Measures and mechanisms for process monitoring in evolving business networks

Marco Comuzzi[a,*], Jochem Vonk[a], Paul Grefen[a]

[a]*School of Industrial Engineering, Eindhoven University of Technology, P.O. Box 513, 5600MB, Eindhoven, The Netherlands*

Abstract

The literature on monitoring of cross-organizational processes, executed within business networks, considers monitoring only in the network formation phase, since network establishment determines what can be monitored during process execution. In particular, the impact of evolution in such networks on monitoring is not considered. When a business network evolves, e.g. contracts are introduced, updated, or dropped, or actors join or leave the network, the monitoring requirements of the network actors change as well. As a result, the monitorability of processes in the network may be disrupted. This paper proposes a framework to solve the problem of preserving the monitorability of processes in an evolving business network. We first propose a formal model of business networks, contracts, and monitoring requirements. Then, we model network evolution and the mechanisms to preserve the monitorability of the processes in the network for different types of evolution. In particular, the preservation of monitorability requires the actors in the network to take appropriate actions in case of dependencies between already established contracts, and update their monitoring infrastructure to satisfy the new monitoring requirements introduced by evolution. We also define a set of metrics that can be used for supporting decisions regarding the potential evolution of a business network. A case study in healthcare and the discussion of a prototype implementation show the applicability of our framework in real-world scenarios.

Keywords: monitoring, contracts, business networks, business process, business network evolution, monitoring capabilities, monitoring metrics, service monitoring.

1. Introduction

Faster market dynamics and fiercer competition push organizations to engage in complex, Internet-enabled, highly dynamic collaborations, referred to as virtual enterprises (organizations) or collaborative Business Networks (BNs) [28, 68]. Collaboration entails the enactment of cross-organizational business processes, which are regulated by agreements between the actors constituting the business network [2, 5, 14]. Typical examples of business networks can be found in the automotive industry, where a main player, i.e. the vehicles manufacturer, coordinates a complex network for the procurement of spare parts [28]. Other examples can be found in the financial industry where, for instance, an insurance company may outsource parts of its processes, such as loss adjustment, complaints management, and fraud detection, to external companies, in order to focus on its core business.

Information technologies (IT) supporting BNs help the network partners to lower coordination costs, i.e. reducing costs for communication [38], increasing opportunities for matchmaking, and enabling a more agile switch among business partners [53]. As a form of market-based collaboration, however, BNs, when compared with in-house business process execution, are characterized by higher control costs [24, 38], which

*Corresponding author

Email addresses: m.comuzzi@tue.nl, Tel: +31 40 247 2183 (Marco Comuzzi), j.vonk@tue.nl, Tel: +31 40 247 5804 (Jochem Vonk), p.w.p.j.grefen@tue.nl, Tel: +31 40 247 5650 (Paul Grefen)

IT can help to reduce as well. When part of a business process is outsourced to external business partners, in fact, an organization faces the need for controlling the execution of the outsourced process. Control is characterized by risks derived from the possible inability of external providers to meet the expectations of the outsourcing organizations [69]. Risks may translate into costs for the outsourcing organization. Control costs can be ascribed either to the opportunistic behavior of the external providers (moral hazard) or to their unsuitability to perform the task that has been assigned to them (adverse selection) [1, 7].

From a governance perspective [11], organization have to consider a strategy to reduce the risk of increasing control costs. Organizations can choose to establish contracts with other partners in the BN and/or monitor the execution of the outsourced business processes.

Contracts shift (part of) the risk of the external provider not meeting the expectations of the outsourcing organization from the outsourcer to the external provider. In case the contract terms are not met, the contract may define penalties for the external provider or, even in case the contract does not define penalties, the external provider's reputation may decrease, making it less likely to be selected in the future by the same outsourcer [32].

Contracts, however, are a *passive* element of control, since they are signed before the collaboration takes place. Contracts should always be coupled with the *active* monitoring of the business process taking place during the enactment of the collaboration [55]. By collecting relevant information on the externally executed business processes, the outsourcer can detect non-compliance of established contracts and decide which control actions should be taken. Monitoring an external business process can be performed while it is being executed, or after it has been completed (by reconstructing the relevant aspects of a process). The former is referred to as *on-line* monitoring [21], *run-time* monitoring [9], *continuous* monitoring [18, 3], or, simply, process monitoring [35, 76]. The latter is usually referred to as *ex-post* monitoring [18], *off-line* monitoring [21], or process controlling [76]. This paper focuses on the former type of monitoring.

Monitoring is required to improve collaboration, for instance, in cases of *synchronization* issues on the consumer side or in the case of *dynamic sourcing*. Synchronization is needed when a decision point in the consumer process, e.g. a XOR split, requires information generated by the provider in its own process [30]. The synchronization between the consumer and provider is feasible only if the provider makes available such information to the consumer, i.e. if the provider matches the consumer's monitoring requirements on the outsourced process. Note that each consumer may have different monitoring requirements and, therefore, the provider cannot embed the provisioning of monitoring information directly in its own process.

Dynamic sourcing represents the case in which a consumer selects a provider at the very last possible moment during the enactment of a process [28, 68]. Dynamic sourcing scenarios often imply spot collaboration, i.e. outsourcing may occur only during the enactment of one specific instance. In this case, the only way for the consumer to verify the fulfillment of a contract is through monitoring while the outsourced process is executed. The consumer, in fact, is not guaranteed to make business with the same provider in the future and, most importantly, the provider may leave the business network before the consumer can run the contract ex-post verification.

This paper proposes a framework to solve the problems arising in the aforementioned collaborative scenarios when contract establishment and monitoring are not jointly considered as control mechanisms. If the consumers monitoring requirements are only written in contracts, but the provider does not make information available to match those, then the consumer is prevented from verifying the satisfaction of such contracts. Specifically, in the synchronization scenario the consumer will not be provided with the right information to synchronize its process, whereas in the dynamic sourcing scenario the consumer will simply not have a means to verify the fulfillment of the established contracts. Similarly, the dynamic sourcing scenario becomes particularly critical when monitoring information is available to the consumer, but not associated to an established contract. Without a contract, the consumer has no guarantee on the behavior of the provider, e.g. on the provided quality of service.

In collaborative settings, control through contract establishment and business process monitoring becomes additionally complex because:

- Process execution is likely to be distributed across a complex network of providers [2]. Contracts between providers, therefore, should be *balanced* to make the expectations of the providers contributing

to the execution of the process coherent. In other words, outsourcing entails a dependency relation among contracts. If, for instance, an organization has established an agreement on the completion time of the activities delegated to an external actor, then the external actor should project such requirement in the contracts established with its own providers;

- BNs can evolve as new business opportunities arise. Evolution, e.g. partner substitution, outsourcing, or contract revision, is likely to disrupt the dependency among contracts and their monitorability. Thus, partners in the network should strive for reestablishing the correct dependencies among contracts and for updating their monitoring infrastructure as new monitoring requirements arise. As a sample, outsourcing in highly regulated industries, such as healthcare or finance, often requires the outsourcing organization to ensure that a process, when outsourced, still complies with strict regulations on privacy on personal data management or traceability of billing and payments.

Hence, the framework presented in this paper, besides jointly considering contracts and monitoring of business processes as a means to reduce the risk for synchronization and dynamic sourcing in business networks, focuses also on the problem of preserving the monitorability of the agreements established in a BN as it evolves, i.e. partners join or leave the network, or contracts are established, updated, revised, completed or dropped.

First, we introduce a meta-model for contract-based business networks and their monitorability. Then, we show that the preservation of the monitorability of the BN relies on a combination of (i) contract replication (projection), to maintain the correct set of dependencies among the expectations of the partners in the BN, and (ii) the update of the monitoring infrastructure, which has to the match the new monitoring requirements introduced by the BN evolution. Additionally, we also propose a set of metrics for quantifying the monitorability of the contracts established in the BN. The monitorability metrics serve as a further criterion to evaluate the opportunity, for the whole BN or for specific partners, to make the BN evolve, i.e. substituting a partner or outsourcing a business process.

An industrial case study and a prototype implementation demonstrate the applicability of the framework. The case study concerns an evolving BN in healthcare involving both aspects of synchronization and dynamic sourcing. Through the case study, we show how to use our framework to solve many problems related to collaboration that are known to arise in current healthcare practice. The prototype shows the technical feasibility of the framework adopting Web service technology and how the framework can be embedded within different business process management and execution infrastructures.

This paper is organized as follows. Section 2 discusses the lifecycle of agreements in BNs and introduces a meta-model for BNs and for the evolution of the contracts that regulate the business relationship among the BN partners. Section 3 presents the mechanisms for preserving the monitorability of a BN in the presence of BN evolution. The metrics to quantify the monitorability of a BN are given in Section 4. The case study is discussed in Section 5, while the prototype implementation demonstrating the case study is presented in Section 6. Eventually, the paper ends with a discussion on related work in Section 7 and conclusions and outlining future work in Section 8.

2. Business Networks

To contextualize our framework, this section first discusses the lifecycle of agreements in BNs where evolution can occur. Then, we introduce a meta-model of evolving business networks, which is later used to exemplify the mechanisms for preserving the monitorability of agreements as a BN evolves.

2.1. The lifecycle of agreements in evolving Business Networks

We make the assumption that the collaboration within a BN is template-based and regulated by a set of bilateral agreements, established between a process provider and a process consumer. Template-based advertisement of providers' capabilities and establishment of agreements is a typical contracting mechanism [5, 37, 54], considered also by the service computing research community [17, 50, 53]. It is assumed, for

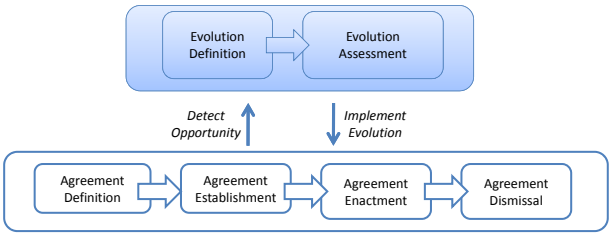

Figure 1: Extended Agreement lifecycle of business networks

instance, by the WS-Agreement [75] specification, a *de facto* standard for the definition of agreements for the usage of Web services.

Our proposed lifecycle of agreements in a BN extends the agreement lifecycle commonly envisioned in the service-oriented computing research community [53]. The typical lifecycle of an agreement (or contract) within a BN is shown in white shapes in Figure 1. It involves the phases of agreement definition, establishment, enactment, and dismissal. In the agreement definition, the parties agree on the terms on which the agreement can be established, e.g. parties may agree to include specific terms into agreement templates, such as terms on the completion time of activities. This phase leads to the definition of new agreement templates, or the update of existing ones. Then, during agreement establishment, the parties reach an agreement on the terms according to which a process will be provisioned, e.g. parties negotiate the actual completion time, expressed in days or hours, of the activities in the process. This phase results in the signing of an actual agreement, which instantiates and completes an agreement template. Then, the agreement is enacted, i.e. the process on which the agreement has been reached is executed. Eventually, the agreement can be dismissed. This can occur when a partner leaves the network, when an agreement is revised, and therefore substituted by an updated version, when one of the parties (or both) decide to terminate the collaboration, or simply when a contract expires or the obligations specified in it have been fulfilled.

In the context of evolving BNs, besides the agreement lifecycle, we introduce the *evolution lifecycle*, represented in shaded shapes in Figure 1. While the BN is in operation, business actors in the BN may realize that the network efficiency and/or effectiveness can be increased by changing the BN in some of its aspects, i.e. if new agreements are introduced or if existing agreements are updated or dropped from the BN. This phase is represented by the *Evolution Definition* phase and it leads to the definition of a set of changes (or, as later defined, *evolution types*).

The implementation of an evolution type, e.g. the substitution of a partner, may have several impacts on the BN such as the redefinition of existing contracts or the adaptation of existing protocols [61]. In this paper, we focus specifically on the impact on monitorability of existing and new agreements in the BN. More in detail, an evolution type may disrupt the monitorability of existing agreements and agreement templates. In the case of partner substitution, for instance, the new partner may not be able to provide monitoring information as its predecessor and, therefore, the agreements with the new partner should be modified accordingly.

Hence, similarly to Business Process Reengineering (BPR) initiatives [44], the envisioned evolution types must be assessed (see *Evolution Assessment* phase), to understand which changes are beneficial from the monitorability perspective, and therefore worth implementing, and which changes, on the contrary, are not beneficial, leading to a decreased monitorability of the BN. As for BPR initiatives, where potential change is assessed against the key performance indicators of the reengineered process, the evolution assessment relies on a set of monitorability metrics, which we describe in Section 4.

The *Evolution Assessment* phase objective is twofold. On the one hand, it assesses the actions that should be undertaken to preserve (or improve, if possible) the monitorability of the BN. On the other hand, it evaluates the monitorability metrics in the to-be situation, i.e. the one in which such actions are applied in the BN. The output of this phase, therefore, is a set of actions that must be undertaken to restore the monitorability of the BN for each evolution type that has been positively assessed. Eventually, the evolution

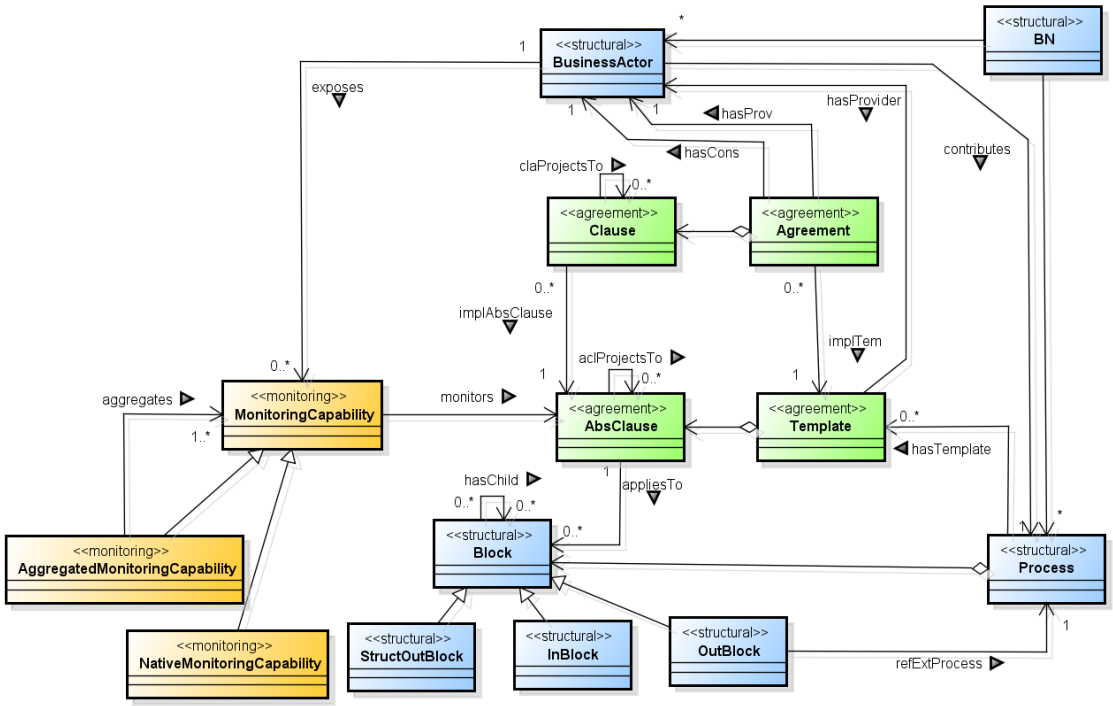

Figure 2: UML Class Diagram of business networks metamodel

types will be implemented within the BN, e.g. through the establishment of new templates and contracts, or the update of the monitoring infrastucture of the actors constituting the BN.

