine sends a message to the provider side BP-Engine to lock the activities on the LCA(P); this is done through the interface “CTRL(L)”, operation “lock activity” at the provider side. Step 14 and step 15 gives feedback. In step 16, the component “Monitor & Control” informs the global process BP-Engine to release one activity on the LCA(P). In step 17, the global process BP-Engine sends a message to the provider side BP-Engine to release the activity on the “LCA(P). This is done through the interface “CTRL(L)”, operation “release activity” at the provider side. Step 18 and step 19 are feedbacks of the release success. Step 16 through step 19 can repeat multiple times, if there are multiple activities on the “LCA(p). After arbitrary execution time, we assume the instance has been successfully executed. Step 20 through step 21 give feedback of successful instance execution.

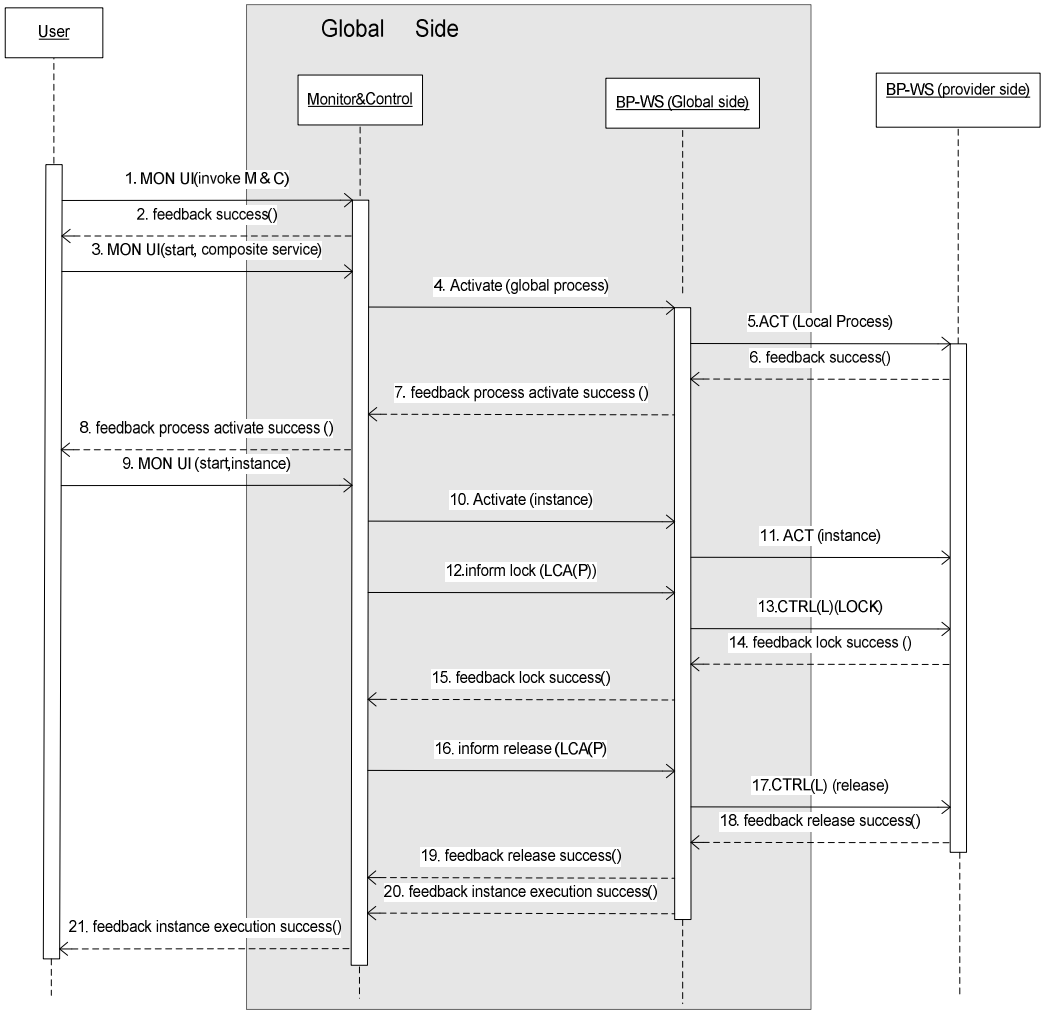

Figure 4.3: Sequence diagram from the global process to the local process

The component "Monitor & Control" uses rule 4.3 to decide on when the activities on the LCA(P)/LCA(C) to release.

Rule 4.3: If all the inputs of an activity *y* on the LCA(P)/LCA(C) have been satisfied, this activity *y* needs to be released.

Here, we treat the global process as a normal workflow. All the inputs of an activity have been satisfied means that one activity can be fired. We use Figure 3.8 in Chapter 3 to illustrate our idea. In Figure 3.8, activity "A2" needs to be controlled (known from Figure 3.5). When activity "A1" and activity "B1" have been completed (noticed respectively from service A and service B to the global process), activity "A2" is released.

