A Framework for Service Outsourcing using Process Views

Rik Eshuis
Eindhoven University of Technology,
Faculty of Technology and Management,
Department of Information Systems,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
h.eshuis@tue.nl

Alex Norta
Department of Computer Science,
P.O. Box 68 (Gustaf Hällströmin katu 2b),
FI-00014 University of Helsinki, Finland
anorta@cs.helsinki.fi

***Abstract*—Service outsourcing is a business paradigm in which an organization has a part of its business process performed by a service provider. Process views are pivotal to support this way of working. A process view shields secret or irrelevant details from a private business process, thus allowing an organization to reveal only public, relevant parts of its private business process to partner organizations. The paper introduces a conceptual framework to support service outsourcing using process views. The framework gives rules that can be used to construct a process view from a conceptual process and vice versa. Based on these rules, the framework defines several projection relations that can exist between conceptual processes of consumers and providers and their process views. Finally, the framework gives a set of configuration options that specify which combinations of projection relations are useful for service consumers and service providers. The framework is applied in a BPEL-based case study.**

***Keywords*-Cross-organizational; process trees; matching; B2B; process visibility**

I. Introduction

The way companies collaborate with each other experiences significant changes. With the emergence of service-oriented computing (SOC), companies embrace the vision of using web services for engaging dynamically and flexibly in business-to-business (B2B) collaboration. Web services [3] are an important vehicle for enabling organizations to co-operate with each other and ways of inter-organizationally linking business processes [13], [14] with orchestration languages have been investigated, opening up the way to service outsourcing.

Service outsourcing is a business paradigm in which a service-consumer organization has a business-process part performed by a service-provider organization. Outsourced services need to expose details of the private provider process in a process view [9], [11]. Service consumers can use the process view to monitor and control the progress of service execution [12] at the provider.

Figure 1 shows a specification framework for service outsourcing proposed by Grefen et al. [11]. At the conceptual level, business processes are specified independent from technology, so no infrastructure and collaboration details are specified there. The internal level is infrastructure specific, so depending on existing technology. Conceptual processes are mapped to internal processes for enactment, see [25] for details. Both conceptual level and internal level processes are private to each organization. The external level contains a shared process view, which is fuelled by the conceptual process of the provider, and used by the consumer to monitor the progress and interact with its own local, inhouse process. The external process view is a projection of the conceptual level model, in which business internals at the conceptual level can be shielded from the environment. This paper focuses on process models at the conceptual and external level.

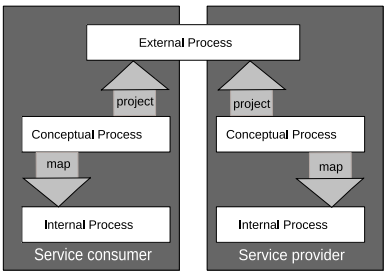

Figure 1. A three-level specification framework for service outsourcing.

While the distinction between conceptual and external processes is generally recognized as useful [5], [11], in the context of service outsourcing, concrete guidelines for constructing process views and relating them to underlying conceptual process are missing. A few approaches tackle the generic problem of constructing process views from conceptual processes [9], [19] or the reverse direction [2], but these approaches only consider one part of the problem, and are not specific to service outsourcing. Instead, they only support one kind of projection relation whereas service outsourcing requires multiple.

This paper fills the gap by defining a framework that uses projection rules for constructing a process view from a conceptual process. The rules ensure that the process view and its underlying process are consistent with each other. Based on these projection rules, we define extreme

projection relations that may exist between a conceptual process and a process view. Some of these projection relations have been identified in a Petri-net setting [25] before, but not in the context of abstraction and extension rules for process views. Moreover, we also show that in the context of outsourcing, only specific combinations of projection relations for consumer and provider make sense. We refer to related work [18], [25] for a mapping to the internal level.

The remainder of this paper is structured as follows. As preliminaries, Section II gives a running example of service outsourcing. Section III introduces a formal definition of process trees. Section IV defines projection rules on process trees that can be used for constructing a process view. The rules can also be used to extend an process view into a conceptual process. Based on the projection rules, Section V identifies several useful projection relations that exist between a conceptual process and a process view. Section VI defines configuration options for an outsourcing collaboration. Section VII presents a case study showing how the definitions and rules are applied in a BPEL setting. Section VIII presents related work. Finally, Section IX concludes this paper.

II. Overview

For the service-outsourcing approach of this paper, we focus on block-structured process models [17], or structured process models for short. Many existing process description languages, including industry standard BPEL [10] and OWL-S [22], are structured into blocks. Each block has a unique entry and a unique exit point, and blocks are properly nested. If a structured process model is sequential, its structure is similar to that of a structured program. Block-structured process models have the advantage that they do not contain structural errors such as deadlocks [17], for example, the block-structure forbids that an OR-split is immediately followed by an AND-join. Note that there exist approaches for converting an unstructured process model into a structured one [17], [27]. Extending the approach to unstructured, error-free process models is part of future work.

We introduce the problem of service outsourcing using process views by means of an example, adapted from [31]. In the next section, processes are formalized. Figure 2(a) shows an example consumer-process view of a telecom company. In the process, a GSM is delivered to a client. The bold lined nodes are invokable nodes: the corresponding activities like gG (get GSM) need to be initiated by the environment. All the other nodes in Figure 2(a) are observable but not initiated by the environment. The prefix c: for activities is used to indicate that the activities are consumer-side activities.