2.2. A meta-model for Business Networks

The meta-model of BNs considered within our framework is shown in Figure 2. The elements of a BN are partitioned into three different groups, or *overlays*, i.e. the *structural*, *agreement*, and *monitoring* overlays. Overlays are identified in Figure 2 by the corresponding stereotypes.

2.2.1. Structural overlay

The structural overlay defines the BN in terms of the business actors participating in it and the processes they contribute to it. A business actor can be the provider and/or the consumer (customer, user) of one or more business processes. Considering the three level process framework introduced in [29], we deal with processes specified using the *public view*. The public view of a process retains those aspects that are relevant for the collaboration among partners in a BN and hides the internal detailed specification required for process enactment by individual partners (specified in the *private* view).

We consider single-entry, single-exit, block-structured processes [25, 41]. Hence, a process is constituted by a set of blocks. As it will be discussed later in this section, blocks are the process decomposition unit to which clauses constituting agreements are associated. Blocks can be either atomic or structured. A structured block has one or more children, i.e. sub-blocks (note the self-relation hasChild, which introduces a hierarchical relation among blocks in a process). Blocks may be internal (*InBlock* in Figure 2), (atomic) outsourced (*OutBlock*), or structured-outsourced (*StructOutBlock*).

Internal blocks can be either atomic or structured, and are executed by the business actor that performs the business process to which the blocks belong. Outsourced blocks are delegated by the business actor contributing the business process to a different business actor, i.e. a provider, in the BN. Hence, an outsourced block refers to an external process contributed to the BN by a different businss actor (see the relation refExtProcess). Structured-outsourced blocks are special blocks for which at least one of the children is an

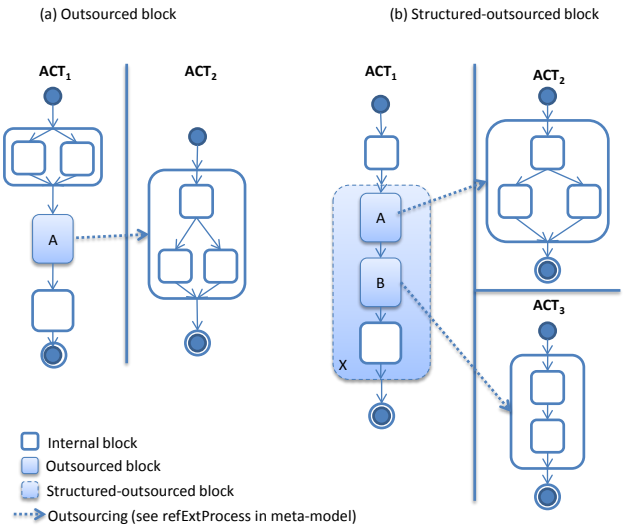

Figure 3: Business process outsourcing in business networks

outsourced block or a structured-outsourced block again. Structured-outsourced blocks allow the definition of agreements on blocks for which children can be outsourced to different providers. Their purpose will be further explained later in this section.

We also consider the following constraints on outsourcing, which are not explicitly captured by the UML class diagram of Figure 2:

1. An internal block cannot have outsourced or structured-outsourced children;
2. Outsourced block can only be atomic, i.e. they cannot have children, and they are outsourced to at most one provider (who may then outsource again).

Figure 3 illustrates the possible structural overlays, in example BNs, allowed by our model using a UML Activity diagram-like notation[1]. Figure 3a shows the outsourcing of the outsourced block A from ACT_1 to ACT_2. Figure 3b exemplifies a structured-outsourced block (X). In particular, X has one internal child block and two outsourced child blocks A and B that are outsourced to two different providers, i.e. ACT_2 and ACT_3, respectively. Note that only blocks A and B refer to a corresponding external process, while the structured-outsourced block itself (i.e. X) is not outsourced and therefore does not refer to an external process. Note also that the framework is not concerned with the blocks internal control structure. The reason for this will be discussed later on, while introducing the monitoring overlay.

2.2.2. Agreement Overlay

The agreement overlay sits on top of the structural overlay, modeling the agreements among partners in the BN.

Providers advertise agreement templates associated to the processes they contribute to the BN. Templates contain the customizable terms under which an agreement can be established between a provider and a consumer. Templates are not enforceable and are used by consumers to look for potential partners in the BN. Consumers and providers establish agreements for the usage of a process contributed by a provider to the BN. Agreements are enforceable and result from the customization of corresponding templates. Such customization process is usually a negotiation between the provider and consumer, leading to an agreement

[1]We do not make an explicit choice on a business process modeling language, since processes in our framework can be expressed using any language that allows hierarchy among activities and block-structuredness, e.g. UML Activity Diagrams and BPMN [49].

on the terms associated with the process provisioning, which are enforceable during the agreement lifespan, of which the process enactment itself is a large part (see *Agreement Establishment* phase in Figure 1).

Besides other information, which is not of interest for the purpose of this paper (e.g. the description of partners or a human-readable specification of the agreement [5, 54]), templates are constituted by a set of abstract clauses. Abstract clauses represent customizable constraints on the enactment of the process and they refer to blocks of the process. Agreements are constituted by clauses, implementing corresponding abstract clauses. A clause represents a customized constraint on the usage of a process and it is enforceable. For instance, insurance companies may outsource the loss adjustment phase of their own claim management process to a risk manager. A template advertised by a specific risk manager (provider) can allow insurance companies (consumers) to set the number of different loss adjusters, i.e. experts required to provide an estimate for a given claim. A corresponding clause, which derives from a negotiation, can state that the loss adjusters will be at least two, providing two separate evaluations for any given claim.

It is worth noticing that our framework does not consider the semantics of abstract clauses and clauses. Common languages for expressing electronic contracts and SLAs, e.g. WS-Agreement [75] or ebXML [48] take a similar perspective defining a container structure for expressing service and SLA terms and leaving the specification of those to domain specific languages. Work in [17, 16], for instance, considers a domain specific language to express performance-related QoS of enterprise systems which is used within WS-Agreement contract specifications. Domain specific languages should also embed the semantic of contract clauses, e.g. by referring to domain specific ontologies. However, a generic contract management framework should abstract from domain specific aspects and delegate the management of semantics to a domain specific functionality.

After having introduced clauses and abstract clauses, we can now refine the conceptual motivation for having structured outsourced blocks in our meta-model. Abstract clauses in templates (and, consequently, clauses in agreements) are defined over blocks, which can be atomic or structured. An (abstract) clause concerning a structured block may *not* be equivalent to the combination of (abstract) clauses over its children.

In the claim management scenario, introduced before, the insurance company may define abstract clauses for its prospective customers referring to the damage loss evaluation. The damage loss evaluation process may have the following structure: the loss adjustment activity is outsourced to a risk manager, the fraud check is outsourced to an external 3rd party auditor, and the company itself makes some internal processing to integrate the results of these two activities. An abstract clause may state that the insurance company can make available information on the internal progress of the damage loss evaluation (this information can be used by consumers, for instance, to synchronize their own claim management internal process). Since part of the block to which the abstract clause refers is outsourced, however, the insurance company is required to establish two new individual abstract clauses, with the risk manager (to get progress information on the loss adjustment) and the auditor (to get progress information on fraud check). The insurance company can then integrate the information obtained from outsourcers with its internal progress information to make progress information for damage loss evaluation available to the consumers. We can model this situation only introducing the structured outsourced block. The structural and agreement overlays of the insurance company will therefore include a structured outsourced block (damage loss evaluation) which has two outsourced blocks as children, i.e. the loss adjustment, outsourced to the risk manager, and the fraud check, outsourced to the auditor, and an internal block, i.e. the internal processing for integrating results. Note that this scenario is also modeled by the process in Figure 3b when considering X as the damage loss evaluation block (structured-outsourced), A as the loss adjustment and B as the fraud check (both outsourced), and the remaining internally executed block as the processing required by the insurance company to integrate the results of the two preceding activities.

We define the association projectsTo between abstract clauses of different templates to capture the contract dependency introduced by process outsourcing. In case of an outsourced block, an abstract clause in a template referring to an outsourced block projects to an abstract clause in a template advertised for the process to which the block is outsourced. In case of structured-outsourced blocks, an abstract clause can be projected onto a set of abstract clauses referring to templates of the processes to which the children of the block have been outsourced.

2.2.3. Monitoring Overlay

The monitoring overlay sits on top of the agreement overlay, modeling the monitorability requirements of the agreements established in the BN. In order to be monitored, in fact, an abstract clause induces monitoring information requirements of the consumer, i.e. information that the consumer must obtain from the provider to monitor a clause derived from such abstract clause. In the insurance claim management scenario, for instance, a consumer requires information on the identity of the damage loss evaluators to monitor a clause that states that two different experts are required to perform the evaluation of a given claim.

We define a monitoring capability as the ability of a provider to provide monitoring information, matching the information requirements of consumers to monitor the clauses in agreements. Note that monitoring capabilities are associated to abstract clauses (see the monitors relation in Figure 2). Therefore, a monitoring capability is used to monitor a clause that implements the abstract clause to which the monitoring capability refers. From a technical standpoint, a monitoring capability is a point of access for the consumer to the monitoring information made available by the provider to monitor a given abstract clause. In our implementation (see Section 6) the provider exposes a single Web service operation for accessing its monitoring capabilities. Consumers use a parameterized Web service call to access the right monitoring capability and to obtain information to monitor a specific clause.

Monitoring capabilities enable the monitoring of clauses derived from abstract clauses, which are associated to process blocks. The ability of a monitoring capability to make available the right monitoring information to consumers does not depend on the specific path followed by the enactment (execution) of a process block. In other words, monitoring capabilities in our framework are associated to process specifications, rather than process instances. This is the reason why our process model in the structural overlay does not take into account the internal control structure of blocks. We argue that the provider could optimize resource usage, for instance, by not guaranteeing monitoring for abstract clauses associated to blocks belonging to branches that are only very seldom executed or by devoting more resources to guarantee monitoring information on a critical loop structure that must be executed multiple times in every process instance. However, we consider the aforementioned issues out of scope in this paper and we leave the discussion of efficient resource allocation for monitoring to future work.

Monitoring capabilities are either *native* or *aggregated*. A *native* monitoring capability is used to retrieve information to monitor a clause implementing an abstract clause that refers to an internal block. In other words, monitoring information made available through a native monitoring capability can be captured by the provider itself, within its own business domain. An *aggregated* monitoring capability is used to monitor clauses referring to outsourced blocks or structured-outsourced blocks. The provider builds aggregated monitoring capabilities as the aggregation of internally captured information and/or information obtained through the monitoring capabilities exposed by the providers of the processes to which the corresponding block is outsourced. Note that, in case of abstract clauses referring to structured-outsourced blocks, the aggregate relation may link a monitoring capability to a set of monitoring capabilities, i.e one for each children of the structured-outsourced block that is outsourced to a different provider.

It has to be noticed that our framework does not impose any specific constraint on the language for expressing (abstract) clauses, such as the supported arithmetical or logical operators. The framework considers (abstract) clauses only to the extent to which they induce monitoring requirements, which are satisfied by corresponding monitoring capabilities. How the monitoring capabilities acquire the data or what data is necessary to satisfy the monitoring requirements, and consequently satisfy the clauses in the agreements is determined (and encapsulated) by the provider and is considered out of scope for this paper. As an example, the previously mentioned abstract clause between the insurance company and the risk manager on the number of loss adjusters requires a language in which it is possible to express the "min" operator, but what is important for our framework are the induced monitoring requirements, i.e. the risk manager should make the identity of the loss adjusters available to the insurance company. Similarly, an abstract clause allowing the consumer to monitor the intermediate progress of a request is identified by its induced monitoring requirements, i.e. the provider should make intermediate progress information available to its consumers.

Table 1: Set-theoretic representation of the Structural overlay meta-model.

Set	Notation	Related Predicates
Business Network	$BN = \langle ACT, PRO, TEM, AGR, MCP \rangle$	
Business Actors	$ACT = \{act_a\}_{a=1,\dots,A}$	
Processes	$PRO = \{pro_p\}_{p=1,\dots,P}$	$contribute(act_a, pro_p)$
Blocks (activities)	$BLK_p = \left\{blk_{p,b_p}\right\}_{b_p=1,\dots,B_p}$ $BLK = \bigcup_{p=1}^{P} BLK_p$	$hasChild\left(blk_{p,b_p}, blk_{p,c_p}\right) \wedge 1 \leq b_p, c_p \leq B_p, b_p \neq c_p$ $refExtProcess\left(blk_{p,b_p}, pro_q\right) \wedge p \neq q.$

Table 2: Set-theoretic representation of the Agreement and Monitoring overlays meta-model.

Set	Notation	Related Predicates	Comments
Templates	$TEM = \{tem_t\}_{t=1,\dots,T}$	$hasProvider\,(tem_t, act_a)$ $hasTemplate\,(pro_p, tem_t)$	act_a advertises the template tem_t, which refers to process pro_p
Abstract Clauses	$ACL_t = \left\{acl_{t,l_t}\right\}_{l_t=1,\dots,L_t}$ $ACL = \bigcup_{t=1}^{T} ACL_t$	$appliesTo\left(acl_{t,l_t}, blk_{p,b_p}\right)$ $aclProjectsTo\left(acl_{t,l_t}, acl_{s,l_s}\right) \wedge t \neq s$	Abstract clause acl_{s,l_s} is a projection of acl_{t,l_t}
Agreements	$AGR = \{agr_g\}_{g=1,\dots,G}$	$implTem\,(agr_g, tem_t)$ $hasCons(agr_g, act_a)$ $hasProv(agr_g, act_b)$	The agreement agr_g implements the template tem_t, and has act_a as a consumer and act_b as provider.
Clauses	$CLA_g = \left\{cla_{g,c_g}\right\}_{c_g=1,\dots,C_g}$ $CLA = \bigcup_{g=1}^{G} CLA_g$	$implAcl\left(cla_{g,c_g}, acl_{t,l_t}\right)$ $claProjectsTo\left(cla_{g,c_g}, cla_{f,c_f}\right) \wedge g \neq f$	Clause implement abstract clauses. The clause cla_{f,c_f} is a projection of the clause cla_{g,c_g}
Monitoring Capabilities	$MCP = \{mcp_m\}_{m=1,\dots,M}$	$exposes\,(act_a, mcp_m)$ $monitors\,(mcp_m, acl_{t,l_t})$ $aggregates\,(mcp_m, mcp_1, \cdots, mcp_J)$	A monitoring capability mcp_m aggregates one or more capabilities mcp_j, with $1 \leq j \leq J$.