The above presented procedure is only applicable if there are no loops in the local process and the global process. To see why, consider the segment of a process in Figure 4.4. We assume activity "A4" is on the LCA(P) or the LCA(C). Every time, the start of this activity should be controlled by the global process. However, according to our procedure, the release of this activity is permanent. After the first iteration, the

start of activity “A4” is totally controlled by the local side. How to extend our work to deal with processes with loops is suggested for future work.

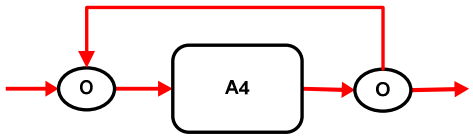

Figure 4.4: A process segment with a loop

There is another potential problem with the above presented procedure. When an instance is activated at the local process, the first activity of the local process is executed immediately. This implies if the first activity is on the LCA, it would be out of control from the global process. To solve this problem, one activity “activities on the LCA locked” can be added before the first activity of the original process at the provider side and the consumer side. This activity can be completed if and only if the activities on the LCA have been locked.

4.3.2 Control Flow from Local Service to Composite Service

In this sub-section, we intend to solve the problem how an activity at the local process can be synchronized to the global process. We use a sequence diagram to illustrate the interaction between the BP-WS — global process and the BP-WS — local process. The local process refers to either the service consumer or the service provider in the bilateral scenario; however, for clear reference, we take the service provider as an example in the description below.

In Figure 4.5, the sequence diagram illustrates the synchronization procedure from the provider side to the global process, described below.

Step 1 through step 11 are the same in this sequence diagram with that from global process to local process. Step 12 and step 13 are feedbacks. In step 14, when an activity at the provider side is started, the provider side sends a message to inform the global process. This is done through the interface CTRL(G), operation “start activity”. In step 15, when an activity at the provider side is completed, the provider side sends a message to inform the global process. This is done through the interface CTRL(G), operation “complete activity”. Step 14 and step 15 can repeat multiple times, since there are multiple activities at the local process. Step 16 and step 17 are feedbacks. Note, the start and the completion of every activity at the local process needs to be sent to the global process. In this way, the global process can reflect the latest status of the local processes.