The consumer process can be completely outsourced. In Figure 2(b) and (c), two conceptual provider processes are shown. The prefixes px: and py: for an activity indicates that

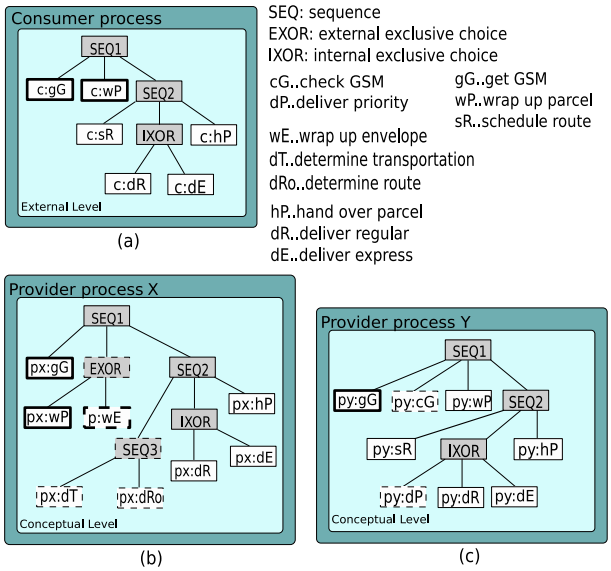

Figure 2. Consumer-process view and provider processes.

the activity is done at the provider-side. Both processes share some activities and ordering constraints with the consumer process, for example gG, but have some extra activities, that are not always observable. The question arises whether the consumer-process view is actually satisfied or implemented by these two conceptual provider process. We will answer this question in the next sections.

Finally, it must be mentioned that the consumer-process view of Figure 2(a) is located on the external level of a collaboration configuration while the provider processes are located on the conceptual level (see Figure 1). The conceptual consumer process (not shown here) is typically partitioned in two parts that interact with each other. One of the parts is outsourced through the consumer-process view, while the other part interacts with the provider processes through the consumer-process view. Section VII discusses this issue in more detail. In the remainder of this paper, we will use the example processes of Figure 2 to explain the main concepts and rules of the outsourcing framework.

III. Process Trees

As explained in the introduction, we consider structured processes, of which examples are presented in Figure 2. We formalize structured processes as trees, of which the leaves specify the execution of basic activities and internal nodes specify ordering constraints on their child nodes. The following ordering types are considered. A *sequence* node specifies sequential execution of children nodes. An *and* node specifies concurrent execution. An *xor* node specifies one of its child nodes is executed. There are two kinds of xor: an *internal* xor, where the choice is made by the system, and an *external* xor, where the choice is made by the

environment of the system. This distinction between internal and external choice is also made in other languages, notably CSP [29] and YAWL [1]. Finally, a while node specifies structured loops.

To cater for outsourcing, we employ a framework-distinguishing feature, namely observable and invokable activities [7], [28]. Observable activities are initiated and executed by the service provider, but can be monitored by the service consumer. Invokable activities are initiated by the service consumer (or requester) but executed by the service provider. Thus, a service consumer can exert more control over an invokable activity than over an observable activity. The sets of observable and invokable activities are disjoint. At the external level, each activity is either observable or invokable. At the conceptual level, an activity may also be not observable and not invokable, i.e., the collaborating counterpart does not perceive the enactment effects.

We now present a formal definition for process trees [8], [9]. A process tree P is a tuple $(A, N, child, type, rank, label)$ where:

- A is the set of basic activities. As subsets of A, we consider disjoint sets In, Ob, so $In \cap Ob = \emptyset$, where In is the set of invokable activities, and Ob is the set of observable activities,
- N is the set of nodes,
- $child \subseteq N \times N$ is a relation such that $(x, y) \in child$ if x is a child of y,
- $type : N \rightarrow \{BASIC, SEQ, AND, EXOR, IXOR, LOOP\}$ is a function that assigns to each node its type of which a $BASIC$ node has no children, a SEQ node specifies a sequential behavior, an AND node parallel behavior, an $IXOR$ and $EXOR$ node exclusive behavior, where the specific branch to be executed is decided internally for $IXOR$ and externally for $EXOR$ nodes, and a $LOOP$ node specifies iterative behavior,
- $rank : N \rightarrow \mathbb{N}$ is a partial function that is used to indicate the ordering of children of SEQ nodes. We require that two different nodes with the same parent have different ranks.
- $label : N \rightarrow A$ is a function labeling a basic node with a basic activity.

We require that the start activities under an $EXOR$ node are invokable. That is, the selection which branch of the $EXOR$ is to be executed externally by invoking the corresponding start activity of the branch.

For a node n, let $children(n) = \{x | (x, n) \in child\}$. By $children^*$ and $children^+$, we denote the reflexive-transitive closure and the irreflexive-transitive closure of $children$, respectively. If $n \in children^*(n')$, we say that n is a descendant of n' and that n' is an ancestor of n. In particular, each node n is ancestor and descendant of itself, so $n \in children^*(n)$, but $n \notin children^+(n)$. We require that the $child$ relation leads to a tree, so each node has one parent, except one node r, which has no parent. Additionally, it is required that r is ancestor of every node in N. These constraints ensure that nodes are structured in a tree with root r. Leaves of the tree are $BASIC$ nodes while internal nodes have type SEQ, AND, $EXOR$, $IXOR$, and $LOOP$.

An internal node may have only one child. In that case, if the internal node is not a $LOOP$ node, the internal node i can be safely removed by letting its child c become a child of the parent of i. We will use this transformation in the sequel when constructing process views.