2.2.4. Set-theoretic representation of business networks

From the meta-model of Figure 2, we derive the set-theoretic representation of BN overlays shown in Table 1 and Table 2 for the structural and the agreement and monitoring overlays, respectively. The set-theoretic representation is required to give a clear specification of the algorithms for the preservation of the monitorability of the BN presented in Section 3.

Classes in the meta-model are represented by sets, e.g. business actors or templates, whereas predicates capture the relations among classes. Predicates take as arguments elements of the corresponding sets. Note that, for the sake of clarity and conciseness of the notation, we do not use predicates for aggregation relations, but we capture aggregation relations using subscripts of the elements of a set (see, as a sample, the aggregation of abstract clauses l_t, with $l_t = 1, \dots, L_t$ in a template tem_t in Table 2).

The type of a block is a value bt, with $bt \in \{I, O, SO\}$, for, respectively, Internal, Outsourced, and Structured-Outsourced blocks. The type of monitoring capabilities is a value mt, with $mt \in \{N, A\}$, for, respectively, native and aggregated capabilities. The block and monitoring capability type can be obtained through the corresponding functions $blockType()$ and $mcpType()$, with:

$$blockType : BLK \mapsto \{I, O, SO\}$$
$$mcpType : MCP \mapsto \{N, A\}$$

Table 3: Evolution types and related acronyms.

Element	Action	Acronym
Actor	ADD	ADD-ACT
	REMOVE	REM-ACT
Process	ADD	ADD-PRO
	REMOVE	REM-PRO
Abstract Clause	ADD	ADD-ACL
	REMOVE	REM-ACL
Template	ADD	ADD-TEM
	REMOVE	REM-TEM
Clause	ADD	ADD-CLA
	REMOVE	REM-CLA
Agreement	ADD	ADD-AGR
	REMOVE	REM-AGR

For the sake of clarity, we will also express the type of an element using a superscript. For instance, blk^{SO}_{p,b_p} represents a structured-outsourced block.

The modelling of the structural overlay is complemented by the definition of the function *children*(), which takes as argument a block and returns the set of its children, i.e. sub-blocks:

$$children : BLK \mapsto \mathcal{P}BLK$$
$$children\left(blk_{p,b_p}\right) = \left\{blk \in BLK_p : hasChild\left(blk_{p,b_p}, blk\right)\right\}$$

Note that the function *children*() returns a set when called on structured-outsourced blocks and a single element when called on outsourced blocks.

3. Evolution and Monitorability of Business Networks

Evolution of a business network may occur at the structural and agreement overlay levels, e.g. actors may join or leave a BN and templates or agreements within the BN may be added or dropped.

More specifically, we classify evolution in a BN into different types according to the element of the BN that will be modified, i.e. evolution of *actors*, *processes*, *clauses*, *agreements*, *abstract clauses*, and *templates*, and the action to be taken on the considered element, i.e. *add* or *remove* to/from the BN (see Table 3 for the evolution types acronyms). Note that the *update* of an element, e.g. the update of an agreement, is modeled as the sequential combination of removing the current element and adding a new version of it.

For each evolution type in Table 3, we first list the corresponding pre- and post-conditions. Pre-conditions should be verified in order for the evolution type to take place, whereas post-conditions describe the status of the BN after the evolution type has occurred. Then, we discuss mechanisms that define how the BN should react to the application of the evolution type. An evolution type, in fact, may lead the BN in a state characterized by different types of inconsistencies. The mechanisms aim at guaranteeing the monitorability of the elements that are added to the BN and preserving the monitorability of the elements already in use in the BN. If mechanisms are not run, in fact, several inconsistencies in the BN may remain. Examples of possible inconsistencies are abstract clauses that are not monitorable and, consequently, clauses that cannot be monitored by consumers, templates and contracts that do not respect the correct dependencies, or clauses linked to wrong monitoring capabilities, that is, capabilities that monitor the wrong abstract clause.

3.1. Structural evolution of a BN

Table 4 shows the pre- and post-conditions for the evolution types of the structural overlay of a BN. Note that we consider the addition or removal of the generic actor act_a and process pro_p and that we express pre-

Table 4: Pre- and post-conditions for Structural overlay evolution types.

Evolution type	**Pre-conditions**	**Post-conditions**
ADD-ACT(act_{A+1})	$act_{A+1} \notin ACT$	$ACT^{to-be} = ACT \cup \{act_{A+1}\}$
REM-ACT(act_a)	$act_a \in ACT \wedge$ $\{pro \in PRO : contribute(act_a, pro)\} = \emptyset$	$ACT^{to-be} = ACT \setminus \{act_a\}$
ADD-PRO(pro_{P+1}, act_a)	$act_a \in ACT \wedge$ $pro_{P+1} \notin PRO$	$PRO^{to-be} = PRO \cup \{pro_{P+1}\} \wedge$ $contribute(act_a, pro_{P+1})$
REM-PRO(pro_p)	$pro_p \in PRO \wedge$ $\{tem \in TEM : hasTemplate(pro_p, tem)\} = \emptyset$	$PRO^{to-be} = PRO \setminus \{pro_p\}$

Table 5: Actions for preserving monitorability.

Name	**Action**
Abstract Clause Projection	$acl_{s,L_s+1} \leftarrow ProjectAcl(acl_{t,l_t}, tem_s)$
Clause Projection	$cla_{f,C_f+1} \leftarrow ProjectCla(cla_{g,c_g}, agr_f)$
Native Monitoring Capability Creation	$mcp^N_{M+1} \leftarrow CreateNatMcp(act_a, acl_{t,l_t})$
Aggregated Monitoring Capability Creation	$mcp^A_{M+1} \leftarrow CreateAggMcp(act_a, acl_{t,l_t}, mcp_{m_j}\|_{j=1,\ldots,J})$
Internal re-sourcing of process block	$blk^I_{p,b_p} \leftarrow ReSource(blk^O_{p,b_p})$

and post-conditions comparing the sets constituting the BN in the current situation in which the evolution type has not yet been applied, i.e. the *as-is* situation, with the *to-be* situation, i.e. the one in which the evolution type has been applied. The post-conditions also contain the predicates becoming true after the implementation of the evolution type (see, as a sample, the predicate *contributes*(), which evaluates to true after the evolution type ADD-PRO has been applied).

Concerning Table 4, the pre-condition for the introduction of new actors ensures that the actor to be added does not exist yet in the BN. New processes can be introduced only by actors that already belong to the BN. As post-condition, the new actor or process will be added to the corresponding actor or process set of the BN. Only existing actors and processes can be removed from the BN. Additional pre-conditions state that a process can only be removed if no templates referring to that process exist and an actor can only be removed if it does not contribute a process to the BN.

From the point of view of the monitorability of the BN, introducing or removing actors and processes do not modify the monitoring requirements of the actors belonging to the BN. Therefore, when structural evolution occurs, no specific actions have to be taken to preserve the monitorability of the BN.

3.2. Actions to support monitorability

Before discussing the mechanism for reacting to evolution at the agreement overlay level, i.e. abstract clause- and template-level evolution (in Section 3.3) and clause- and agreement-level evolution (in Section 3.4), we describe the set of standard *actions* that may need to be undertaken by actors to preserve the monitorability of the BN (see Table 5). Actions may concern (i) the projection of clauses or abstract clauses made by a provider towards agreements or templates of external providers, (ii) the creation of native or aggregated monitoring capabilities, and (iii) the internal sourcing of an outsourced block by an actor. Note that (i) concerns the maintenance of correct dependencies among agreements established in the BN, (ii) concerns the update of the monitoring infrastructure of the actors belonging to the BN, and (iii) concerns the reconfiguration of the BN process in case a service provider is removed from the BN.

The projection of clauses is required, for instance, when new (abstract) clauses referring to a process block that is outsourced by a provider are introduced. In this case, in fact, the provider should project the

new clause towards the actors to which the block (or part of it) is outsourced, in order to be able to reduce the risk of not being able to deliver what is promised to its consumers.

Projections may concern either abstract clauses or clauses. Thus, we define two different projection actions:

- $acl_{r,L_r+1} \leftarrow ProjectAcl(acl_{t,l_t}, tem_r)$, which projects the abstract clause acl_{t,l_t} into a new abstract clause acl_{r,L_r+1} belonging to the template tem_r; Besides the creation of the new abstract clause, the projection also creates the link, made by the provider of the template tem_r, between the new abstract clause and one of the blocks in the process pro_q to which the template tem_r refers, say blk_{q,b_q}. In other words, from the modeling perspective, after the execution of the action, the predicates $aclProjectsTo(acl_{t,l_t}, acl_{r,L_r+1})$ and $refersTo(acl_{r,L_r+1}, blk_{q,b_q})$ will evaluate to true.

- $cla_{f,C_f+1} \leftarrow ProjectCla(cla_{g,c_g}, agr_f)$, which projects the clause cla_{g,c_g}, which complies to an abstract clause, say acl_{t,l_t}, into a new clause cla_{f,C_f+1} belonging to the agreement agr_f, which complies to an abstract clause acl_{r,l_r}. Note that the projection can be executed only if the predicate $projectsTo(acl_{t,l_t}, acl_{r,l_r})$ evaluates to true. Furthermore, as for the projection of abstract clauses, after the execution of the action the predicate $claProjectsTo(acl_{t,l_t}, acl_{r,L_r+1})$ will evaluate to true.

For the structured outsourcing case, when a (abstract) clause referring to a structured-outsourced block needs to be projected, the projection results in a set of projections actions, one for each external actor to which the execution of parts of the children blocks is delegated. For example, as previously indicated, if an insurance company outsources part of its damage loss evaluation to a risk manager (loss adjustment) and a 3rd party auditor (fraud check), then an abstract clause allowing customers of the insurance company to monitor the intermediate progress of the damage loss evaluation should be projected into two separate clause, i.e. a clause in the agreement with (and consequently an abstract clause in the template defined by) the risk manager, allowing to monitor the intermediate states of the loss adjustment, and a clause in the agreement with (and an abstract clause in the template defined by) the auditor, allowing to monitor the intermediate states of the fraud check.

Also, as previously introduced, since our framework does not consider the semantic of (abstract) clauses, we make the hypothesis that projection can always occur always occur between semantically related clauses. The implementation of this requirement is left specific to the application domain at hand.

The creation of monitoring capabilities is required, for instance, when adding a new abstract clause. The creation of monitoring capabilities may involve only the provider of the block to which the abstract clause refers (native monitoring capability) or it may concern also the partners to which such block has been outsourced (aggregated monitoring capability). Hence, we define the following actions:

- $mcp^N_{M+1} \leftarrow CreateNatMcp(act_a, acl_{t,l_t})$, which returns a native monitoring capability mcp_{M+1} for allowing the monitoring of the abstract clause acl_{t,l_t}. The monitoring capability is created by the actor act_a. After the execution of the action, the predicate $monitors(mcp_{M+1}, acl_{t,l_t})$ will evaluate to true. The new capability mcp_{M+1} is exposed by the provider of the template to which the clause acl_{t,l_t} belongs;

- $mcp^A_{M+1} \leftarrow CreateAggMcp(act_a, acl_{t,l_t}, mcp_{m_1}, \cdots, mcp_{m_J})$, which returns an aggregated monitoring capability mcp_{M+1} for allowing the monitoring of the abstract clause acl_{t,l_t}. The monitoring capability is created by the actor act_a. The capability is obtained through the aggregation of the capabilities mcp_{m_j}, with $j = 1, \ldots, J$ and $J \leq M$. If the clause acl_{t,l_t} refers to an outsourced block blk, with $blockType(blk) = O$, then the aggregation will involve only one external capability, i.e. $J = 1$. If the block blk is structured-outsourced, i.e. $blockType(blk) = SO$, the aggregation may involve more than one external capability, i.e. one for each of the child blocks of blk that is outsourced to an external provider. After the execution of the action sets, the predicate $aggregates(mcp_{M+1}, mcp_{m_1}, \cdots, mcp_{m_J})$ will evaluate to true. The new capability mcp_{M+1} is exposed by the provider of the template to which the abstract clause acl_{t,l_t} belongs.

Table 6: Pre- and post-conditions for Abstract Clause- and Template-level evolution types.

Evolution type	Pre-conditions	Post-conditions
ADD-ACL(acl_{t,L_t+1}, blk_{p,b_p}, act_a)	$pro_p \in PRO \wedge tem_t \in TEM \wedge act_a \in ACT$ $hasProvider(tem_t, act_a) \wedge$ $hasTemplate(pro_p, tem_t) \wedge$ $acl_{t,L_t+1} \notin ACL_t$	$ACL_t^{to-be} = ACL_t \cup \{acl_{t,L_t+1}\} \wedge$ $appliesTo(acl_{t,L_t+1}, blk_{p,b_p})$
ADD-TEM(tem_{T+1}, pro_p, act_a)	$act_a \in ACT \wedge pro_p \in PRO \wedge$ $contributes(act_a, pro_p) \wedge$ $tem_{T+1} \notin TEM$	$TEM^{to-be} = TEM \cup \{tem_{T+1}\} \wedge$ $hasTemplate(pro_p, tem_{T+1}) \wedge$ $hasProvider(tem_{T+1}, act_a)$
REM-ACL(acl_{t,l_t})	$acl_{t,l_t} \in ACL_t$	$ACL_t^{to-be} = ACL_t \setminus \{acl_{t,l_t}\}$
REM-TEM(tem_t)	$tem_t \in TEM \wedge ACL_t = \emptyset$	$TEM^{to-be} = TEM \setminus \{tem_t\}$

Eventually, actors should be able to in-source a block of the processes that they contribute to the BN in order, for instance, to preserve the monitorability of the BN when an actor, to which part of a process is outsourced, leaves the BN.

Hence, we define the following action:

- $blk^I_{p,b_p} \leftarrow InSource(blk^O_{p,b_p})$, which changes the type of a block blk_{p,b_p} from outsourced, i.e. $blkType(blk_{p,b_p}) = O$, to internal i.e. $blkType(blk_{p,b_p}) = I$, signifying the in-sourcing made by the consumer of a block originally outsourced to an external provider. Note that only outsourced blocks blk, with $blockType(blk) = O$, can be used as arguments of this action.

Since we focus on the monitoring problem, we assume that the projection of clauses and abstract clauses and the internal sourcing of a block by a provider is always successful, i.e. the execution of the action always leads to the correct creation of the expected outcome. Conversely, the creation of monitoring capabilities may not be successful, i.e. a provider may not be able to create a mechanisms for providing monitoring information to the consumers or it may not be able to aggregate correctly the monitoring information obtained by its own providers. The actions $CreateNatMcp()$ and $CreateAggMcp()$ will return the conventional value null to signify the failure of the creation of a monitoring capability. Note that the outcome of the creation of a monitoring capability will have an impact on the value of the monitorability metrics for the evolving BN, e.g. failure to create a monitoring capability associated to an abstract clause can decrease the monitorability of the agreements with clauses derived from such an abstract clause.