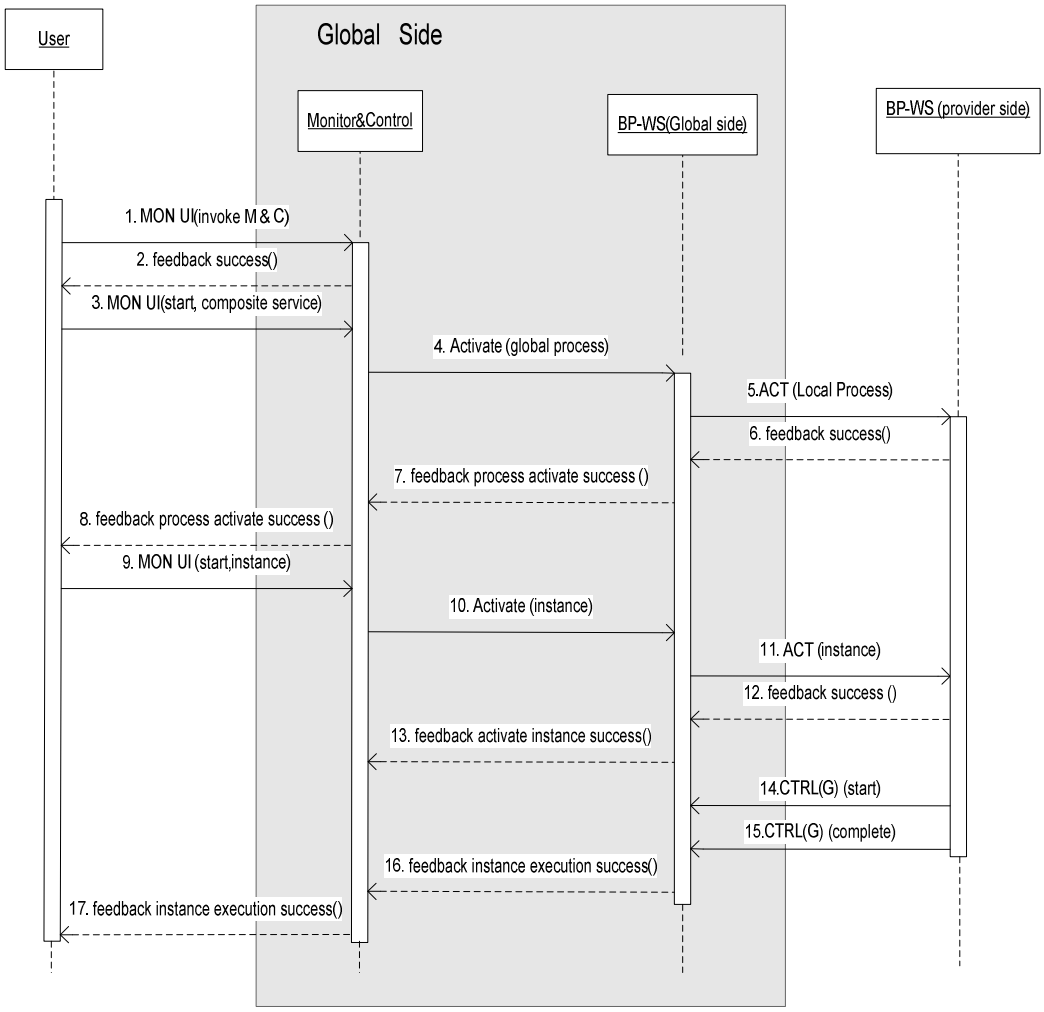

Figure 4.5: Sequence diagram from the local process to the global process

4.4 Architecture Supporting Synchronization in the Multilateral Scenario

For the multi-lateral scenario, the architecture proposed by Grefen et al. (2007) is a good starting point. In (Grefen et al., 2007), two levels of workflow enactment, at the global process and at the local process, are clearly identified in the enactment architecture. Next to this, a monitoring user interface module is employed. A “Monitor & Control” engine is used to monitor and to control the global enactment. For the real-time enactment, at either the global process or the local process, a standard BP-Engine is used. BP-WS interfaces are used to access to specifications and states of the processes, at local process and/or at global process.

Chapter 5

CASE STUDY

In this chapter, the complete approach for composing services in the open-box model is put to the test by means of case studies. In the bilateral scenario, we use a case from healthcare domain; in the multilateral scenario, we use a case from the car insurance domain.

5.1 Case Study in the Bilateral Scenario

In this case study, one hospital (consumer) outsources parts of its scan interpretation process to a consulting company (provider) which has the competency in scan interpretation. The goal of the composer from the hospital is to compose the service from the consulting company to that from the hospital into a structured composite service. Note, both the services from the consulting company and the hospital are expressed as structured business processes.

The business process at the hospital side is shown in Figure 5.1. The process starts with receiving a scan order. Next, the scan order is scheduled. On the scheduled date, one out of the three scans (CT, MRI and X-Ray) is operated. The scan manuscripts need to be processed and stored in a database which can be accessed by the authorized units. Next, the reports are created by some radiologist. When the reports are ready, they are distributed to the relevant doctors. The doctors must sign off the reports. Next, the reports are analyzed by doctors; and at the same time, the scan is billed. Finally, the reports are archived. In Figure 5.1, the inputs from / outputs to the environment (consulting company) are shown as dotted arrow.

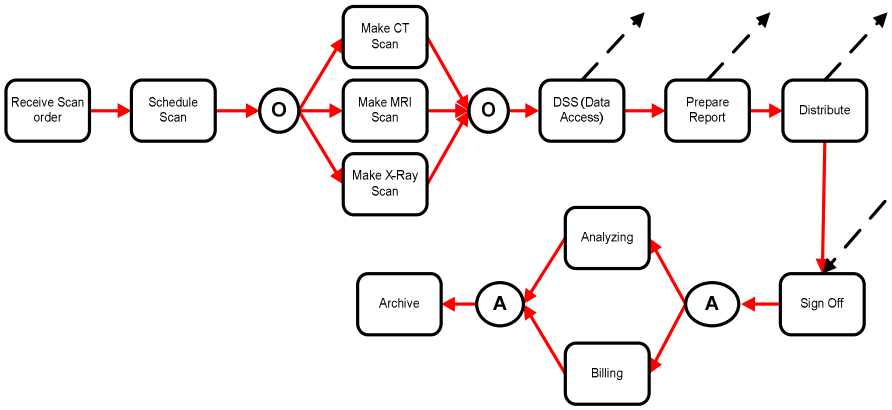

Figure 5.1: Business process at the hospital side

In the above process, two activities — "Prepare Report" and "Distribute" are outsourced to one consulting company. The business process at the consulting company is shown in Figure 5.2. The process starts with the receiving of the scan manuscripts from hospital. Next, the interpretation is scheduled. On the scheduled date, the manuscripts are analyzed, creating the reports. After the creating of the reports, a native speaker from the hospital verifies the reports in terms of language; a second expert within the consulting company guarantees that the report is correct. Finally, the checked reports are sent to the relevant doctors in the hospital. Activity "delivery" needs input from hospital; for instance, to whom the reports are to be sent.

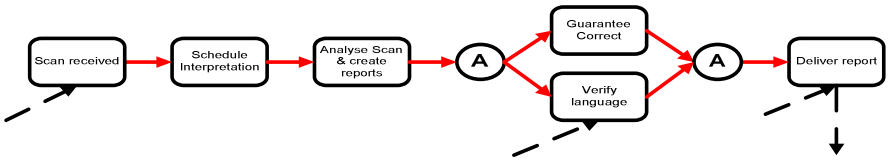

Figure 5.2: Business process at the consulting company

To employ our approach, the DGA is firstly constructed. The DGA is shown in Figure 5.3. From Figure 5.3, we can see that the execution progress of both parties depends on one another and the execution autonomy of both parties is reduced. Note, we insert one dummy activity between activity "Distribute" and activity "Sign off" to solve the violation of the constraint 2; and the COMP block containing activity "Verify language" in the consulting side is aggregated into "aggregate A", because the "Verify language" is involved in an inter-organizational dependency. Using the algorithm in (Eshuis et al., 2006), the above two services are composed; and in accordance with the concrete CDG, the composite service is properly typed. The composite service is shown in Figure 5.4. This composite service is now ready to be fed into a process engine.