IV. Projection Rules

In this section, we define three projection rules for constructing an external-level process view from a process at the conceptual level. The projection rules are useful for both provider and consumer side processes. The three rules are:

- Hiding: a set of nodes that are executed at the conceptual level are not shown in the process view at the external level.
- Omitting: a set of nodes that do not need to be executed at the conceptual level are not shown in the process view at the external level.
- Aggregation: a set of nodes that are executed at the conceptual level are aggregated into a single node in the process view at the external level.

At the end of the section, we explain how the rules can be used for constructing conceptual processes from external process views.

We now explain and define the three rules. Each rule takes as input a process tree P and a set S of nodes, and returns an abstracted process tree P' in which the input nodes are hidden, omitted, or aggregated into a new activity. For the last rule, the new aggregate activity is an additional input. In the descriptions, we use the concept of observable and invokable nodes, which are nodes of which the activities are labeled as observable or invokable, respectively.

Rule 1 (Hiding): A set of nodes is hidden in the view if the nodes and their corresponding activities are not shown in the process view, but still executed at the conceptual level. This way, execution details from the conceptual process can be hidden at the external level. If a single node is hidden, this coincides with projection inheritance [2]. Following the rules of projection inheritance, a hidden node can either be shown at the external level as an internal τ-action or be completely omitted.

However, hiding is only allowed if the set S of omitted nodes only contains observable nodes. An invokable node cannot be hidden, since the corresponding activity, which is executed at the conceptual level, cannot be invoked if it is hidden. Moreover, set S must form a complete branch of a SEQ node or an AND node or a $LOOP$ node, that is, there is a node $n \in S$ such that n is ancestor of all nodes in S, so S is a subtree with root n, and the parent of s is a SEQ, AND, or $LOOP$ node. If S is a complete branch

of an $EXOR$ or $IXOR$ node, then hiding is not allowed, since the branch cannot be omitted at the external level.

Let P be a process tree $(A, N, child, type, rank, label)$. If $S \subseteq N$ is the set of nodes to be hidden from process tree P, then the resulting process tree P' is defined as $(A', N, child, type, rank, label')$ where :

- $A' = \{\ y \mid \exists x \in N' \setminus S : (x, y) \in label'\ \}$
- $label' = label \oplus \{(n, \tau) \mid n \in S\}$

To illustrate the definition, consider Figure 2(c). Node p:cG can be hidden; it's label then becomes τ. However, node p:DP cannot be hidden, since its parent is an $IXOR$ node. This means the provider decides internally whether p:dP or one if its sibling nodes is entered. Thus, p:dP is always executable, and cannot be omitted. If p:dP were hidden, so p:dP is not shown in the process view, then the process view would not offer the appropriate status to the consumer if p:dP is active at the conceptual level, since the τ action hides the business activity actually being executed.

Rule 2 (Omitting): A set of nodes is omitted in the process view if the nodes and their corresponding activities are not shown in the process view and not executed at the conceptual level. If a single node is omitted, this coincides with protocol inheritance [2]. There is no restriction on the set of omitted nodes, that is, omitted nodes can be observable, invokable, or neither observable nor invokable.

However, omitting is only allowed if the set S of omitted nodes forms a complete $EXOR$ branch, or more precisely, if there is a node n such that S is a subtree with root n, so $S = children^*(n)$, and n has an $EXOR$ parent. In all other cases, omitting the nodes results in a process tree with deadlocks. For example, if in Figure 2(b) node SEQ3 and its children are omitted, the conceptual process deadlocks, since IXOR requires SEQ3 is completed before it can start.

Note that if n has an $IXOR$ parent, omitting is not allowed. For example, if from Figure 2(b) node p:dR is omitted, then the provider-process view does not contain this node, but the provider conceptual process still does, and therefore may decide to execute it. If n has an $EXOR$ parent, this is not true, since there the consumer, not the provider process, decides which choice is made

Let P be a process tree $(A, N, child, type, rank, label)$. If $S \subseteq N$ is the set of nodes to be omitted from process tree P, then the resulting process tree $P' = (A', N', child', type', rank', label')$ is defined as:

- $A' = \{\ y \mid \exists x \in N' : (x, y) \in label'\ \}$
- $N' = N \setminus S$
- $child' = child \cap (N' \times N')$
- $type' = type \cap (N' \times \{BASIC, SEQ, AND, EXOR, IXOR, LOOP\})$
- $rank' = rank \cap (N' \times \mathbb{N})$
- $label' = label \cap (N' \times A)$

Note that the $rank$ numbers does not need to be updated, so the rank of the nodes that are in P stays the same for P', because after omitting nodes, still two nodes sharing the same parent have different ranks.

The omitting rule can be applied to Figure 2(b). By omitting node p:wE, node EXOR has one child only. As explained in Section III, then EXOR can be eliminated too, and thus p:wP becomes child of SEQ1.

Rule 3 (Aggregation): If nodes are aggregated, they are still executed at the conceptual level, but not explicitly shown in the process view at the external level. Instead, a new node n_{agg} with a new activity a_{new} is shown. However, this is only allowed if each of the aggregated nodes is observable, so not invokable.

For the purpose of this paper, a set of nodes S is an aggregate if they form a subtree, so they share a common ancestor n of which all descendants are in S, so $S = children^*(n)$. For example, in Figure 2(b) set {SEQ3,p:dT,p:DRo} is an aggregate, since SEQ3 is common ancestor to all nodes in the set, and all descendants of SEQ3 are in the set. For a more complex definition, that allows an aggregate to have multiple common ancestors, which share the same parent that is not in the aggregate, we refer to [9].