3.3. Abstract clause- and Template-level evolution

The pre- and post conditions for *Abstract Clause-* and *Template-level* evolution types are shown in Table 6.

ADD-ACL evolution type. In this case, a new abstract clause acl_{t,L_t+1} is added to a template tem_t for which act_a is the provider. The abstract clause acl_{t,L_t+1} must be added to an existing template and, as post-condition, the abstract clause will be added to the set of existing abstract clauses and associated to a specific block blk_{p,b_p}, belonging to the process pro_p to which tem_t refers.

ADD-TEM evolution type. In this case, a new empty template tem_{T+1} referring to the process pro_p and for which the actor act_a is the provider is added to the BN. As post-condition, tem_{T+1} is added to the set of existing templates and associated to the process pro_p and provider act_a.

REM-ACL evolution type. In this case, an existing abstract clause acl_{t,l_t}, with $1 \leq l_t \leq L_t$ is removed from the BN. As a post condition, the abstract clause will be removed from the set ACL_t of abstract clauses belonging to the template tem_t.

REM-TEM evolution type. In this case, an existing empty template tem_t, with $1 \leq t \leq T$ is removed from the BN. As a post-condition, the template is removed from the set of existing templates.

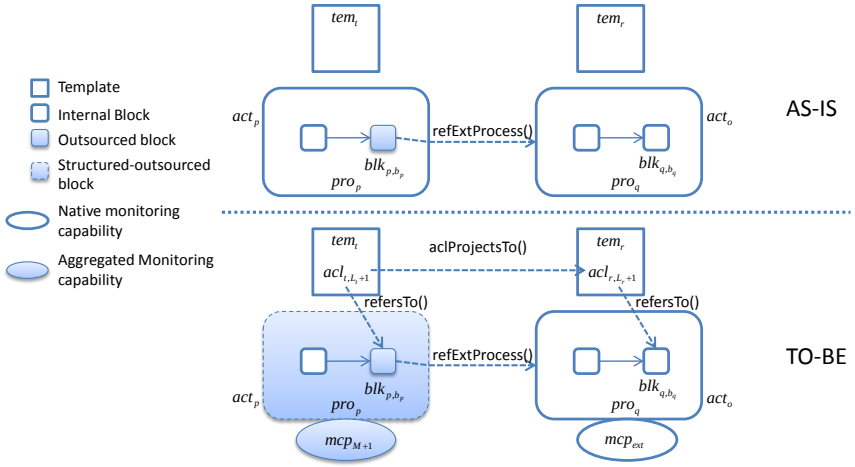

Figure 4: Preserving monitorability for ADD-ACL evolution type

Note that template-level evolution considers only empty templates. In real-world scenarios, we model the addition of a new template as the sequential combination of adding a new empty template and adding each new abstract clause that appears in it. Similarly, a template can be deleted only when it is empty, i.e. when all its abstract clauses have been previously deleted.

Algorithm 1 shows the mechanisms for preserving the monitorability of the BN in case of adding a new abstract clause to an existing agreement (ADD-ACL evolution type, see Figure 4 for a graphical representation).

In order to guarantee the monitorability of the template in which the new abstract clause is introduced, the provider act_p of the template should create a corresponding new monitoring capability. The results of the creation of the new monitoring capability may be either successful or not. In the former case, the new abstract clause will be monitorable, whereas in the latter case the new abstract clause could still be included in the template (for example to allow ex-post monitoring), but it will not be monitorable. Note that the creation of a monitoring capability is not successful when the value of the created monitoring capability is null.

The creation of the new monitoring capability is made by calling the $BuildCapability()$ procedure. Three cases may occur. In the first case (lines 2-3), the new abstract clause acl refers to a block that is internally executed by actor act. Consequently, actor act creates a new native monitoring capability.

In the second case (lines 4-10), the new abstract clause acl refers to an atomic outsourced block. In such situation, the outsourced block refers to an outsourced process pro_q executed by an external actor act_o. This outsourced process is governed by an agreement, which implements a specific template tem_r of that external actor. The new abstract clause should be projected to that specific template (see line 5-6)[2]. After the projection of the abstract clause, the external actor should create a suitable monitoring capability. If the creation of such capability is successful, then act will create an aggregated monitoring capability for the newly introduced abstract clause. Note that this process is recursive (see recursive call on line 9), i.e. it is iterated until the projected clause refers to a block executed internally by the external actor.

In the third case (lines 11-26) the new abstract clause acl refers to a structured outsourced block. In this situation, the mechanism for preserving the monitorability of the BN consists of recursively building monitoring capabilities and projecting the abstract clause if required for each of the child blocks of the structured outsourced block, after which those monitoring capabilities are aggregated by actor act. Note that the creation of monitoring capability by act can be successful only if all the providers to which the child blocks are outsourced are able to create their own monitoring capability and native monitoring capabilities

[2]We assume that only one agreement exists per outsourced process and argue that in situations where an outsourced process could be governed by multiple agreements, these can be combined into one agreement containing conditional statements.

Algorithm 1 Restoring monitorability for ADD-ACL evolution type

1: **procedure** ADD-ACL($acl_{t,L_t+1}, blk_{p,b_p}, act_p$)
2: $mcp_{M+1} \leftarrow$ BuildCapability($acl_{t,L_t+1}, blk_{p,b_p}, act_p$)
3: **end procedure**

1: **procedure** BuildCapability(acl, blk, act)
2: **if** $blockType\,(blk) = \{I\}$ **then** {block is internal}
3: $mcp \leftarrow CreateNatMcp(act, acl)$
4: **else if** $blockType\,(blk) = \{O\}$ **then** {block is outsourced}
5: $tem_r \leftarrow \{tem \in TEM : hasTemplate(pro_q, tem) \wedge refExtProcess(blk, pro_q) \wedge implTem(agr, tem) \wedge hasConsumer(agr, act)\}$ {find the template implemented by the agreement that governs the outsourced process pro_q, referenced to by block blk}
6: $acl_{r,L_r+1} \leftarrow ProjectAcl(acl, tem_r)$ { Project the abstract clause}
7: $act_o \leftarrow \{act \in ACT : hasProvider(tem_r, act)\}$
8: $blk_{q,b_q} \leftarrow \{blk \in BLK_q : appliesTo(acl_{r,L_r+1}, blk)\}$
9: $mcp_{ext} \leftarrow BuildCapability(acl_{r,L_r+1}, blk_{q,b_q}, act_o)$ { recursive call to external actor}
10: $mcp \leftarrow CreateAggMcp(act, acl_{t,L_t+1}, mcp_{ext})$
11: **else** {block is structured outsourced}
12: $MCP_{agg} \leftarrow \emptyset$; $agg \leftarrow true$ {define set of monitoring capabilities to aggregate}
13: **for all** $blk_{ch} : blk_{ch} \in children(blk)$ **do**
14: $mcp_{ch} \leftarrow BuildCapability(acl, blk_{ch}, act)$
15: **if** $mcp_{ch} =$ null **then**
16: $agg \leftarrow false$; **break** {depth-first scan of the outsourcing tree}
17: **else**
18: $MCP_{agg} \leftarrow MCP_{agg} \cup mcp_{ch}$
19: **end if**
20: **end for**
21: **if** agg **then**
22: $mcp \leftarrow CreateAggMcp(act, acl, MCP_{agg})$ {Create aggregated monitoring capability}
23: **else**
24: $mcp \leftarrow$ null
25: **end if**
26: **end if**
27: **return** mcp
28: **end procedure**

can be created for all child blocks that are executed internally. The mechanism adopts a depth-first scan of the outsourcing *tree*, i.e. the mechanism stops (see line 16) when a provider is not able to provide the native monitoring capability required by its consumer for aggregation. Depth-first analysis is more efficient than breadth-first analysis in this case, since the mechanism stops as soon as one of the capability required by the actor *act* for aggregation cannot be created or when *act* cannot create a native monitoring capability for internal blocks.

The time complexity of Algorithm 1 is $O(n)$, where n is the number of children blocks of blk_{p,b_p}, under the hypothesis that the non-recursive part of procedure $BuildCapability()$ executes in constant time. From a scalability point of view, the time complexity of $BuildCapability()$ is determined by the time required to physically create or to instantiate a new monitoring capability (line 22), which in many cases can be a manual activity.

Since, as previously discussed, we consider empty templates, adding a new template in the BN (ADD-TEM evolution type) does not represent a critical evolution type from the point of view of monitorability, i.e. no specific action should be undertaken to preserve the monitorability of the BN.

Algorithm 2 shows the algorithm for preserving the monitorability of the BN in case of deletion of an

Algorithm 2 Preserving monitorability for REM-ACL evolution type

1: **procedure** REM-ACL(acl_{t,l_t})
2: **if** $\nexists acl \in ACL : projectsTo(acl, acl_{t,l_t})$ **then**
3: $ACL_t \leftarrow ACL_t \setminus \{acl_{t,l_t}\}$
4: **end if**
5: **end procedure**

Algorithm 3 Preserving monitorability for REM-TEM evolution type

1: **procedure** REM-TEM(tem_t)
2: **if** $ACL_t = \emptyset$ **then**
3: $TEM \leftarrow TEM \setminus \{tem_t\}$ {Remove the template}
4: **else**
5: $pro_p \leftarrow \{pro \in PRO : hasTemplate(pro, tem_t)\}$
6: $act_p \leftarrow \{act \in ACT : contributes(act, pro_p)\}$
7: **for all** $acl_{t,l_t} \in TEM_t$ **do**
8: **for all** $acl_{s,l_s} \in ACL : projectsTo(acl_{s,l_s}, acl_{t,l_t})$ **do**
9: $blk^O_{c,b_c} \leftarrow \{blk \in BLK_q : appliesTo(acl_{s,l_s}, blk)\}$
10: $blk^I_{c,b_c} \leftarrow InSource(blk^O_{c,b_c}, pro_p)$ {in-source the external process}
11: $act_c \leftarrow \{act \in ACT : hasProvider(tem_s, act)\}$
12: $mcp_{M+1} \leftarrow BuildCapability(acl_{s,l_s}, act_c, blk_{c,b_c}, pro_c)$; {Build the new native capability}
13: **end for**
14: **end for**
15: $TEM \leftarrow TEM \setminus \{tem_t\}$ {Remove the template}
16: **end if**
17: **end procedure**

abstract clause from an existing template (REM-ACL evolution type). The abstract clause acl_{t,l_t} to be deleted (see Figure 5) may either be a projection of one or more clauses acl_{s,m_s} in templates published by act_p's consumers, or not. In the former case (see conditional statement on line 2), the abstract clause cannot be deleted straight away, but can be deleted only when the template to which it belongs is deleted (this case is discussed later in this section). The deletion of the projected abstract clause acl_{t,l_t}, in fact, would result in an incorrect dependency between future agreements, since act_c will not be able to project clauses of an agreement with its own consumer into the agreement with act_p, because the corresponding abstract clause acl_{t,l_t} has been deleted. In the latter case (acl_{t,l_t} is not a projection), the abstract clause can be simply removed from the template (line 3).

When an abstract clause is deleted, the corresponding monitoring capability that monitors it should not be deleted. Agreements can still exist in the BN that include clauses that implement the abstract clause that is deleted, i.e. the deletion of an abstract clause does not lead to the deletion of the corresponding clause in agreements. Consumers of agreements in the BN may still need to access the capability for monitoring those clauses (for which the corresponding abstract clause is deleted). This principles draws from the literature on dynamic workflow evolution, where evolution of a process should maintain consistency not only for the new process instances, but also for instances that have not terminated by the time the evolution occurs [12, 60, 19]. Note that garbage collection will also be needed when, for instance, a monitoring capability is no longer used by any consumer and the provider wants to free the resources allocated to the implementation of such capability. This considerations are out of scope in this paper.

For what concerns deleting an existing template, the algorithm for preserving monitorability for REM-TEM evolution type is reported in Algorithm 3. If the template tem_t is empty, then it can be removed from the BN (lines 2-4). If the template tem_t is not empty, then it contains one or more abstract clauses that are the projection of other clauses acl_{s,l_s} (see Figure 5). For each of these clauses, the corresponding consumer, e.g. act_c in Figure 5, needs first to in-source internally the block to which the abstract clause $acl_{s,m}$ refers,

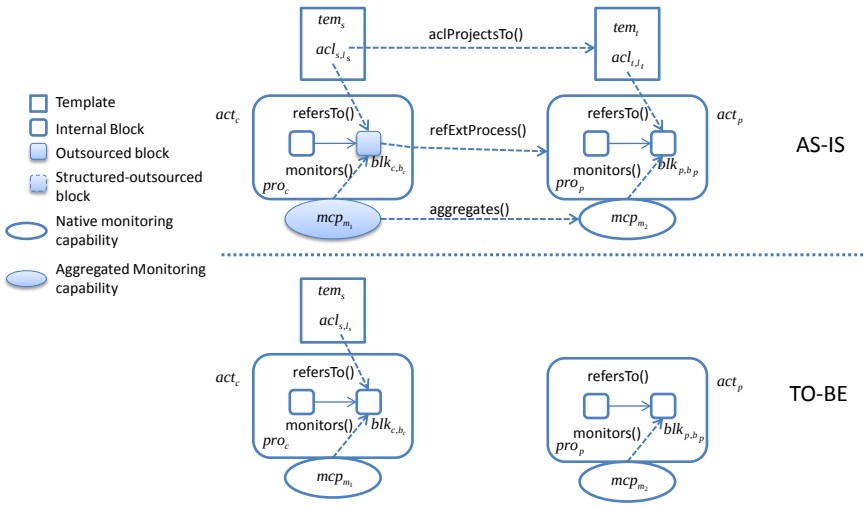

Figure 5: Preserving monitorability for REM-ACL and REM-TEM evolution types

e.g. block blk_{c,b_c} (lines 9-10). Second, in order to preserve monitorability, act_c is required to create a new native monitoring capability for the monitoring of the clause acl_{s,l_s} (lines 11-12).

The time complexity of Algorithm 3 is $O(n \cdot m)$, where n is the number of abstract clauses in tem_t and m is the average number of abstract clauses in other templates that project to abstract clause in tem_t. This means that the time required to react to evolution depends, in this case, on both the structures of templates and on the structure of the BN.

3.4. Clause- and Agreement-level evolution

The pre- and post-conditions for *Clause-* and *Agreement-level* evolution types are shown in Table 7.

Table 7: Pre- and post-conditions for clause-level evolution types.