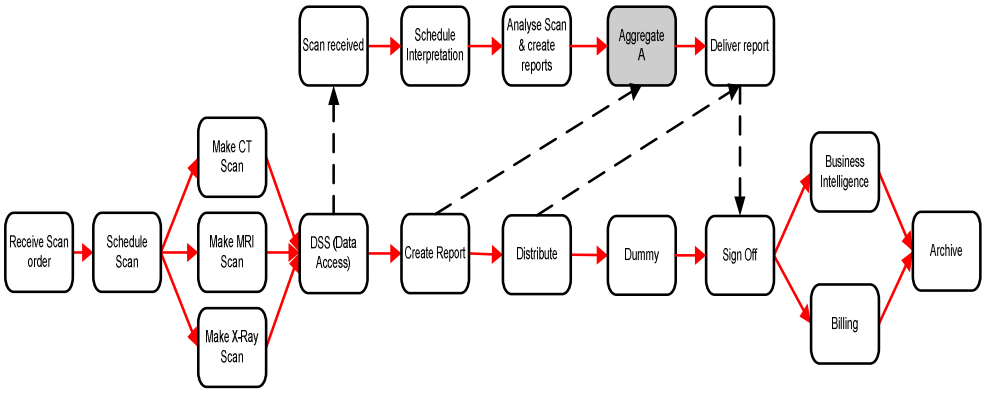

Figure 5.3: The DGA between hospital and consulting company

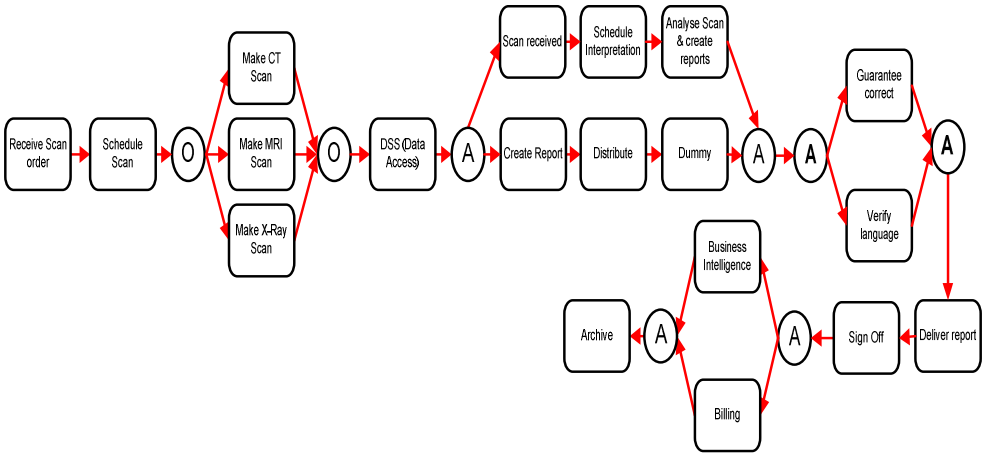

Figure 5.4: The composite service complying with Figure 5.3

5.2 Case Study in the Multilateral Scenario

In the multilateral scenario, the complete approach is explained by means of a case study from the project Cross flow (Browne and Kellett, 1999). The goal of CrossFlow is to link the process within the insurance company (AGFIL, AGF Irish Life Holding Plc—an insurance company operating in the Irish market; the service consumer.) with the processes from its partners. The partners (service providers) include Europ Assist, Lee Consulting Services and Garages. The relationship between AGFIL and its partners in this case is based on the operation of a scheme known as the Emergency Service which is available to private motor policyholders.

The responsibilities of each actor are described next. AGFIL is responsible for underwriting the motor policy and covering losses incurred. Europ Assist records the initial advice over the telephone. Lee Consulting Services co-ordinates and manages the operation of the Emergency Service on a day to day level on behalf of AGFIL. Garages are responsible for offering the approved repair service and the courtesy cars (if required).

A new composite service should be designed to encompass all the processes from all the parties. The composite service will offer AGFIL a process which will have control over all the activities. A multi-step structured process will be desired where an outside agent will start the work flow process.

The process from AGFIL is shown in Figure 5.5. After receiving information about claims, the AGFIL notifies Lee immediately. Next, the claim forms are obtained from the policyholders. The claims must be checked. Next, the estimations of the damages are amended in accordance with the information from Lee. Next, the information must be reconciled after Lee gets the invoices. Last step of this process is the claim finalization.