Let agg be an aggregate with common ancestor n for process tree $P = (A, N, child, type, rank, label)$, so $agg \subseteq N$, and let a_{new} be the new basic activity that replaces the activities corresponding to the nodes in agg. Then the process tree under aggregation P' is constructed by replacing agg with a new node $n_{agg} \notin N$ that does not get any children in the process view P' and gets label a_{new}. Node n_{agg} is attached to the parent l of n.

Formally, $P' = (A', N', child', type', rank', label')$ where

- $A' = \{\ y \mid \exists x \in N' : (x, y) \in label'\ \} \cup \{a_{new}\}$
- $N' = N \setminus agg \cup \{n_{agg}\}$
- $child' = (child \cap (N' \times N')) \cup \{(n_{agg}, l)\}$
- $type' = type \cap (N' \times \{BASIC, SEQ, AND, EXOR, IXOR, LOOP\}) \cup \{(n_{agg}, BASIC)\}$
- $rank' = (rank \cap (N' \times \mathbb{N})) \cup \{(n_{agg}, rank(n))\}$
- $label' = (label \cap (N' \times A)) \cup \{(n_{agg}, a_{new})\}$

The aggregation rule can be applied to Figure 2(b): node SEQ3 and its children can be aggregated into node p:sR (which becomes c:sR at the consumer side in Figure 2(a)).

Example: Now that we have explained and illustrated the abstraction rules, we revisit the example processes in Figure 2. Let Figure 2(a) specify the process view and let Figure 2(b) and (c) be provider processes. Then Figure 2(a) is a correct view of Figure 2(b), but not of Figure 2(c). For the latter, no hiding or omitting rule can be applied to the node p:dP, which has no counterpart in the consumer-process view.

Extending process views: The projection rules are not only useful for constructing external process views from conceptual processes, but also for deriving a conceptual process from a process view. The conceptual process can be

created in a process editor by adding or inserting activities and control flow to the process view. If this extension is equivalent to applying the thee projection rules in a certain order, the derivation is allowed. For example, this allows the derivation of Figure 2(b) from Figure 2(a) since the extensions can be omitted and aggregated.

V. Projection Relations between Process View and Conceptual Process

In the previous section, we defined general rules that can be used to transform a conceptual process to a process view and vice versa. In this section, we look at possible projection relations that may exist between a process view and a conceptual process. Each projection relation is realized using a combination of the projection rules. Table I lists the extreme projection relations we consider. Black box, glass box, and open box have been identified in a web service outsourcing setting by Grefen et al. [12] while Norta [25] has identified gray-box and white-box projection in a Petri-net setting. All the other possible projection relations are hybrid forms of these extreme relations. In the remainder of this section, we explain the projection relations listed in Table I.

Projection Relation	Omitting	Hiding	Aggregation	Relation between process view and conceptual process
Black box		x	x	process view is single observable activity, conceptual process has no invokable activities
Glass box		x	x	process view has only multiple observable activities, conceptual process has no invokable activities
Gray box	x	x		process view has multiple invokable and observable activities, conceptual process has invokable and observable activities
Open box	x	x	x	process view has multiple invokable and observable activities, conceptual process has invokable and observable activities
White box				process view and conceptual process are identical

Table I
Extreme projection relations between conceptual process and external process view.

Black-box projection occurs if the external process tree contains only a single node with a single observable activity. Thus, the nodes in the conceptual process are aggregated or hidden into this single node in the process view. Invokable nodes cannot be hidden or aggregated, so the conceptual process does not contain any invokable nodes. Moreover, since the external process tree cannot contain any $EXOR$ node with invokable nodes as descendants, omitting is not used.

Glass-box projection is realized if the process view only contains observable activities; the consumer cannot invoke any of the provider activities. A glass box view can be obtained through hiding and aggregation from the conceptual process. Since the process view does not contain any invokable nodes, omitting is not used. Black-box projection can be seen as a special case of glass-box projection.

Gray-box projection is established if the process view is obtained through hiding and omitting from the conceptual process. The process view can contain both observable and invokable activities. However, aggregation is not used. Norta [25] identifies gray box by only applying hiding as abstraction operator.

Open-box projection is achieved if the process view is obtained through hiding, omitting, and aggregation from the conceptual process. The process view can contain both observable and invokable activities. Thus, the consumer can influence the progress at the provider side.

Finally, to use a *white-box projection*, the process view is identical to the conceptual process. Thus, none of the abstraction rules is applied, and the consumer has a direct view on the conceptual process of the provider. A white box can be seen as a special case of an open box.

Revisiting the processes in Figure 2, we have that the consumer-process view in Figure 2(a) is white box related to the conceptual consumer process if that process is identical to Figure 2(a). Next, Figure 2(a) is related by an open-box projection to Figure 2(b), since both omitting and aggregation rules are applied. Both provider processes in Figure 2(b) and (c) cannot be related to Figure 2(a) using black-box or glass-box projections, since both processes contain at least one invokable node.

In summary, the projection relations support a collaboration scenario in which collaborating parties can expose in flexible ways only so many details of their conceptual processes as they deem necessary. However, as we show in the next section, not every combination of projection relations for consumer and provider-side is suitable for achieving external level harmonization between consumer and provider.

VI. Collaboration Configuration for Outsourcing

While the previous section identified some extreme projection relations that can exist between a conceptual process and an external process view, we now turn to the concrete setting of service outsourcing between a service consumer and a service provider. We first list the possible configuration options that define how the conceptual processes of the consumer and provider and the shared process view can relate to each other.

A. Configuration Options

This section presents the possible configuration options for a collaboration between a service consumer and a service provider. The options are visualized in Figure 3. As explained in the introduction, we abstract from the internal level of the three-level model of Figure 1.