Evolution type	**Pre-conditions**	**Post-conditions**
ADD-CLA(cla_{g,C_g+1}, acl_{t,l_t})	$acl_{t,l_t} \in ACL_t \wedge agr_g \in AGR \wedge$ $implAbsClause(cla_{g,C_g+1}, acl_{t,l_t}) \wedge$ $cla_{g,C_g+1} \notin CLA_g$	$CLA_g^{to-be} = CLA_g \cup \left\{cla_{g,g,C_g+1}\right\}$
ADD-AGR(agr_{G+1}, tem_t, act_a)	$tem_t \in TEM \wedge act_a \in ACT \wedge$ $implTem\,(agr_{G+1}, tem_t) \wedge$ $agr_{G+1} \notin AGR$	$AGR^{to-be} = AGR \cup \{agr_{G+1}\} \wedge$ $hasConsumer(agr_{G+1}, act_a)$
REM-CLA(cla_{g,c_g})	$cla_{g,c_g} \in CLA_g$	$CLA_g^{to-be} = CLA_g \setminus \left\{cla_{g,c_g}\right\}$
REM-AGR(agr_g)	$agr_g \in AGR \wedge CLA_g \neq \emptyset$	$AGR^{to-be} = AGR \setminus \{agr_g\}$

ADD-CLA evolution type. In this case, a new clause cla_{g,C_g+1} is added to an existing agreement agr_g. Note that, in this paper, we do not focus on the mechanism to determine the compliance of clauses to abstract clauses. In other words, only clauses compliant with abstract clauses (templates) can be added to the BN. As a consequence, the predicate $implAbsClause()$ appears as a pre-condition for the implementation of the evolution type. As post-condition, the clause is added to the set of existing clauses for the agreement agr_g.

ADD-AGR evolution type. In this case, a new agreement agr_{G+1} for which act_a is the consumer is added to the BN. Similar as for clauses, only an agreement compliant with an existing template tem_t can be added. As post-condition, the agreement is added to the set of existing agreements and associated to the actor act_a (as consumer).

REM-CLA evolution type. In this case, a clause cla_{g,c_g}, with $1 \leq c \leq C_g$, is removed from an existing agreement agr_g.

REM-AGR evolution type. In this case, an empty agreement agr_g, with $1 \leq g \leq G$, is removed from the BN. As a result of the application of the evolution type, the agreement is removed from the set of existing agreements.

Note that we consider the addition or deletion of empty agreements. Usually, however, agreements are not empty, but they contain a set of clauses. The establishment of a new agreement in real-world settings is the sequential combination of adding a new empty agreement and then adding all the clauses that have been agreed by the provider and the consumer during the agreement establishment process. Similarly, an agreement can be removed from the BN only when it is empty, i.e. when it does not contain clauses.

The mechanism for preserving monitorability in case of addition of a new clause is shown in Algorithm 4 and illustrated in Figure 6. If the new clause cla_{g,C_g+1} implementing abstract clause acl_{t,l_t} refers to an block blk_{p,b_p} internally executed by act_p, then the new clause will be automatically linked to the monitoring capability exposed by act_p for the monitoring of the corresponding abstract clause acl_{t,l_t} (note that this operation is not made explicit in the algorithm).

Algorithm 4 Preserving monitorability for ADD-CLA evolution type

1: **procedure** ADD-CLA($cla_{g,C_g+1}, acl_{t,l_t}$)
2: $blk_{p,b_p} \leftarrow \{blk \in BLK : refersTo(acl_{t,l_t}, blk)\}$ {identify the block refered to}
3: **if** $blockType(blk_{p,b_p}) = \{O\}$ **then** {block is ousourced}
4: $agr_f \leftarrow \{agr \in AGR : aclProjectsTo(acl_{t,l_t}, acl_{s,l_s}) \wedge implTem(agr, tem_s) \wedge hasConsumer(agr, act) \wedge hasProvider(tem_t, act)\}$ {identify the agreement to which to project to}
5: $cla_{f,C_f+1} \leftarrow claProject(cla_{g,C_g+1}, agr_f)$
6: $acl_{s,l_s} \leftarrow \{acl \in ACL : implAcl(cla_{f,C_f+1}, acl)\}$
7: ADD-CLA($cla_{f,C_f+1}, acl_{s,l_s}$) {recursive call in case the block is further outsourced}
8: **else if** $blockType(blk_{p,b_p}) = \{SO\}$ **then** {block is structured outsourced}
9: **for all** $blk_{ch} : blk_{ch} \in children(blk_{p,b_p})$ **do**
10: ADD-CLA($cla_{g,C_g+1}, acl_{t,l_t}$)
11: **end for**
12: **end if**
13: **end procedure**

If the new clause cla_{g,C_g+1} refers to a block that is outsourced (see conditional statement on line 3), then the clause should be projected to the agreement that governs the outsourced block. In line 4, the agreement to which to project to is identified and the projection is performed in line 5. Because the outsourced block can, in turn, also be outsourced, the ADD-CLA() procedure is called recursively(line 7) with the clause and abstract clause identified in lines 5 and 6. If the new clause cla_{g,C_g+1} refers to a block that is structured outsourced, the clause should be projected to all agreements involved in the outsourced blocks that are part of this structured outsourced block (see the recursive call on line 10). The time complexity of Algorithm 4 is $O(n)$, where n is the number of children of the block to which cla_{g,g_c} refers, under the hypothesis that the non-recursive part of procedure $ADD - CLA()$ executes in constant time.

Algorithm 5 shows the mechanism for preserving monitorability in case of deletion of a clause from an existing agreement. A clause can be deleted only if it does not represent the projection of other clauses in agreements that an actor has established with its own consumers. In other words, in order to preserve the monitorability of the BN, a clause cannot be deleted if other clauses rely on it to project the monitoring requirements on the actors to which the block to which the clause refers is outsourced. A clause can always be removed if it has projections, since the removal does not introduce additional monitoring requirements for the actors in the BN.

Agreement-level evolution is not critical from the point of view of the preservation of the uous moni-

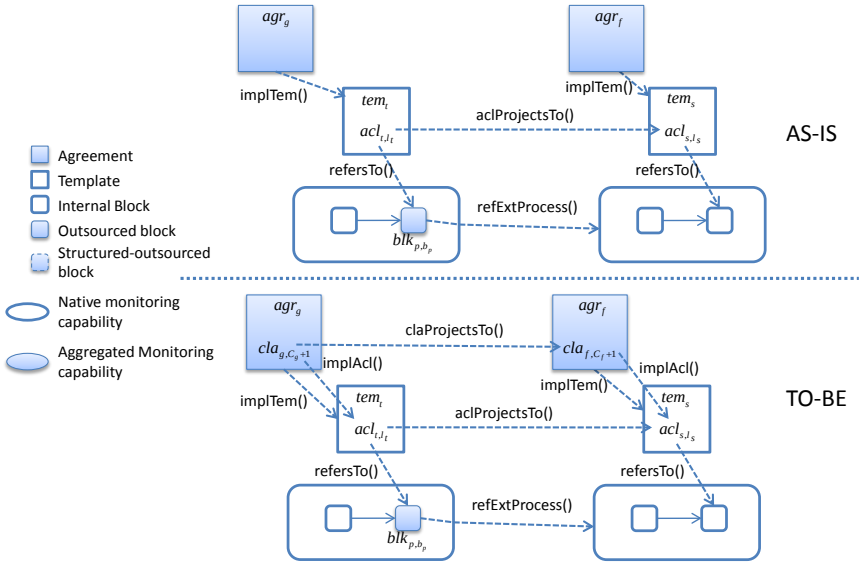

Figure 6: Preserving monitorability for ADD-CLA evolution type

Algorithm 5 Preserving monitorability for REM-CLA evolution type

1: **procedure** REM-CLA(cla_{g,c_g})
2: **if** $\nexists cla \in CLA : claProjectsTo(cla, cla_{g,c_g})$ **then**
3: $CLA_g \leftarrow CLA_g \setminus \{cla_{g,c_g}\}$
4: **end if**
5: **end procedure**

torability of the BN, i.e. the removal or addition of empty agreements in the BN do not entail any specific actions. Note that, as previously remarked, an agreement can be removed only when it is empty, i.e. when all the clauses that appeared in it have been deleted.

4. Monitorability metrics

As discussed in Section 2.1, the *evolution assessment* phase of the extended lifecycle of agreements in evolving business networks requires the definition of a set of monitorability metrics. Monitorability metrics are required to assess the possible evolution of the BN with respect to changes in monitorability within the BN. Every evolution of the BN can potentially lead to an increase or decrease of the monitorability. The monitorability metrics are used in determining whether an evolution is beneficial for the actors involved in the evolution and/or for the BN in its entirety. Again, note that the monitorability metrics are concerned with the possibility of monitoring, and do not include ex-post monitoring possibilities.

We define monitorability metrics along four different perspectives, i.e. *basic*, *provider*, *consumer*, and *network* (see Table 8 for the list of monitorability metrics). *Basic* monitorability metrics refer to the evaluation of the monitorability of an individual template or agreement in the BN. In the *provider* and *consumer* perspectives we define metrics of individual actors within the BN. Specifically, the provider perspective concerns the evaluation of the monitorability of the templates offered and agreements established within the BN by a generic actor act_p as a provider. The consumer perspective concerns the monitorability of the agreements established by a generic actor act_c as a consumer. Eventually, in the *network* perspective we define the global monitorability of the whole set of templates or agreements in the BN.

The basis for the monitorability metrics is the $mon()$ function, which takes as argument an abstract clause acl and returns 1 if the provider of the template to which acl belongs exposes a monitoring capability

Table 8: Monitorability metrics.

Metric	Perspective	Comments
$TM\,(tem_t)$	basic	Monitorability of a template tem_t (Template Monitorability)
$AM\,(agr_g)$	basic	Monitorability of an agreement agr_g (Agreement Monitorability)
$TMPRO\,(act_p)$	provider	Average monitorability of all templates for which act_p is a provider
$AMPRO\,(act_p)$	provider	Average monitorability of all agreements for which act_p is a provider, i.e. agreements that implement templates advertised by act_p
$AMCON\,(act_c)$	consumer	Average monitorability of all agreements for which act_c is a consumer
$TMNET\,(BN)$	network	Average monitorability of all templates in a BN
$AMNET\,(BN)$	network	Average monitorability of all agreements in a BN

mcp associated to *acl*, i.e. the clauses derived from *acl* are monitorable by consumers, and 0 otherwise. In other words, the *mon*() function determines whether an abstract clause is monitorable or not:

$$mon : ACL \mapsto \{0,1\}$$

$$mon\,(acl) = \begin{cases} 1 & \text{if } \exists mcp \in MCP : monitors\,(mcp, acl) \\ 0 & \text{otherwise.} \end{cases}$$

We define the monitorability of a template $TM(tem_t)$ as the percentage of abstract clauses in the template for which a corresponding monitoring capability is exposed by the provider of the template:

$$TM : TEM \mapsto [0,1] \subseteq \mathbb{R}$$

$$TM(tem_t) = \frac{1}{|ACL_t|} \cdot \left[\sum_{acl \in ACL_t} mon\,(acl) \right]$$

The monitorability of an agreement $AM(agr_g)$ is defined as the percentage of clauses in the agreement agr_g that can be monitored through an exposed monitoring capability. Because an agreement contains clauses and monitoring capabilities are associated to abstract clauses, monitorability of an agreement cannot be determined directly. First, the set of abstract clauses that are implemented by clauses in the agreement need to be determined. In this regard, we define the set $IMPL(agr_g)$ as the set of abstract clauses in the template tem_t, with $implTemp(agr_g, tem_t)$, that are implemented by the entire set of clauses in the agreement agr_g:

$$IMPL(agr_g) = \{acl \in ACL_t : implTempl(agr_g, tem_t) \wedge (\exists cla \in CLA_g : implAcl(cla, acl))\}. \quad (1)$$

The percentage of abstract clauses in the set $IMPL(agr_g)$ for which a monitoring capability is exposed provides the monitorability of the agreement agr_g:

$$AM : AGR \mapsto [0,1] \subseteq \mathbb{R}$$

$$AM(agr_g) = \frac{1}{|IMPL(agr_g)|} \cdot \left[\sum_{acl \in IMPL(agr_g)} mon\,(acl) \right]$$

From the provider point of view, we define the metrics $TMPRO(act_p)$ and $AMPRO(act_p)$, which evaluate the average monitorability of the set of templates offered and set of agreements established as a provider, respectively, by the actor act_p.

Let us define the set $TADV(act_p)$, as the set of templates advertised by actor act_p:

$$TADV(act_p) = \{tem \in TEM : hasProvider(tem, act_p)\} \tag{2}$$

The metric $TMPRO(act_p)$ is defined as:

$$TMPRO : ACT \mapsto [0,1] \subseteq \mathbb{R}$$

$$TMPRO(act_p) = \frac{1}{|TADV(act_p)|} \cdot \left[\sum_{tem \in TADV(act_p)} TM(tem) \right]$$

Similarly, to determine the average monitorability of an agreement, in which the actor act_p is a provider, we need to define the set of agreements $AEST_{pro}(act_p)$ established by act_p as provider:

$$AEST_{pro}(agr_p) = \{agr \in AGR : implTem(agr, tem) \wedge hasProvider(tem, act_p)\}. \tag{3}$$

The metric $AMPRO(agr_g, act_p)$ is subsequently defined as:

$$AMPRO : ACT \mapsto [0,1] \subseteq \mathbb{R}$$

$$AMPRO(act_p) = \frac{1}{|AEST_{pro}(act_p)|} \cdot \left[\sum_{agr \in AEST_{pro}(act_p)} AM(agr) \right]$$

The *provider* monitorability metrics are useful for a provider to determine monitoring deviations in the advertised templates and established agreements. For example, if the monitorability of templates is higher than the monitorability of corresponding agreements, then it means that, from a monitorability point of view, the templates advertised by the provider offer more monitoring capabilities than are actually desired (and chosen) by prospective consumers. As such, the resources used by the provider to guarantee the offered monitoring capabilities are not used in the actual agreement established with a consumer, i.e. of the set of monitorable abstract clauses, only a few are actually implemented as clauses in the agreement, or the clauses in the agreement implement mostly those abstract clauses of the template that are not monitorable (that have no monitoring capability associated to them). The definition of provider metrics allows providers to detect such discrepancies so that they can decide to use such resources for other purposes, i.e. those resources can be freed for other uses instead of being reserved as monitoring capabilities for abstract clauses that will not be implemented as agreements (and for which monitoring is not required).

From the consumer point of view, we define the metric $AMCON(act_c)$, which evaluates the average monitorability of the set of agreements established as a consumer by the actor act_c. Note that, in the consumer perspective, we cannot define a metric regarding templates, since, as we already discussed, a template is not directly linked to a consumer (but only through an agreement).

Similar as with the *provider* perspective, we first define the set $AEST_{con}(act_c)$ to be the set of established agreements by act_c as a consumer:

$$AEST_{con}(act_c) = \{agr \in AGR : hasConsumer(agr, act_c)\}. \tag{4}$$

The metric $AMCON(act_c)$ is then defined as:

$$AMCON : ACT \mapsto [0,1] \subseteq \mathbb{R}$$

$$AMCON(act_c) = \frac{1}{|AEST_{con}(act_c)|} \cdot \left[\sum_{agr \in AEST_{con}(act_c)} AM(agr) \right]$$

A high value of the $AMCON(act_c)$ metric can be advantageous for a consumer from a risk management point of view, since the consumer can be aware of the service status of its providers. Moreover, a situation

in which consumer monitoring is linked to consumer control on the service execution enables consumers to immediately take suitable control actions if the service execution deviates from an established agreement [6]. If it is less necessary for a consumer to be constantly aware of the status at its service providers, a lower value of the $AMCON(act_c)$ metric can be acceptable.