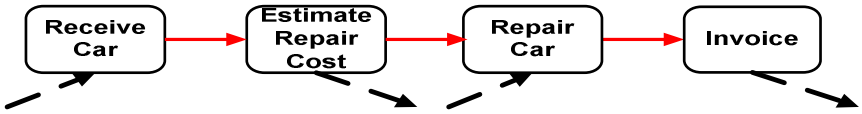

Figure 5.5: Business process from AGFIL

The process from Europ Assist is shown in Figure 5.6. The process starts with receiving calls from the policyholders. The information about the policyholder is collected in this step. This information is validated before any further processes. Next, the damaged car is assigned to a garage; at the same time, AGFIL is notified. The last step is the "archive" of the file.

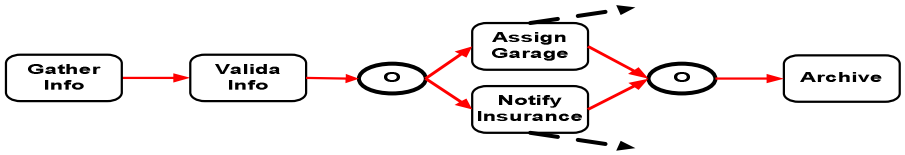

Figure 5.6: Business process from Europ Assist

The process from Lee is shown in Figure 5.7. The process starts from obtaining details about the claim. Next, Lee will contact garage to which a damaged car has been sent by the Europ Assist. If the estimated cost is more than 500 euro, an adjustor will be assigned to inspect the damaged car; if not the repair is immediately agreed. The last step is the checking of the invoice.

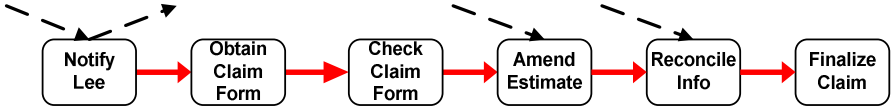

Figure 5.7: Business process from Lee

The process from Garage is shown in Figure 5.8. The process at the garage side starts with the receiving of the damaged car. Next, the repair cost is estimated. After getting feedback from Lee, the car is repaired. The last step of this process is sending invoice to Lee

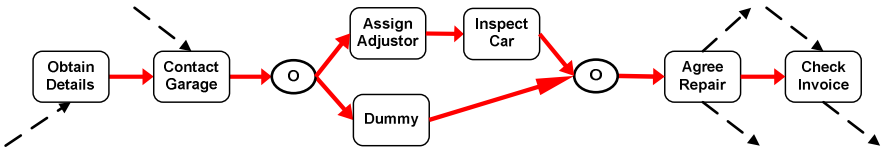

Figure 5.8: Business process from Garage

. To employ our approach to compose all these processes together into a structured composite service, the DGA needs to be constructed, as shown in 5.9.

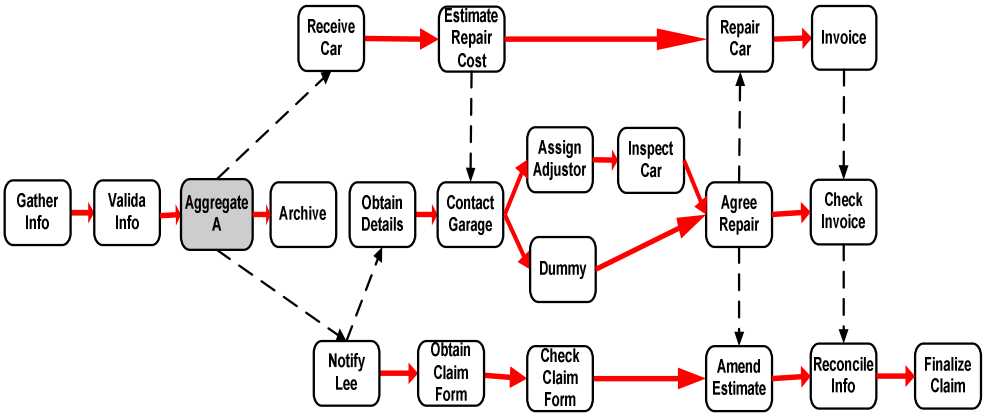

Figure 5.9: The DGA including all services

In the composition procedure, one COMP block is aggregated. “aggregate A” contains activity “Notify insurance” and “Assign Garage” in Europ Assist. Using the algorithm in (Eshuis et al., 2006), a structured composite service is constructed; and in accordance with the concrete CDG, the composite service is properly typed. The composite service is shown in Figure 5.10. This composite service is now ready to be fed into a process engine.