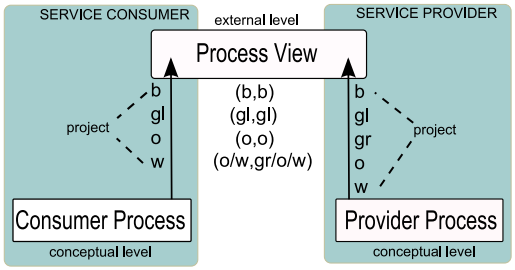

Figure 3. A collaboration configuration.

The conceptual levels of Figure 3 show on the left and right side the consumer and provider process, respectively. For both parties, the useful projection relations are depicted near the projection arrow. In the middle of Figure 3, the tuples with projection combinations represent the meaningful options for establishing process views on the external level. We now explain and motivate these combinations.

If the service consumer performs a black-box projection, the service provider only may use a black-box projection. All other projections result in processes having more than one activity, and thus the resulting process view of the provider does not equal the consumer-process view. Likewise, if the service consumer performs a glass-box projection, the service provider can only use a glass-box projection, since that is the only projection relation resulting in an external process view with only observable activities. If a service consumer performs an open-box or white-box projection, the service provider may respond with either a gray-box, open-box, or white-box projection. Since the consumer-view may contain invokable activities, black-box and glass-box projections are not applicable

Finally, a service consumer cannot use a gray-box projection. To see why, suppose the outsourced conceptual process of the service consumer contains an invokable node that is omitted in the consumer-process view. Since the invokable node is not in the process view, the provider process at the conceptual level does not need to have a corresponding invokable node. However, since the original node at the consumer-side is invokable, that node can be invoked by some other internal processes of the consumer that interacts with the process that is being outsourced. But then the provider process cannot replace the outsourced consumer process. Therefore, if the conceptual process of the consumer contains an invokable node, only open-box and white-box projection are applicable.

B. Enactment Deployment

Since an established collaboration configuration is distributed on several levels, the deployment requires special attention. Given that internal business process content from different parties is projected to the external level in different ways, it must be ensured before enactment that a collaboration configuration is correct and deadlock free. For this purpose, collapsing the consumer and provider processes is useful. Also, collapsing is useful to show how the actual outsourcing can take place during enactment.

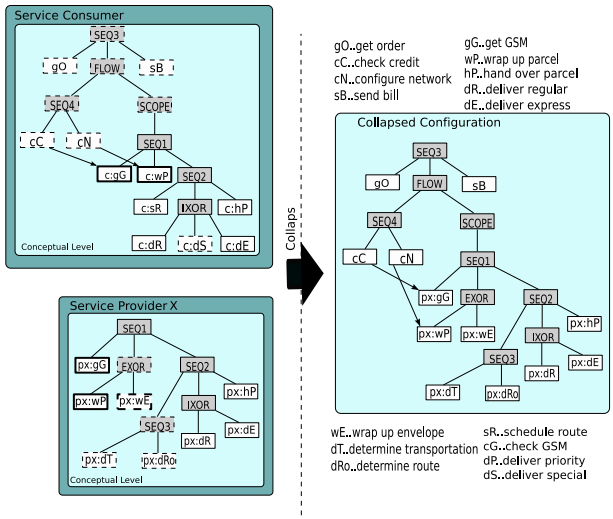

Figure 4. A collapsed collaboration configuration.

Figure 4 shows the application of an existing collapsing method [25] to the example processes in Figure 2(a) and (b), for which the collaboration configuration is white-box projection for the consumer and gray-box projection for the provider. The outsourced consumer process is embedded in a larger consumer process, which is shown on the top left of Figure 4. The subtree below SCOPE is the conceptual consumer process that is outsourced; it is identical to Figure 2(a). Note that consumer activity cC invokes outsourced activity gG while cN invokes wP. The provider process on the bottom left is identical to Figure 2(b). On the right hand of Figure 4, the collapsed collaboration configuration is depicted in which the consumer scope of the overall consumer process is replaced with the provider process.

The resulting collapsed configuration on the right side of Figure 4 needs to be evaluated with tool support for correctness issues, e.g., deadlocks or lack of synchronization. If the resulting process tree is represented in BPEL, it is possible to verify properties if, e.g., a mapping to place/transition-nets is performed [20].

Since the collapsing method requires that the conceptual-level processes of all collaborating domains are disclosed, the replacement must be performed by an independent trusted third party. Otherwise, it is likely that business secrets are disclosed, which might result in a loss of competitive advantages. Alternatively, the collapsing can be guaranteed to be correct if the projection options identified in Section VI-A are followed. Then deadlocks cannot occur, since for example invokable nodes of the consumer are guaranteed

to be preserved at the provider side.

Finally, it must be stressed that BPEL currently does not cater for constructs that can link the respected processes on the different levels of a collaboration configuration. For that it would be necessary to have extra language constructs available that link the various processes so that the enactment progress is monitorable in a flexible way. For example, a service consumer may find it uneconomical to monitor the enactment progress of every basic activity. Such a situation could also be undesirable for the service provider, who does not want to be monitored extensively during enactment. In Figure 4, links are used for starting and terminating the provider process. However, if the collaborating parties agree, cross-platform linking constructs should also be available to observe the enactment progress of only specific basic activities contained in a process tree. In [25], [26] such monitorability constructs are explained in further detail.

VII. Case Study

To show the feasibility of the approach, we next discuss a case study about inter-organizational business process collaboration from the IST CrossWork[1] project. An example of a BPEL process is depicted in Figure 5(a), which is a conceptual-level process of a service consumer. In Figure 5, several details are abstracted from, e.g., the full definitions of partner-links, variables, and so on. Instead, the focus lies on the structural business-process behavior.