Finally, in the network perspective, the metric $TMNET(BN)$ is defined as the average monitorability of all the templates belonging to the BN:

$$TMNET : BN \mapsto [0,1] \subseteq \mathbb{R}$$
$$TMNET(BN) = \frac{1}{|TEM|} \cdot \left[\sum_{tem \in TEM} TM(tem) \right]$$

Similarly, the metric $AMNET(BN)$ is defined as the average monitorability of all the agreements belonging to the BN:

$$AMNET : BN \mapsto [0,1] \subseteq \mathbb{R}$$
$$AMNET(BN) = \frac{1}{|AGR|} \cdot \left[\sum_{agr \in AGR} AM(agr) \right]$$

Both *network* metrics provide an indication on possible improvements in monitorability for the network, by comparing the $AMNET(BN)$ with the $TMNET(BN)$. These metrics can be helpful for organizations that play a coordinator role in the business network. If the monitorability can be improved for the entire network, the coordinator organization can enforce a BN evolution, e.g., by exchanging/replacing actors in the network.

For a more precise indication of where monitorability can be improved in the BN, it is useful to provide a metric that determines the average monitorability of only those templates that refer to the processes/services that are actually executed within the BN, and compare the value of that metric with the $AMNET(BN)$ metric. Note, in fact, that there could be processes in the BN referred to by templates that are never instantiated into agreements. The monitorability of such templates is not important from a network perspective, since the corresponding processes are actually never executed within the BN. In other words, the set of processes in use in the BN is kept fixed in this case, to determine if the monitorability can be improved by changing the set of templates (and consequently the set of agreements governing those processes). The definition of such metric can be achieved in a few steps:

First we define the set of templates that correspond to the set of agreements defined in the BN:

$$TEMA(AGR) = \{tem \in TEM : \exists agr \in AGR \wedge implTem(agr, tem)\} \tag{5}$$

Then we define the set of processes that are actually executed within the BN. As each template refers to a process, but not all templates lead to agreements, this set contains only those processes that are governed by an agreement:

$$PROA(AGR) = \{pro \in PRO : \exists tem \in TEMA(AGR) \wedge hasTemplate(pro, tem)\} \tag{6}$$

Using the set of processes executed in the BN ($PROA(AGR)$), the set of templates that refer to those processes is defined:

$$PROT(PRO) = \{tem \in TEM : \exists pro \in PROA(AGR) \wedge hasTemplate(pro, tem)\} \tag{7}$$

Finally, the monitorability metric can be defined over the set that contains all templates that refer to

processes that are executed in the BN ($PROT(PRO)$):

$$TMPRO : BN \mapsto [0,1] \subseteq \mathbb{R}$$

$$TMPRO(BN) = \frac{1}{|PROT(PRO)|} \cdot \left[\sum_{tem \in PROT(PRO)} TM(tem) \right]$$

If the value of $TMPRO(BN)$ is higher than the value of $AMNET(BN)$, then the monitorability of the templates, and, consequently, processes, actually in use in the BN can possibly be improved. Determining improvements requires further investigation, as the change for improvement depends on the actual abstract clauses that are implemented as clauses in the agreements. If $TMPRO(BN)$ is lower or equal than $AMNET(BN)$, then monitorability cannot be improved and evolution in the BN should not take place (at least not with respect to monitorability; other reasons for evolution could exist though, such as process execution times, costs, etc.).

As illustrated for the $TMPRO(BN)$ metric above, it is (relatively) easy to derive metrics other than the ones defined in Table 8. More advanced metrics can be envisioned that distinguish among the importance of abstract clauses for consumers or among the types of monitoring capabilities associated to abstract clauses, or a combination thereof. Generally, our monitoring framework facilitates the evaluation and the design of monitorabiliy metrics. The evaluation of monitorability metrics should rely on monitoring *dashboards* that, similarly to Key Performance Indicators (KPI) dashboard, continuously assess their value during the operation of a BN.

For a consumer, certain clauses in an agreement and/or template may be more important than others. This can be accommodated by adding a weight factor to the (abstract) clauses, so that the monitorability metric takes the importance of certain (abstract) clauses into account.

A provider could distinguish the monitorability of templates and/or agreements according to the nature of the monitoring capabilities, i.e. native or aggregated. In this way, a provider may discover whether a low monitorability of its own templates is due to shortcomings of its own monitoring infrastructure or of the monitoring options made available by the actors to which parts of the process are outsourced. Aggregated monitorability can be further separated into (i) aggregation of a structured outsourced block in which all children are outsourced (ii) aggregation of a block for which the monitorability of all children is native, or (iii) a combination thereof, i.e. the children of an structured outsourced block are outsourced and/or performed in house.

We consider the definition of advanced metrics, such as the ones mentioned above, as future work.

5. Case study: teleradiology as an evolving business network

To show the applicability of our framework, we present an example scenario derived from a case study in the teleradiology domain. The case study is extensively described, without reference to monitorability issues, in [72] For reasons of clarity, we simplified the scenario to contain only parts relevant for this paper.

Process monitoring, support for providing reminders and alerts, and the ability to capture real-time process information are considered crucial aspects for automated process support in the healthcare domain [64]. Additionally, improving collaborations between healthcare providers [34], linking hospitals with numerous other organizations, e.g. general practitioners, insurance companies, and governmental departments, requires a monitoring framework to enable synchronization of processes and to check collaboration agreement adherence (and to act upon it) [6]. Within the healthcare domain, teleradiology is an already well-established area in which collaborations between different organizations is commonplace and succesful [23, 45, 57], although collaborations are usually considered to be static. In our view, collaborations (and thus agreements) should be established much more dynamically. Hence, the example scenario presented here considers such dynamic possibilities as appropriate.

Consider the situation as depicted in Figure 7. A Hospital (HOS) performs a radiology process, which contains two activities performed in-house (*Acquire*, to acquire the required radiology scans and *Diagnose*

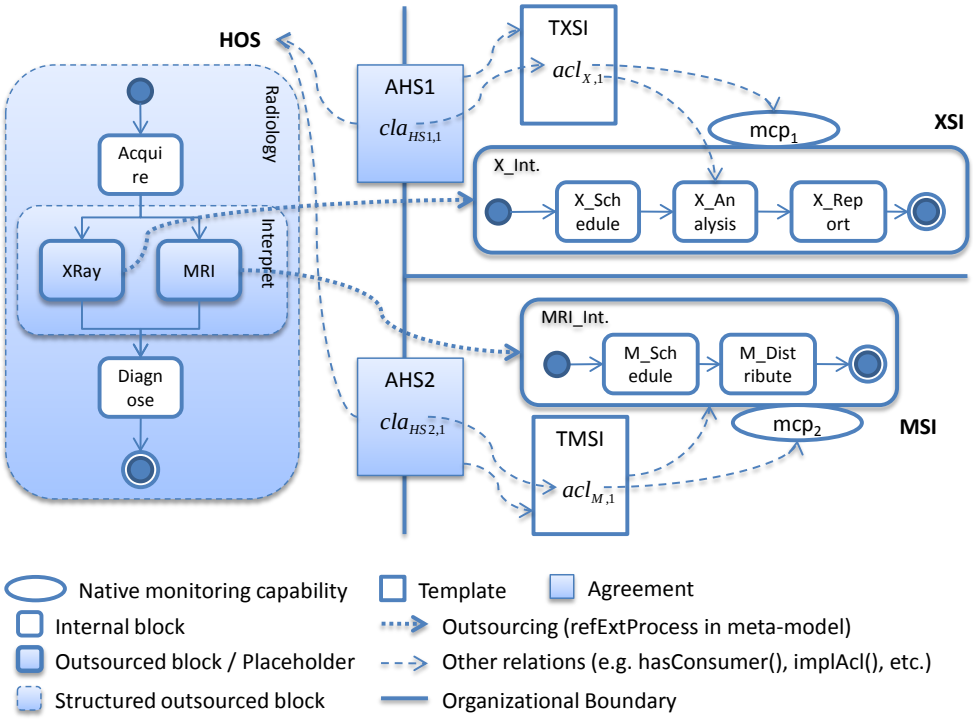

Figure 7: Teleradiology example: initial setting

to determine a diagnosis on the bases of those acquired scans). HOS outsources the interpretation of the acquired scans (*Interpret*) to two specialized scan interpretation service providers (*XSI* and *MSI*), each performing its own process, i.e., *X_Int.* and *MRI_Int.* for the interpretation of X ray scans and MRI scans, respectively. Both scan interpretation service providers have established an agreement with HOS (*AHS1* and *AHS2* in Figure 7), based on the templates they have advertised, i.e. $TXSI$ and $TMSI$, respectively. For simplicity, each template contains only one abstract clause and each agreement contains only one clause (the numbering is for referencing only and does not have a specific meaning). As can be seen in the figure, *XSI* exposes monitoring capability mcp_1 through abstract clause $acl_{X,1}$, which is related to block *X_Analysis* and *MSI* exposes monitoring capability mcp_2 through abstract clause $acl_{M,1}$, related to block *MRI_Int.*. The set-theoretic notation (leaving out subscripts where the meaning is clear) of this initial collaborative setting is shown in Table 9.

The templates and, consequently, the agreements, may contain clauses regarding the possibility for HOS to monitor the identity of the experts performing the interpretation or to be able to monitor the progress of the interpretation process. Hospitals, for instance, may require the identification of scan interpreter for maintaining compliance to internal quality policies or they may need information on the progress of a request to synchronize their own internal processes with the outsourced scan interpretation service. The monitoring requirements derived from the templates MSI and XSI are matched by the monitoring capabilities mcp_1 and mcp_2 exposed by XSI and MSI, respectively.

To illustrate evolution of the teleradiology business network, we first consider the case of adding a consumer organization making use of the radiology service of the hospital. A second evolution consists of the removal of one of the scan interpreters from the business network.

5.1. Evolution example 1: adding a service consumer

Because of spare capacity of the radiology department, HOS decides to offer the radiology process as a service to other parties. To accomplish this, HOS defines a template that advertises the offered service. We first consider a template defining a clause on the acquisition activity *Acquire*. HOS may allow consumers to

Table 9: Set-theoretic representation of the initial teleradiology scenario.

Set Name	**Set Value**	**Predicates with *true* value**
Business Network	$BN = \langle ACT, PRO, TEM, AGR, MCP \rangle$	
Business Actors	$ACT = \{HOS, XSI, MSI\}$	
Processes	$PRO = \{Radiology, X\_Int., MRI\_Int.\}$	$contribute(HOS, Radiology)$ $contribute(XSI, X\_Int.)$ $contribute(MSI, MRI\_Int.)$
Blocks (activities)	$BLK_{Radiology} = \{Radiology^{SO}, Acquire^{I}, Interpret^{SO}, XRay^{O}, MRI^{O}, Diagnose^{I}\}$ $BLK_{X\_Int.} = \{X\_Int.^{I}, X\_Schedule^{I}, X\_Analysis^{I}, X\_Report^{I}\}$ $BLK_{MRI\_Int.} = \{MRI\_Int.^{I}, M\_Schedule^{I}, M\_Distribute^{I}\}$ $BLK = BLK_{Radiology} \bigcup BLK_{X\_Int.} \bigcup BLK_{MRI\_Int.}$	$hasChild\,(Radiology, Acquire)$ $hasChild\,(Radiology, Interpret)$ $hasChild\,(Radiology, Diagnose)$ $hasChild\,(Interpret, XRay)$ $hasChild\,(Interpret, MRI)$ $hasChild\,(X\_Int., X\_Schedule)$ $hasChild\,(X\_Int., X\_Analysis)$ $hasChild\,(X\_Int., X\_Report)$ $hasChild\,(MRI\_Int, M\_Schedule)$ $hasChild\,(MRI\_Int., M\_Distribute)$ $refExtProcess\,(XRay, X\_Int.)$ $refExtProcess\,(MRI, MRI\_Int.)$
Templates	$TEM = \{TXSI, TMSI\}$	$hasProvider\,(TXSI, XSI)$ $hasProvider\,(TMSI, MSI)$ $hasTemplate\,(X\_Int., TXSI)$ $hasTemplate\,(MRI\_Int., TMSI)$
Abstract Clauses	$ACL_{TXSI} = \{acl_{X,1}\}$ $ACL_{TMSI} = \{acl_{M,1}\}$ $ACL = ACL_{TXSI} \bigcup ACL_{TMSI}$	$appliesTo\,(acl_{X,1}, X\_Analysis)$ $appliesTo\,(acl_{M,1}, MRI\_Int.)$
Agreements	$AGR = \{AHS1, AHS2\}$	$implTem\,(AHS1, TXSI)$ $implTem\,(AHS2, TMSI)$ $hasCons\,(AHS1, HOS)$ $hasCons\,(AHS2, HOS)$ $hasProv\,(AHS1, XSI)$ $hasProv\,(AHS2, MSI)$
Clauses	$CLA_{AHS1} = \{cla_{HS1,1}\}$ $CLA_{AHS2} = \{cla_{HS2,1}\}$ $CLA = CLA_{AHS1} \bigcup ACL_{AHS2}$	$implAcl\,(cla_{HS1,1}, acl_{X,1})$ $implAcl\,(cla_{HS2,1}, acl_{M,1})$
Monitoring Capabilities	$MCP = \{mcp_1, mcp_2\}$	$exposes\,(XSI, mcp_1)$ $exposes\,(MSI, mcp_2)$ $monitors\,(mcp_1, acl_{X,1})$ $monitors\,(mcp_2, acl_{M,1})$

monitor the quality level of acquired scans. Getting information on the scan quality may be important for specialized clinics managing complex cases, since low quality scans may prevent the application of advanced diagnosis techniques. In other cases a lower level of quality may be sufficient, e.g. when patients are referred to HOS by a General Practitioner (acting as a consumer of the *Radiology* process).

The evolution of the BN, as described above, includes adding a new template ($ADD-TEM()$ evolution type), followed by the addition of an abstract clause ($ADD-ACL()$ evolution type) to that new template. In order to maintain full monitorability of the published template, i.e. $TMNET(BN) = 1$, adding a new abstract clause requires the addition of a monitoring capability.

If, for instance, the new abstract clause $acl_{HOS,4}$ in the new template $THOS$ concerns the ability of customers to get information on the quality of the acquired scans, then the new native monitoring capability mcp_3 will represent a point of access for customers to the quality of the scan, as obtained by HOS during the execution of the activity *Acquire* in the Radiology process. Note that the creation of the monitoring capability is implied by the application of Algorithm 1 to restore the monitorability of the BN. The changes to the set-theoretic representation, as a result of applying those evolution types, are listed in Table 10.

Now that the radiology service is offered (through template $THOS$), it can be used by service consumers.

Table 10: Changes as a result of offering the radiology process as a service.