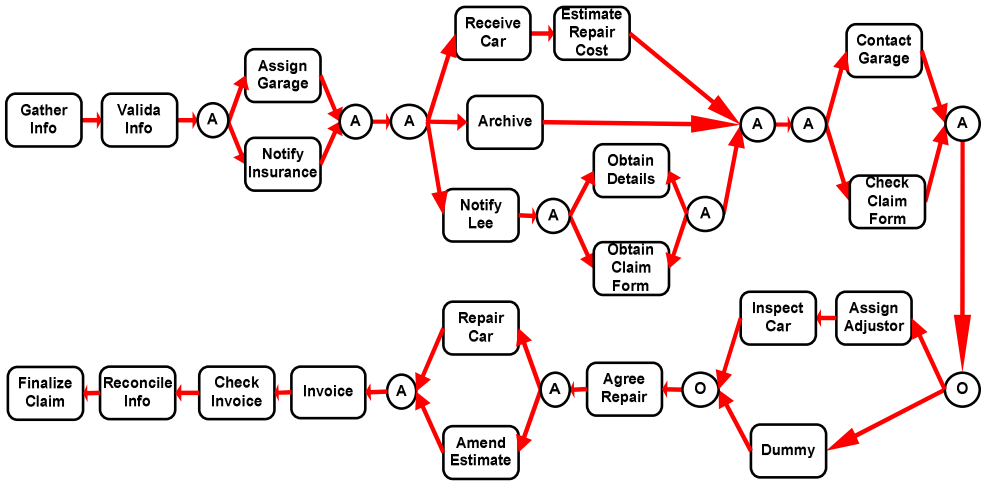

Figure 5.10: The composite service complying with Figure 5. 9

5.3 Conclusion of Case Studies

In this chapter, the full approach has been tested, and has successfully constructed the desired composited services in two cases. From these case studies we can observe that the composition procedures are almost the same in the bilateral scenario and in the multilateral scenario. In the multilateral scenario, the CDGs must include all the services which we want to compose; however, in the bilateral scenario, only two services are included. These case studies have confirmed our approach is applicable.

Chapter 6

CONCLUSION

Throughout this graduation project, the emphasis of our research has been on the analysis of service composition. In chapter 1 of this document, three research questions were defined. During our research we have tried to answer these questions.

The first research question was defined as: “How can services expressed as explicitly visible business processes be composed together? ”. In order to answer this question, we have proposed a Business Process Web Services (BP-WS, in short) based approach. BP-WS is an extension of the basic web service, and has an internal business process specification that can be accessed externally. We have shown that services expressed as structured business processes can be composed together, generating a new structured composite service. We proposed a semi-automatic procedure for service composition, applicable in both the bilateral and the multilateral scenario.

The semi-automatic procedure reused the approach proposed by Eshuis et al. (2006). Our semi-automatic procedure is also based on the dependency graphs; and the algorithm in (Eshuis et al., 2006) is reused to construct the structured composite services. The most interesting part of this paper is the argument that we must treat some COMP blocks as groups in the composition procedure. The reason why we need to treat some COMP blocks as groups is that the inter-organizational dependencies put direct/indirect influences on all SEQ blocks in the COMP blocks. One of the other contributions of this paper is the introduction of the concept “the most abstract COMP block (*mac*)”. The most abstract COMP block is used to aggregate the COMP blocks in the composition procedure; and the most abstract COMP blocks are treated as groups in the composition procedure. With this most abstract COMP block, the indirect influence generated by the inter-organizational dependencies on some SEQ blocks within this COMP block are properly treated in the composition procedure. The composition procedure is applicable in both the bilateral scenario and the multi-lateral scenario. In the multi-lateral scenario, we have shown that a stepwise approach is not applicable; and the dependency graph containing aggregates in the multi-lateral scenario must include all the services to be composed.

To compose services, two approaches exist in the design of a composite service: Top-down design and bottom-up. Top-down design means the overall process across all participating organizations is defined first, then the global process is fragmented and the fragments are delegated to different organizations (Aalst and Weske, 2001). Bottom-up design means several organizations define and advertise services, which are composed by other organizations (probably the service requester) (Eshuis et al., 2006). We follow the bottom-up design. The bottom-up approach gives individual organizations and users most flexibility, as they can choose to use their personal styles of working and describing their ‘local’ process descriptions (Eshuis et al. 2006; Preuner and Schrefl, 2005).

The second research question was defined as: “How can the global process be synchronized with the local processes?”. To solve this problem, we proposed an automatic synchronization procedure between the global process and the local process in the bilateral scenario. In our synchronization procedure, the control activity is an activity at the local processes the start of which the global process needs to control (Grefen et al., 2006a). The local process refers to either the service provider side or the service consumer side. The execution of the local process is automatic; the execution of the activities not on the “LCA” is controlled by the local side itself.

The key part of this automatic procedure is constructing the list of control activities (LCA); and the component “Monitor & Control” controls when the activities on the LCA can start by means of message exchange from global process to the local process. Our synchronization approach relies on the dedicated architectures at the global process side respectively the local process side.

The third research question was defined as: “What architecture can support our approach?”. We developed new dedicated architectures for the global process respectively for the local process to fully support our approach. This part of work is inspired by Grefen et al. (2006a and 2007). BP-WS interfaces were reused. Component “Monitor & Control” and Component “Construct LCA” were introduced. The “LCA” is stored in the document “LCA” as an extension to the global process specification.

We next present our reflections on this project.