The process behavioral definition of the example in Figure 5(a) starts with a sequence node in Line 14 that lists the ordering of a watertank as first task. Then a flow construct has two parallel branches embedded in sequences. The first branch starts with an invoke node for preparing the watertank specification in Line 18, which is linked to another invoke node in Line 27 to indicate that the latter node is invokable.

The distinction between invokable and observable is not made in current BPEL, since BPEL does not fully support service outsourcing [12]. For that purpose, we use a slight extension to standard BPEL in Line 5 of Figure 5(a). Nodes that are either invokable or observable specify that with a status attribute, e.g., in Line 27 the receive node has the status invokable. In [12], suitable extension suggestions for BPEL are contained.

In Line 27, the invoke node is embedded in a scope that demarcates the part of the conceptual-level process that is sourced from a collaborating counterpart, which is an approach in line with proposed business-process separation and collaboration methods [16], [25]. After configuring the production resources, a flow construct embeds two parallel nodes for producing the tank body and the pump engine. After that, a node in Line 34 starts the assembly of all parts into the finished water tank. That node of Line 34 is also linked with the invoke node for preparing the payment that is located in Line 21. Finally, after all tasks in the flow are completed, payment takes place. In Figure 5(b), the projected process view is listed and both the conceptual and external processes are visualized as process trees.

In Figure 6, the corresponding provider process is depicted that is located on the conceptual level. The BPEL process contains a similar structure as the process view in Figure 5(b). However, additionally inserted nodes are depicted as dashed boxes in the process tree. Since these nodes only exist within the domain of the service consumer, the service provider can not be aware of them during enactment time, although the perceived process behavior conforms to the specifications of the external level. Likewise, the service provider is not aware of the nodes in Figure 5(a) that are located outside of the scope demarcation. Again, dashed boxes depict in the corresponding process tree.

Realizing service outsourcing as put forward in this paper, requires new setup and enactment application systems with components that populate the three-level framework of Figure 1 with required functionality. Due to page limitation, we refer to [25] for a service-outsourcing reference architecture. An implementation of this architecture was carried out during the CrossWork project. In [23], [24] further information about the implemented proof-of-concept prototype and the CrossWork project can be found.

VIII. Related Work

In the area of inter-organizational business-process collaboration, related work exists. We focus on the two most related sub-areas: process views and visibility patterns.

A. Process Views

The importance of process views for service outsourcing has already been recognized in previous papers [6], [12], [30]. However, these approaches typically focus on how a process-view can be supported at run-time and do not address how a process view can actually be constructed. Chebbi et al. [6] also consider the construction of process views, but use only one projection relation. The framework defined in this paper is most closely related to the work of Grefen et al. [12], which defines several of the projection options identified in Section V, but the framework can also be combined with the other papers.

Other papers have studied how process views can be constructed [2], [4], [9], [11], [19]. Van der Aalst and Weske [2] define how local processes can be derived from a global process view. The projection relations they identify correspond to hiding and omitting. Bobrik et al. [4] study the construction of personalized role-based process views. Eshuis and Grefen [9] and Liu and Shen [19] focus on how a process view can be derived from a conceptual process. Aggregation is used as a key abstraction principle in both

[1]CrossWork: Cross-Organizational Workflow Formation and Enactment, IST no. 507590. http://www.crosswork.info

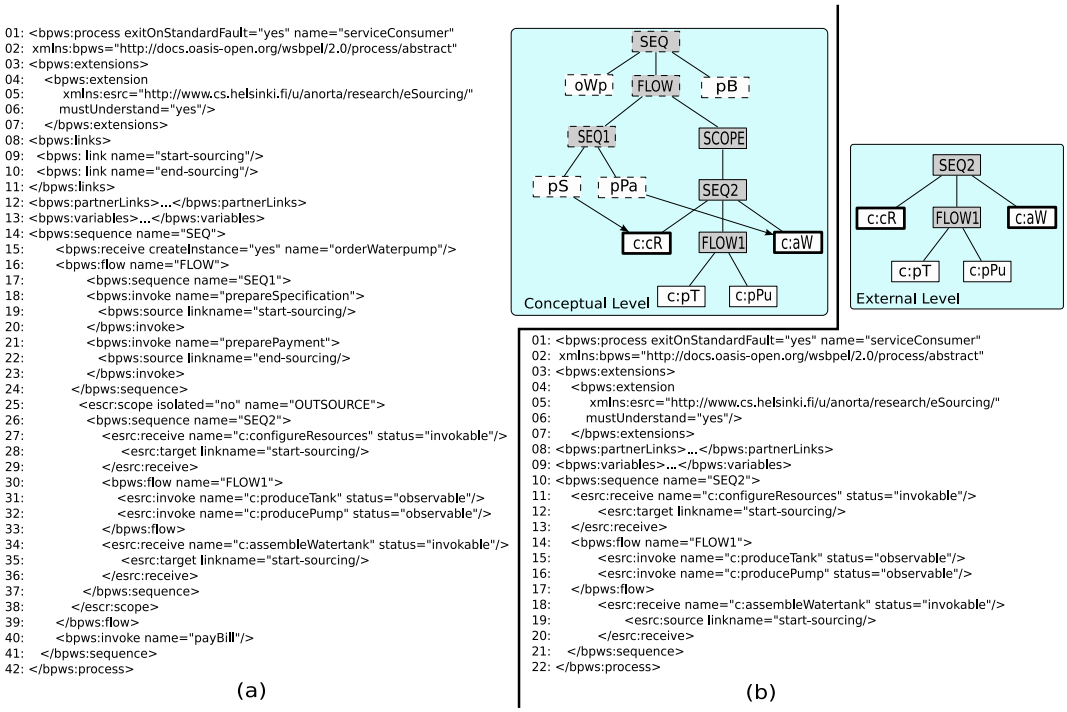

Figure 5. The conceptual and external processes of the service consumer.