Set Name	**Set Value**	**Predicates with *true* value**
Templates	$TEM = \{THOS, TXSI, TMSI\}$	$hasProvider\,(THOS, HOS)$ $hasTemplate\,(Radiology, THOS)$
Abstract Clauses	$ACL_{THOS} = \{acl_{HOS,4}\}$ $ACL = ACL_{THOS} \bigcup ACL_{TSI} \bigcup ACL_{TSIS}$	$appliesTo\,(acl_{HOS,4}, Acquire_{Radiology})$
Monitoring Capabilities	$MCP = \{mcp_3, mcp_1, mcp_2\}$	$exposes\,(HOS, mcp_3)$ $monitors\,(mcp_3, acl_{HOS,4})$

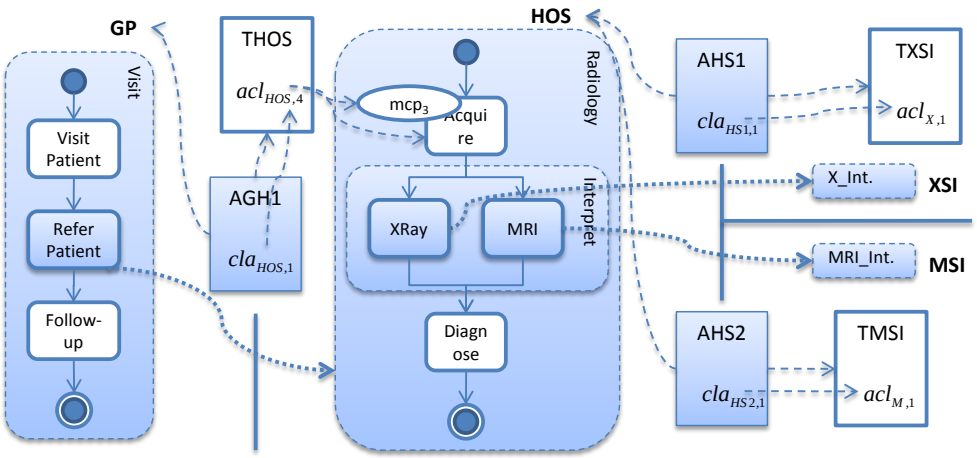

Figure 8: Adding a service consumer to the teleradiology scenario

A general practitioner (GP) is added to the BN and establishes an agreement ($AGH1$) with HOS on the basis of the advertised template ($THOS$). The resulting BN is shown in Figure 8 and the changes to the set-theoretic representation are listed in Table 11.

As an additional complexity in the evolution of the BN, the radiology service consumer (GP) poses an additional monitoring requirement after the agreement has already been established. The desired extra monitoring involves the ability to *monitor intermediate states and data* during the process execution. The GP, for instance, may not want to loose track of a patient, who is referred to HOS for a MRI scan, and requires to monitor the progress of the patient throughout the Radiology process execution.

In this case, the following steps will take place:

1. A new abstract clause (acl_{HOSnew}) is added to the template $THOS$, because the new monitoring requirement is not covered by the existing abstract clauses in the template. The expanded template can subsequently be advertised to other parties, if desired by HOS. It is up to HOS, as the service provider, to decide either to create a new template with the new abstract clause, to update the existing one, or to employ a versioning strategy on the evolution of templates. In this example, we choose to add the new clause to the existing template $THOS$;
2. acl_{HOSnew} is associated with the entire process, which is a block of the structured outsourced type. Therefore, it is (also) linked to the structured outsourced *interpret* block. Then, HOS needs to make sure that its providers can also accommodate for the monitoring specified in acl_{HOSnew}. As a consequence, the abstract clause needs to be projected to the templates that are associated to the externally executed (outsourced) processes. In this case, acl_{HOSnew} needs to be projected to $TXSI$ and $TMSI$ (see abstract clauses acl_{Xnew} and acl_{Mnew}, respectively). Again, new templates can be created, or existing ones can be updates and/or versioned;

Table 11: Changes as a result of adding a consumer for the radiology service.

Set Name	**Set Value**	**Predicates with *true* value**
Business Actors	$ACT = \{GP, HOS, XSI, MSI\}$	
Processes	$PRO = \{Visit, Radiology, X\_Int., MRI\_Int.\}$	$contribute(GP, Visit)$
Blocks (activities)	$BLK_{Visit} = \left\{Visit^{SO}, VisitPatient^{I}, ReferPatient^{O}, Follow-up^{I}\right\}$ $BLK = BLK_{Visit} \bigcup BLK_{Radiology} \bigcup BLK_{X\_Int.} \bigcup BLK_{MRI\_Int.}$	$hasChild\,(Visit, VisitPatient)$ $hasChild\,(Visit, ReferPatient)$ $hasChild\,(Visit, Follow-up)$ $refExtProcess\,(ReferPatient, Radiology)$
Agreements	$AGR = \{AGH1, AHS1, AHS2\}$	$implTem\,(AGH1, THOS)$ $hasCons\,(AGH1, GP)$
Clauses	$CLA_{AGH1} = \{cla_{HOS,1}\}$ $CLA = CLA_{AGH1} \bigcup CLA_{AHS1} \bigcup ACL_{AHS2}$	$implAcl\,(cla_{HOS,1}, acl_{HOS,4})$

3. To maintain the full monitorability of the template $THOS$, according to Algorithm 1, an aggregated monitoring capability needs to be created for this block, i.e., for the entire process. This aggregated monitoring capability aggregates the monitoring capabilities that are created for each of the blocks' children (native for an internal block, aggregated for a (structured) outsourced block). This needs to be done for each of the structured outsourced blocks in a recursive fashion. i.e. for the *interpret* block, XSI and MSI are required to develop native monitoring capabilities concerning the progress of the process, which will be integrated by HOS to let GP monitor intermediate states and data of the *Radiology* process. Note that if HOS cannot create the aggregated monitoring capability, then the monitorability of the BN will decrease in several aspects, in particular, the monitorability of $THOS$ ($TM(THOS) = 0.5$), the monitorability of GP's agreement ($AMCON(GP) = 0.5$), and of the BN as a whole ($TMNET = 0.83$, resulting from a total of three templates, for one of which only half of the abstract clauses are monitorable).
4. The new clause cla_{HOSnew}, which is the implementation of the abstract clause acl_{HOSnew}, can now be added to the agreement following Algorithm 4, which will lead to the clause being projected to the existing agreements between HOS and XSI and between HOS and MSI (cla_{Xnew} and cla_{Mnew}, respectively).

Table 12 presents the changes in the set-theoretic representation as a result of adding the new monitoring requirement of the general practitioner (GP) to the agreement with HOS. Note, only changes with respect to tables 9 to 11 are given.

5.2. Evolution example 2: removing an agreement

For a second example, consider removing the agreement between HOS and one of its interpretation service providers, e.g. the agreement *AHS2* with MSI. The following steps need to be taken to achieve the removal of *AHS2*:

1. The functionality provided by MSI has to be insourced by HOS, so that MRI scans are still being interpreted after the agreement with MSI is removed;
2. The monitoring provided by MSI, has to be provided by HOS natively;
3. The agreement between HOS and MSI, i.e. *AHS2* can now be deleted. Note that the premature deletion of an agreement usually results in some penalties for the party dissolving the agreement.

Note that (2) is required to maintain the monitorability of the template $THOS$ exposed by HOS. HOS may also decide not to create the monitoring capability after insourcing the MRI scan interpretation. In this case, the monitorability of the template $THOS$ will decrease and, consequently, the monitorability of the agreement *AGH1* will also decrease.

Table 12: Changes as a result of adding a new clause to the existing agreement.

Set Name	**Set Value**	**Predicates with *true* value**
Abstract Clauses	$ACL_{THOS} = \{acl_{HOSnew}, acl_{HOS,4}\}$ $ACL_{XSI} = \{acl_{Xnew}, acl_{X,1}\}$ $ACL_{XMSI} = \{acl_{Mnew}, acl_{M,1}\}$	$appliesTo\,(acl_{HOSnew}, Radiology)$ $appliesTo\,(acl_{Xnew}, X\_Int.)$ $appliesTo\,(acl_{Mnew}, MRI\_Int.)$ $aclProjectsTo(acl_{HOSnew}, acl_{Xnew})$ $aclProjectsTo(acl_{HOSnew}, acl_{Mnew})$
Clauses	$CLA_{AGH1} = \{cla_{HOSnew}, cla_{HOS,1}\}$ $CLA_{AHS1} = \{cla_{Xnew}, cla_{HS1,1}\}$ $CLA_{AHS2} = \{cla_{Mnew}, cla_{HS2,1}\}$	$implAcl\,(cla_{HOSnew}, acl_{HOSnew})$ $implAcl\,(cla_{Xnew}, acl_{Xnew})$ $implAcl\,(cla_{Mnew}, acl_{Mnew})$ $claProjectsTo(cla_{HOSnew}, cla_{Xnew})$ $claProjectsTo(cla_{HOSnew}, cla_{Mnew}$
Monitoring Capabilities	$MCP = \{mcp^{A}_{new}, mcp^{N}_{Acquire}, mcp^{A}_{Interpret}, mcp^{N}_{Diagose}, mcp^{N}_{X\_Int.}, mcp^{N}_{MRI\_Int.}, mcp_3, mcp_1, mcp_2\}$	$exposes\left(HOS, mcp^{A}_{new}\right)$ $exposes\,(XSI, mcp_{X\_Int.})$ $exposes\,(MSI, mcp_{MRI\_Int.})$ $monitors\left(mcp^{A}_{new}, acl_{HOSnew}\right)$ $monitors\left(mcp^{A}_{X\_Int.}, acl_{Xnew}\right)$ $monitors\left(mcp^{A}_{MRI\_Int.}, acl_{Mnew}\right)$ $aggregates(mcp^{A}_{new}, (mcp^{N}_{Acqiure}, mcp^{A}_{Interpret}, mcp^{N}_{Diagnose}))$ $aggregates(mcp^{A}_{Interpret}, (mcp^{N}_{X\_Int.}, mcp^{N}_{MRI\_Int.}))$

Figure 9 shows the resulting collaboration setting. Even though *AHS2* was removed from the collaboration, the actor MSI has not been removed from the BN itself. Therefore, MSI still advertises the *MRI_Int.* service with the associated $TMSI$ template. This template contains the original abstract clauses, but also the abstract clause that was added to it as a result of the projection of the abstract clause that was introduced by the service consumer GP, i.e. the *intermediate results monitoring* abstract clause. The actor MSI can be removed after the template $TMSI$ is removed from the BN ($REM - TEM$ evolution type), through the application of a $REM - ACT$ evolution type. This scenario helps to clarify the role of the advanced monitorability metrics given in Section 4. The template TMSI, in fact, belongs to the BN, but it is not instantiated in any agreement and, therefore, it does not refer to any process currently executing in the BN. Let us make the hypothesis that THOS and TXSI have full monitorabiity ($TM() = 1$), whereas TMSI is not monitorable ($TM(TMSI) = 0$). In this case, the monitorability of the BN will be low ($TMNET(BN) = 0.66$), although the processes that are actually executed in the BN are fully monitorable, i.e. $TMPRO(BN) = 1$.

In the next section, a prototype implementation for the monitoring framework is described, using the radiology example scenario as introduced in this section.

6. Prototype implementation

To validate the feasibility of the monitoring framework presented in this paper, we have extended our PROXE prototype system [70]. The PROXE system has been first introduced in [6] offering enhanced control over outsourced business processes to the consumers of such business processes. The system is based on the Business Process Web Service (BPWS) concept [30]. A BPWS is an extension of a regular Web Service (WS) using predefined port types, so that the internal structure of the Web service can be exposed. Through this, it becomes possible to monitor and control the outsourced business process (contained within the service offered and performed by the service provider [71]). What exactly can be monitored and/or controlled is specified in the accompanying electronic contract. The predefined port types in a BPWS are: ACT to invoke a service, MON to monitor a service, $CTRL$ to control a service, $SYNC$ to synchronize with a service, and $SPEC$ to retrieve the specification of the service (process specification, contract, etc.). Because the focus in this paper is on the monitoring framework in relation to business network evolution, the elements of the

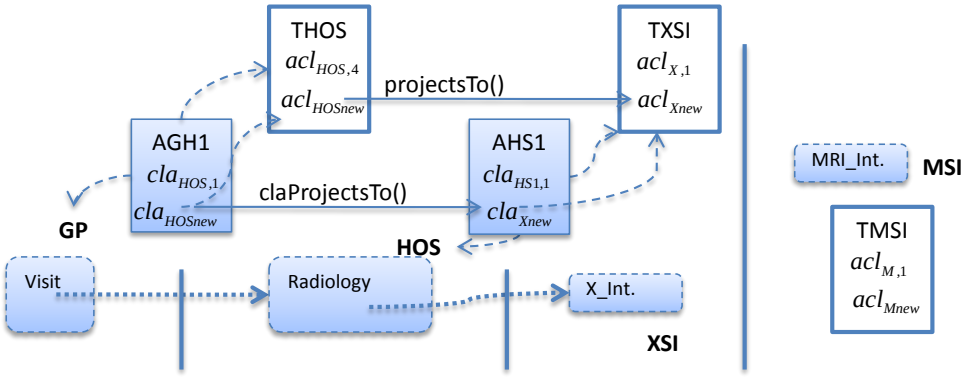

Figure 9: Removing an agreement (AHS2) from the teleradiology scenario

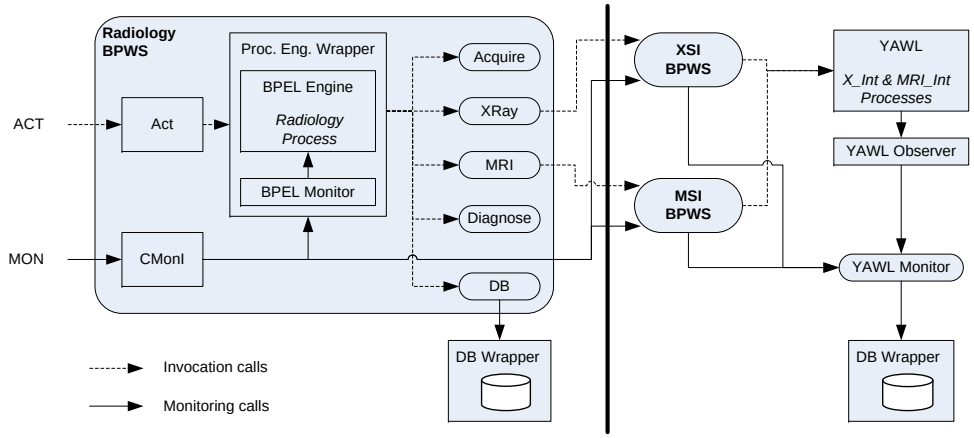

Figure 10: Implementation Architecture

system dedicated to controlling or synchronizing business processes, to retrieve specifications, or to validate against electronic contracts, are not considered here.

Also, the focus of the implementation is on the run-time of the monitoring framework. In particular, we show how we implemented the dynamic sourcing of process blocks and the implementation and instantiation of monitoring capabilities in different process monitoring infrastructures, that is the OpenESB BPEL engine [51] (a BPEL engine) and YAWL [65] (a workflow management system). The implementation does not concern the design time of the framework and, in particular, the automated detection of evolution.