In this paper, our research exclusively focuses on control flow aspects. The underlying foundations for our approach are BP-WS, the open-box model and the architecture supporting the open-box model (Grefen et al., 2006a). Our composition procedure is not limited to a dedicated process modeling language; this ensures the general applicability of our approach. However, the inputs for the composition procedure are limited to structured process models; these structured processes should not contain any loops. Structured processes facilitate the aggregating step, because the COMP blocks are clearly defined in the structured process models. Loops will violate constraints of the algorithm in (Eshuis et al., 2006), and will cause some potential problem in the synchronization procedure as explained previously.

For our composition procedure, we did not provide a formal proof of correctness. However, when we compose the composite service containing aggregates, we always use a constructed structured composition to change to another structured composition; the most abstract COMP blocks which are used to replace the aggregates are still structured. Even though, we did not develop a prototype to validate the complete approach, we have provided implementable rules for all key parts to facilitate implementation.

The research question on the synchronization procedure and the supporting architecture is more complex than expected. The synchronization procedure in the multi-lateral may encounter some new problems, for instance, how to organize the communication channels. We limited our work to the bilateral scenario, because of the limitation of time and space.

We next suggest some future work. In terms of composition procedure, one possible research problem is how to extend our work to deal with process models including loops. In terms of synchronization procedure, one possible research question is how to extend some of our results for the multi-lateral scenario. A new procedure may be needed in the multi-lateral scenario.

END

BIBLIOGRAPHY

Aalst et al. 2003. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros. "Process Patterns". *Distributed and Parallel Databases* 2003;14(3); p. 5-51.

Aalst and Weske, 2001. W.M.P van der Aalst and M. Weske. "The P2P Approach to Interorganizational Processes". K.R. Dittrich, A. Geppert, M.C. Norrie (Eds.): Proc. CAiSE 2001; LNCS 2068; p. 140-156.

Aalst, 1999. W. van der Aalst. "Interorganizational Workflows: an Approach Based on Message Sequence Charts and Petri Nets". *Systems Analysis – Modeling – Simulation* 1999; 34(3); p. 335-367.

Berardi et al. 2005. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. "Automatic service composition based on behavioral descriptions". *International Journal of Cooperative Information Systems* 2005; 14(4); P. 333-376

Business link, 2007. Business link web site: http://www.businesslink.gov.uk/bdotg/action/; accessed 2007.

Eshuis and Grefen, 2007. R. Eshuis and P. Grefen. "Constructing Customized Process Views". BETA Working Paper Series WP 197, 2007; Eindhoven University of Technology.

Eshuis and Grefen, 2007a. R. Eshuis, P. Grefen. "Structural Matching of BPEL Processes". BETA Working Paper, 2007; Eindhoven University of Technology.

Eshuis et al., 2006. R. Eshuis, P. Grefen, S. Till; "Structured Service Composition". Proc. *International Conference on Business Process Management* 2006; Lecture Notes in Computer Science 4102; p. 97-112.

Grefen et al. 2007. P. Grefen, N. Mehandjiev, G. Kouvas, G. Weichhart, and R. Eshuis. "Dynamic Business Network Process Management in Instant Virtual Enterprises". BETA Working Paper Series WP 198, 2007; Eindhoven University of Technology.

Grefen et al. 2006a. P. Grefen, H. Ludwig, A. Dan, S. Angelov. "An Analysis of Web services Support for Dynamic Business Process Outsourcing". *Information and Software Technology* 2006; 48(11); p. 1115-1134.

Grefen, 2006. P. Grefen. "Service-oriented support for dynamic inter-organizational business process management." In D.Georgakopoulos and M. Papazoglou, editors, Service Oriented Computing, 2006.

Grefen et al. 2003. P. Grefen, H. Ludwig, S. Angelov. "A Three-Level Framework for Process and Data Management of Complex E-Services" *Journal of Cooperative Information Systems* 2003;12(4); p. 487-531.

Khalaf and Leymann, 2006. R. Khalaf, F. Leymann. "E Role-based Decomposition of Business Processes Using BPEL". *IEEE International Conference on Web services* (ICWS' 06).

Kiepuszewski et al. 2000. B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. "On structured process modeling". In B. Wangler and L. Bergman, editors, Proc. CAiSE' 2000; p. 431-445.

Liu and Kumar, 2005. R. Liu and A. Kumar. "An analysis and taxonomy of unstructured processes". In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, Proc. 3rd conference on Business Process Management (BPM 2005), Lecture Notes in Computer Science 2005; 3649, p. 268-284.

Manes, 2003. A.T. Manes. "*Web services – A manager's Guide*" 2003; ISBN 0-321-18577-3

Muth et al. 1998. P.Muth, D. Wodkte, J. Wiessenfels, D,A. Kotz, G. Weikum. "From Centralized Process Specification to Distributed Process Execution". *Journal of Intelligent Information Systems* 1998; 10(2); p.159-184

Norta, 2007. A. Norta. "Exploring Dynamic Inter-organizational Business Process Collaboration". Beta Research School for Operations Management and Logistics 2007. Eindhoven University of Technology.

Preuner and Schrefl, 2005. G. Preuner, M. Schrefl. "Requester-centered composition of business processes from internal and external services". *Data & Knowledge Engineering* 2005; 52(1); p. 121-155.