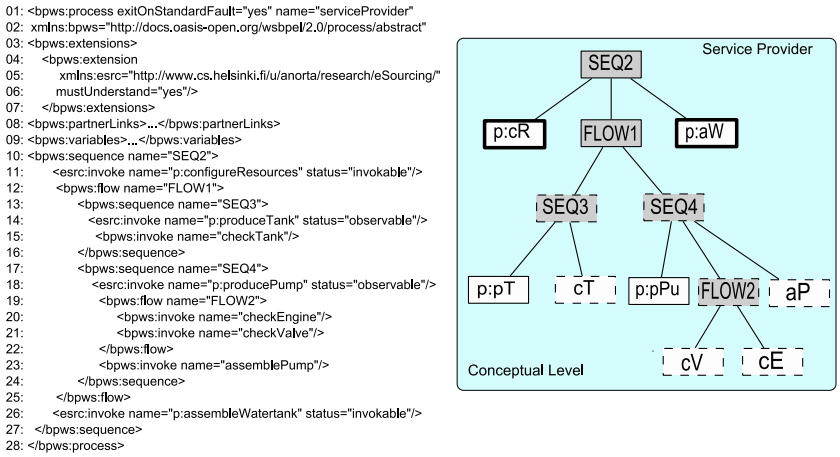

Figure 6. The conceptual processes of the service provider.

approaches, while Eshuis and Grefen also use a form of hiding. None of these process-view approaches focus on service outsourcing, and consequently none identify projection options and meaningful configurations for partners in an outsourcing collaboration.

Preuner and Schrefl [28] define an approach for combining several process-based services into a compound process. They define two consistency relations between the composed process view and the underlying processes: observability consistency and invokability consistency. Observability consistency resembles hiding, whereas invokability consistency resembles omitting. They do not consider aggregation. Also, they do not consider different projection options and collaboration configurations.

Next, the BPEL standard [10] distinguishes between abstract and executable processes where abstract processes correspond to process views. However, no concrete guidelines are offered for relating abstract and executable processes. Khalaf et al. [15] discuss abstract patterns suc for relating an abstract process to an executable process. The

patterns related to the framework in our paper are export (creating an abstract process from executable process) and import (creating an executable process from an abstract process). The three projection rules in Section 4 can all be used in combination with the export and import pattern. Martens [21] proposes a Petri net-based approach to check the consistency of an abstract and an executable BPEL process. König et al. [18] and Zhao et al. [32] define several syntactic guidelines for transforming an abstract BPEL process into an executable one. The rules employ some kind of hiding and omitting. None of these papers distinguishes between different projection relations and none consider aggregation.

Finally, there is theoretical work on the problem of compatibility checking of Petri-net based services [20]. In principle, this work could be used to check the compatibility of a consumer and a provider process. However, there any combination of services that does not deadlock is correct, which is not suitable for outsourcing. The requirements for outsourcing are much more strict, since a process view must mirror the provider process. Thus, even though a consumer process and provider process are compatible, they may not be in an outsourcing relation, since for example the provider can remove some observable activity that the consumer needs to monitor.

B. Related Process-Visibility Patterns

In [25], [26], so-called contractual-visibility patterns are identified and specified for inter-organizational business-process collaboration, which also assume that a partitioned conceptual-level process, i.e., a sphere, is projected to an external collaboration level. However, differently to the process-view patterns described in Section V, the exploration of contractual-visibility patterns are Petri-net based. For the latter patterns, in *black-box* visibility, only the interfaces of a sphere are projected to the external level. *White-box* visibility means that all nodes of a sphere are projected to the process of the external level. Finally, *gray-box* visibility results in the interfaces and a subset of the nodes and arcs of the conceptual-level sphere being projected to the external level.

The main difference between contractual-visibility and process-view patterns is that the latter also incorporate the accessibility of projected nodes i.e., invokable or observable. The contractual-visibility patterns only focus on the relationship between the sets of nodes in the processes of the conceptual and external level and accessibility is covered with separate so-called monitorability patterns [25]. Hence, all process-view patterns can be realized with combining the contractual-visibilities with monitorability patterns.

IX. Conclusion

We have proposed a conceptual framework for service outsourcing, in which a service consumer outsources parts of its business process to a service provider. To support outsourcing, the distinction between observable and invokable activities is vital. The framework distinguishes between process views and conceptual processes. It defines several rules to construct process views from conceptual processes and vice versa. The key abstraction principles used in the rules are hiding, omitting and aggregation.

Next, based on the abstraction rules, the framework defines several extreme projection relations that can exist between a process view and conceptual process. These projection relations also give insight into the possible types of a process view. We have shown that existing approaches for process views from literature mostly focus on only one of these projection relations. We believe the projection relations and especially the projection rules are key to realize any concrete outsourincing relation in practice, but more analysis using case studies is needed to assess whether the framework is complete.

We also identified collaboration configurations for a service consumer and service provider. In particular, we have shown that not every combination of projection relations for provider and consumer-side processes is meaningful. For evaluation and verification purpose of collaboration configurations, a collapsing method is proposed that establishes a process tree in which the consumer scope of the main process is replaced by the provider process. Since it is important that the collaborating counterparts retain their business secrets, such a collapsing method needs to be carried out by a trusted third party.

Open issues for future work research focus mainly on supporting applications for setting up and enacting collaboration configurations. As process description language, we will consider BPEL. However, BPEL needs to be extended with additional language constructs to allow a service consumer to start and stop the enactment of provider processes and it must also be possible for a service consumer to remotely observe the enactment progress of the provider process. For a distributed setup and enactment of a collaboration configuration, it is necessary to develop a reference architecture for supporting application systems. Another topic for future research is applying process matching to ontology languages like OWL-S.

Acknowledgements

We thank Lea Kutvonen and the reviewers for their valuable feedback and advice that helped to improve the paper.

References

[1] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[2] W.M.P. van der Aalst and M. Weske. The P2P approach to interorganizational workflows. In *Proc. of the 13th Int. Conference on Advanced Information Systems Engineering (CAiSE'01)*, Lecture Notes in Computer Science 2068, pages 140–156. Springer, 2001.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag Berlin Heidelberg, 2004.

[4] R. Bobrik, M. Reichert, and T. Bauer. View-based process visualization. In *Proc. Business Process Management (BPM) 2007*, volume 4714 of *Lecture Notes in Computer Science*, pages 88–95. Springer, 2007.

[5] C. Bussler. The Application of Workflow Technology in Semantic B2B Integration. *Distributed and Parallel Databases*, 12:163–191, 2002.

[6] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, 2006.

[7] J. Ebert and G. Engels. Observable or invocable behaviour - you have to choose! Technical report 94-38, Leiden University, 1994.

[8] R. Eshuis and P. Grefen. Structural matching of bpel processes. In *ECOWS '07: Proc. of the Fifth European Conference on Web Services*, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[9] R. Eshuis and P. Grefen. Constructing customized process views. *Data and Knowledge Engineering*, 64(2):419–438, 2008.

[10] A. Alves et al. Web services business process execution language version 2.0 (OASIS standard), 2007.

[11] P. Grefen, H. Ludwig, and S. Angelov. A Three-Level Framework for Process and Data Management of Complex E-Services. *International Journal of Cooperative Information Systems*, 12(4):487–531, 2003.

[12] P. Grefen, H. Ludwig, A. Dan, and S. Angelov. An analysis of web services support for dynamic business process outsourcing. *Information and Software Technology*, 48(11):1115–1134, 2006.

[13] J.Y. Jung, H. Kim, and S.H. Kang. Standards-based approaches to b2b workflow integration. *Comput. Ind. Eng.*, 51(2):321–334, 2006.

[14] R. Khalaf. From rosettanet pips to bpel processes: A three level approach for business protocols. *Data Knowl. Eng.*, 61(1):23–38, 2007.

[15] R. Khalaf, A. Keller, and F. Leymann. Business processes for web services: Principles and applications. *IBM Systems Journal*, 45(2):425–446, 2006.

[16] R. Khalaf and F. Leymann. E role-based decomposition of business processes using bpel. In *ICWS '06: Proc. of the IEEE International Conference on Web Services*, pages 770–780, Washington, DC, USA, 2006. IEEE Computer Society.

[17] B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *Proc. CAiSE '00*, pages 431–445. Springer, 2000.

[18] D. König, N. Lohmann, S. Moser, C. Stahl, and K. Wolf. Extending the compatibility notion for abstract WS-BPEL processes. In *WWW '08: Proc. of the 17th international conference on World Wide Web*, pages 785–794, New York, NY, USA, 2008. ACM.

[19] D.-R. Liu and M. Shen. Workflow modeling for virtual processes: an order-preserving process-view approach. *Inf. Syst*, 28(6):505–532, 2003.

[20] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting ws-bpel processes using flexible model generation. *Data Knowl. Eng.*, 64(1):38–54, 2008.

[21] A. Martens. Consistency between executable and abstract processes. In *Proc. of Intl. IEEE Conference on e-Technology, e-Commerce, and e-Services (EEE'05)*. IEEE Computer Society Press, mar 2005.

[22] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, and S. McIlraith. *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/OWL-S/, 2004.

[23] N. Mehandjiev and P. Grefen. Crosswork: Internet-based support for process-oriented instant virtual enterprises. *IEEE Internet Computing*, 2010. to appear.

[24] N. Mehandjiev and P. Grefen, editors. *Dynamic Business Process Formation for Instant Virtual Enterprises*. Springer, 2010.

[25] A. Norta. *Exploring Dynamic Inter-Organizational Business Process Collaboration*. PhD thesis, Technology University Eindhoven, Department of Information Systems, 2007.

[26] Alex Norta and Paul Grefen. Discovering Patterns for Inter-Organizational Business Collaboration. *International Journal of Cooperative Information Systems*, 16(3/4):507–544, 2007.

[27] C. Ouyang, M. Dumas, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research*, 5(1):42–62, 2007.

[28] G. Preuner and M. Schrefl. Requester-centered composition of business processes from internal and external services. *Data Knowl. Eng*, 52(1):121–155, 2005.

[29] W. A. Roscoe. *Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[30] K.A. Schulz and M.E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.

[31] J. Vonk and P.W.P.J. Grefen. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases*, 14(2):137–172, 2003.

[32] X. Zhao, C. Liu, W. Sadiq, M. Kowalkiewicz, and S. Yongchareon. Ws-bpel business process abstraction and concretisation. In *Proc. DASFAA 2009*, volume 5463 of *Lecture Notes in Computer Science*, pages 555–569. Springer, 2009.