Browne and Kellett, 1999. Sinead Browne & Michael Kellett (AGFIL), "cross-organisational, process CrossFlow" (ESPRIT E/28635) http://www.CrossFlow.org/public/pubdel/D1b.pdf,

Zhao and Cheng, 2005. J.L. Zhao, H.K. Cheng. "Web services and process management: a union of convenience or a new area of research?" *Decision Support System* 2005;40; p.1-8.

APPENDIX A: THREE-LEVEL PROCESS FRAMEWORK

The enactment of the service composition in the agile contexts implies a couple of steps: services have to be identified and defined, business network partners have to be found, process enactment and data management infrastructures have to be set up and coupled. Given the fact that the above mentioned steps are quite diverse in nature, an individual process specification will not be adequate for all these steps (Grefen et al. 2003). Hence, we need a multi-level process framework to facilitate collaboration among the involved organizations. Such kind of process specifications must not only focus on the local process implementation, but also have to facilitate the inter-organizational collaboration.

A three-level process specification is proposed in (Grefen et al. 2003). The three-level process framework consists of an external level, a conceptual level and an internal level. The internal level specifies a conceptual process definition such that it is taken to the local technological infrastructure (e.g., the local enterprise information systems). Internal level aims at the enactment of processes in the specific settings of an organization. The conceptual level is used for conceptual reasoning about a process, a combination of abstraction and aggregation of the internal level. The conceptual level defines a logical business view of a process without constraints by local technological infrastructures or external (market) requirements. The external level aims at communicating a process specification between different organizations, and makes partial conceptual processes visible to other business organizations in a market. The external level made partial conceptual process visible to other business organizations in a market.

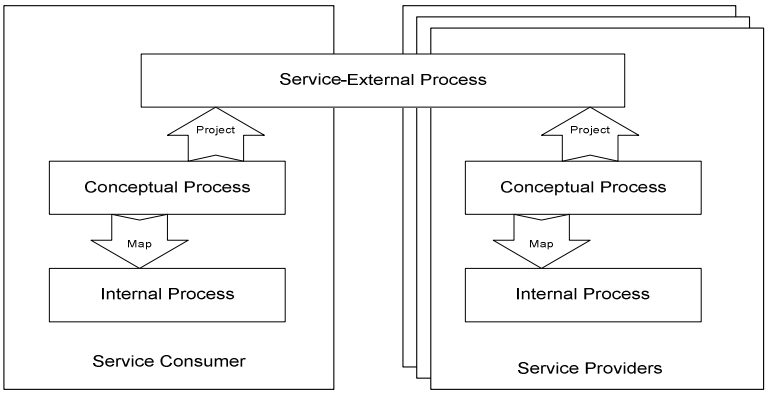

Figure A.1: Three-level process schema

The relationship between the three process levels is illustrated in Figure A.1. The above-presented Figure is slightly different than that in (Grefen et al. 2003). We duplicate the service providers' side to signify that the services requested by the service consumer may be collectively satisfied by multiple service providers.

APPENDIX B: DGA

The DGA capture dependencies between “aggregates and aggregates”, “aggregates and activities” and “activities and activities”.

The DGA is a tuple (A, G, E, *join, fork*) with

— $A = \{a_1, a_2, \ldots, a_n\}$ a set of activities.

— $G = \{g_1, g_2, \ldots, g_n\}$ a set of aggregates.

— $E \subseteq (A \times A) \bigcup (A \times G) \bigcup (G \times G) \bigcup (G \times A)$ a set of dependencies.

and functions *join* and *fork* label respectively the incoming and outgoing dependencies of an activity and an aggregate with the branching type:

$$join, fork: A \bigcup G \rightarrow \{AND, XOR\}$$

In the DGA, the activities and the aggregates are modeled as rectangles. The arrows represent the dependency relationships. For example, in Figure C.1 activity “A2” depends on activity “B2”, then the arrow goes from “B2” to “A2”. In the DGAs, the incoming and outgoing dependencies of an aggregate and an activity are typed with AND or XOR. Only one type, either AND or XOR, can be assigned to incoming resp. outgoing dependencies.

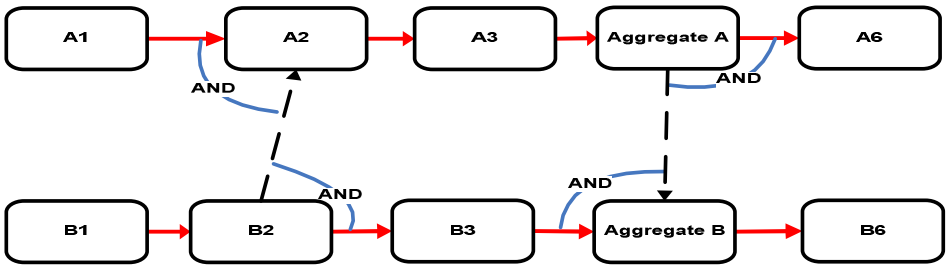

Figure B.1: Example DGA