The example scenario used in the PROXE system is the teleradiology business network introduced in the previous section. The radiology process is offered as a service to a service consumer, while two of the activities that are part of the radiology process (*X_Int* and *MRI_Int*) can be outsourced to two other service providers (*XSI* and *MSI*, respectively). The system architecture is shown in Figure 10. Rounded rectangles represent services, either a BPWS or a regular WS. Normal rectangles represent components of the system. Communication between the components and/or services is either related to service invocation (dashed arrows) or to service monitoring (solid arrows).

The *Radiology* BPWS can be invoked (through *ACT*) and monitored (through *MON*). When the radiology process is invoked, the call is passed on to the *Act* component, which validates the call against the electronic contract associated with the process (we use a simple, ad-hoc, XML-based contract language), and in case it is valid, instructs the BPEL engine to start an instance of the radiology process (see Figure 7 for the process structure). Each activity in the BPEL process calls a similarly named WS that performs the tasks associated with that WS. The *DB* WS is used to store information required to correlate the BPEL

process instance with the specific consumer on which behalf the instance has been started. Within the *XRay* and *MRI* Web Services, the choice whether or not these activities should be outsourced, is offered. In case of outsourcing, the appropriate BPWS is called. In other words, outsourcing can be dynamically chosen while the radiology process is executing.

On a call to the *ACT* interface of either *XSI* or *MSI* BPWSs, the PROXE system generates the appropriate monitoring infrastructure based on the clauses specified in the corresponding electronic contract. This means that for each monitoring clause in the electronic contract, the associated monitoring capability will be instantiated and registered in the *CMonI* component, which is discussed more detail later in this section. After registering the monitoring capabilities, the *XSI* BPWS calls the YAWL workflow management system to start an instance of the *X_Int* process, which performs the interpretation of XRay scans. The YAWL Monitor Web Service acts as the monitoring service for processes running on the YAWL engine. It receives the events generated by the YAWL engine (e.g. case termination or change of status, variables and timestamps) via a custom implementation of the *YAWL Observer* gateway [65] and stores the included information into a database. On receiving a monitoring request, it will query the database for relevant data and returns the desired information back to its caller. For the MRI interpretation, the *MSI* BPWS acts in a similar way as explained for the *XSI* BPWS, except that it is associated to the *MRI_Int* process in the YAWL engine. For efficiency reasons, we are using one YAWL system and one database for both XSI and MSI. In reality, these two services could be performed by two different organizations.

Suppose a service consumer has invoked the radiology service and subsequently wants to monitor the process state of this invoked service. In this case, the service consumer calls the *MON* interface, which relays the call to the monitoring component, called *CMonI*. Naturally, a check against the electronic contract is made. If the call is considered valid, the internal logic of the *CMonI* determines if part of the radiology process is outsourced (i.e. XRay, MRI, or both). If nothing is outsourced, the *monitor dispatcher*, as part of *CMonI*, will call the native monitoring capability associated with the specific monitoring request, i.e. with the *retrieve process status* monitoring capability in this example case. This native monitoring capability calls the *BPEL Monitor* component, a native monitoring component provided by the OpenESB BPEL engine collecting basic information on process instances, such as status and variable values. The result received from the *BPEL Monitor* is returned to the *CMonI*, which in turn passes it on to the service consumer. In contrast, if outsourcing takes place, the aggregate monitoring capability that is associated with process state information retrieval, is called. This aggregate monitoring capability calls the *MON* interface of the outsourced activities and, upon receipt of the result, combines the information so that the correct process state can be returned to the service consumer (via the *CMonI*). At *XSI* and *MSI* similar operations take place as a result to a call to their *MON* interfaces. As these two BPWSs do not further outsource any of their activities, only native monitoring capabilities exist for these services. Within the PROXE system, additional monitoring capabilities are available to monitor the state of a specific activity. However, other monitoring capabilities can be easily added. The PROXE system is available as a demonstrator in a virtualized computing platform via SHARE (Sharing Hosted Autonomous Research Environments, [67]).

7. Related Work

Monitoring of business processes is situated within the broader context of continuous assurance, which can be defined as the set of methodology and tools for issuing audit reports and assessing contract compliance simultaneously with, or within a reasonably short period after, the occurrence of relevant events in the process. Compared to ex-post assurance, i.e. building ex-post audit trails to check the compliance of the process execution to overarching agreements, continuous assurance enables providers and consumers to achieve unprecedented benefits, in terms of reduced costs for information collection, search, and retrieval, and more timely and complete detection of deviations from contracts and regulations. Moreover, continuous assurance allows the application of recovery actions on-the-fly, further reducing the risk and costs associated with violations occurrence [3, 6, 15, 32].

Research on monitoring in cross-organizational processes shows three main limitations. First, monitoring is usually not linked to contracts and limited to the reporting of the status of a process to interested parties [27, 68, 59]; second, the setup of the monitoring infrastructure is considered only in the network formation

phase [27, 28]; third, monitoring and the setup of the IT monitoring infrastructure is always considered in 1:1 settings, i.e. the monitoring of one consumer of the processes outsourced to one specific provider [68, 73]. Concerning electronic contracting, the issue of contract dependency has been often identified as a prerequisite for reducing the risks in inter-organizational cooperation [63, 5, 73, 13]. Besides the lack of a comprehensive analysis on how to ensure correct dependencies among contracts as a BN evolves, we also argue that the issue of contract dependency has never been considered together with the setup of the monitoring infrastructure.

From the perspective of monitoring, our framework overcomes the above mentioned limitations of current research. In our approach, monitoring is not limited to the status of processes, but it rather concerns monitoring requirements of a consumer derived from any clause that may be included in an agreement. Moreover, we consider the need for the IT monitoring infrastructure to be constantly updated during the network operation as agreements in the network, and, consequently, the monitoring requirements of actors, evolve. Eventually, network evolution leads us to extend the scope of monitoring beyond 1:1 settings, since the monitoring requirements may be transitive along the chains derived from process outsourcing in a business network.

The problem of monitoring and control of cross-organizational business processes has been tackled by research from several point of views. Specifically, we consider the point of views of workflow management systems and Web services.

7.1. Workflow Management Systems Perspective on Cross-Organizational Process Monitoring

From a requirements engineering perspective, the work in [20] provides a high level overview of coordination mechanisms for cross-organizational enterprise resource planning. In order to achieve a successful collaboration, the authors stress the importance of process- and communication-oriented mechanisms among collaborating parties, i.e. mechanisms that transmit information across the networked organization and that are highly integrated with the process that is collaboratively executed.

The work in [33] presents a classification of control patterns for business networks in the healthcare industry. In this context, our work can be seen as a specialization of the *monitoring* pattern, in which a primary actor, who delegates a task to a counter-actor, faces the need for monitoring the execution of the delegated task. While the monitoring pattern can be applied to elicit the monitoring capabilities required by actors in a given collaboration setting, the work in [33] does not consider evolution of business networks, and, consequently, of monitoring requirements.

The design of cross-organizational workflows with evolving requirements is tackled in [22]. This work focuses on how to flexibly support the evolution of process specifications as requirements, determined for instance by external regulations, evolve, and does not explicitly consider the corresponding evolution of monitoring requirements.

The mechanisms described in this paper hold a close relationship with mechanisms to manage workflow evolution [12, 74]. Workflow evolution considers the problem of maintaining structural and behavioral consistency of flow structure when the specification of the workflow changes, e.g. by adding or removing activities or roles. While workflow evolution focuses on flow structure, this paper focuses on the consistency of agreements and monitoring capabilities of actors in the BN. Workflow evolution focuses on both static evolution of the flow structure and dynamic evolution, i.e. how to maintain consistency for the process instances or cases that have already started. In our framework, we guarantee consistency for static evolution by projecting clauses and creating new monitoring capabilities when a new abstract clause is added to an existing template. Consistency for dynamic evolution is guaranteed by mechanisms to manage evolution when (abstract) clauses or agreements are deleted. The cancellation of an abstract clause, for instance, does not imply the cancellation of the correspondent monitoring capabilities, since instances (cases) that have already started may rely on those monitoring capabilities for monitoring until their termination.

At the architectural and conceptual levels, monitoring in the CrossFlow project [27] concerns only information on the progress of an outsourced process, which is accessible at specific monitoring points specified in the contract. Moreover, CrossFlow considers one-to-one outsourcing scenario and does not account for the transitivity/aggregation of monitoring information in a business network. Dynamic cross-organizational

collaboration is also considered in the CrossWork project [28]. In CrossWork, however, contracts are not considered and monitoring still concerns the progress of outsourced services. Moreover, the impact of the business network evolution on the monitorability of cross-organizational processes is not taken into account. Recursive mechanisms for the definition of goals and processes during the formation of virtual enterprises are considered by the SUDDEN project [40]. Monitoring requirements and evolution of a formed network are, however, not considered.

Similarly to the monitoring capabilities defined in this paper, the E-Adome workflow engine [14] introduces the notion of external information requirement, i.e. information required by a consumer from its providers to enforce and monitor a contract. External information requirements are not directly linked with contract clauses. Moreover, the architectural support for monitoring based on such external information is not specified.

Eventually, from a more technical standpoint, business process monitoring has been largely investigated under the labels of Business Process Intelligence (BPI) [31] and Business Activity Monitoring (BAM) [39]. BPI and BAM, however, are situated in the context of stand-alone organizations and mostly concern process optimization. Auditing for compliance checking has been investigated by research on process mining [66, 36] and normative reasoning applied in the context of business process management [58]. Process mining does not represent a viable solution for continuous assurance, since it relies on the ex-post analysis of process logs. Normative reasoning is focused on defining languages for the formal definition of compliance. Approaches in this category are usually not focused on cross-organizational processes and they tend to overlook the architectural aspects related to cross-organizational collaboration enactment.

7.2. Web service Perspective on Cross-Organizational Process Monitoring

Research on Web service management has also focused on business process monitoring and assurance. Research in this area, however, maintains a technological focus, concerning the definition of XML-based languages for the definition of clear and precise SLAs [63] or the design of monitoring engines compliant with Web service technology [43, 10, 8]. Monitoring in this context may regard the correctness of Web service conversations [56, 73], functional and non-functional requirements of Web service compositions [10, 62], or verification of composite services transactional behavior [26]. An approach for the controlled evolution of Web service contracts is discussed in [4], but without reference to how such an evolution impacts the monitoring of the service execution. A methodology for auditing Web service-based processes is discussed in [52]. Such a methodology, however, can be applied only within the domain of the orchestrator of the collaboration, who captures data for process auditing from its partners, i.e. Web services invoked by the orchestrating process. Our framework extends this perspective by introducing the transitive definition of monitoring requirements along the outsourcing chain and, consequently, the recursive construction of suitable monitoring capabilities.

Being Web service-based, our implementation can be seen as an instance of the OASIS WSDM-MUWS (Management Using Web Services) [46] standard, where monitoring is the only considered management aspect. In particular, from the architecture point of view, the *CMonI* component of the consumer represents the Manageability Consumer in the MUWS standard, whereas the *MON* interface of providers represents the endpoint of the manageable, i.e. monitorable, resources. In this context, the implementation of coordination protocols among consumers and providers could be implemented extending the existing WS-Transaction [47] family of standards.

Eventually, our Web service-based implementation can be seen as multi-tenant [42], since each consumer of processes contributed by a provider has a customized view on monitoring, i.e. it can only access the monitoring capabilities associated with abstract clauses implemented in the specific contract with the provider. Besides being packaged as SaaS (Software as a Service), as in our implementation, the infrastructure required to expose a monitoring capability could be virtualized and made specific to the customer (IaaS). In this way, consumers may be billed not only when accessing monitoring capabilities, but also for the amount of resources that the execution of a monitoring capability actually requires.

The work presented in [18] contains a preliminary investigation of the issues related to monitoring in evolving business networks. This paper extends [18] by considering template-based contracting and

multiple outsourcing of process blocks, by considering all possible evolution types of a BN in an exhaustive way, by discussing a set of candidate monitorability metrics, and, eventually, by presenting a prototype implementation and the application of the proposed mechanisms in a real-world scenario.

8. Conclusions and future work

In this paper, we presented a framework for monitorability of processes governed by agreements in evolving networks. In particular, we focused on how to preserve the monitorability of agreements as the network evolves, e.g. new actors join the network, or contracts in the network are revised or dropped.

After having introduced a meta-model for business networks, we classified the types of evolution that may occur in a business network and, for each of them, we presented algorithms to preserve monitorability. In order to preserve the monitorability of contracts, actors in the network should be able to project clauses and abstract clauses, in order to maintain the coherence with already established contracts, and update their monitoring infrastructure, developing monitoring capabilities that can match the consumers' monitoring requirements that may arise from evolution.

By jointly considering contract replication and the update of the monitoring infrastructure, the framework presented in this paper solves the problem of maintaining control over outsourced activities in the case of dynamic sourcing or in cases where the process consumer requires monitoring information from the provider to synchronize its own internal processes when a network evolves, e.g. in case of partner substitution or update of existing contracts.

In order to show the applicability of our framework, we discussed its application in a real-world healthcare business network. The need for increasing flexibility in healthcare business collaborations has been widely recognized in the literature. In this context, the case study demonstrates how the framework proposed in this paper can preserve the monitorability of a healthcare BN during its evolution. The prototype demonstrates the feasibility of our approach in practice, showing, in particular, how to interface the monitoring infrastructure with existing business process execution engines.

Our work can be extended along several lines. First, the monitoring framework can be coupled with control actions, to be undertaken when a contract violation is detected during the process enactment. Along this line, we are working on generic patterns for integrated monitoring and control. In particular, given a process specification, our objective is to obtain an extended and executable process specification that includes the monitoring and control options required by the process consumers.

Second, we plan to extend the analysis of the monitorability metrics as a tool for supporting decision regarding the evolution of a business network. Specifically, we want to analyze the suitability of metrics in different types of networks, such as peer-to-peer networks or complex supply chains in which a chain leader orchestrates the activities of all involved partners.

References

[1] G. A. Akerlof. The market for lemons: Quality uncertainty and the market mechanism. *Q. J. Econ.*, 84(3):488–500, 1970.

[2] A. Albani and J. Dietz. Current trends in modeling inter-organizational cooperation. *Journal of Enterprise Information Management*, 22(3):275–297, 2009.

[3] M. G. Alles, A. Kogan, and M. A. Vasarhely. Feasibility and economics of continuous assurance. *Auditing: Journal of Practice and Theory*, 21:1–14, 2002.

[4] V. Andrikopoulos, S. Benbernou, and M. Papazoglou. Evolving services from a contractual perspective. In *Proc. Int. Conf. on Advanced Information Systems Engineering*, pages 290–304, 2009.

[5] S. Angelov and P. Grefen. An E-contracting reference architecture. *J. Syst. Software*, 81(11):1816–1844, 2008.

[6] S. Angelov, J. Vonk, K. Vidyasankar, and P. Grefen. Enhancing business collaborations with client-oriented process control. *International Journal of Cooperative Information Systems*, 20(01):1+, 2011.

[7] B. A. Aubert, M. Patry, and S. Rivard. A framework for information technology outsourcing risk management. *SIGMIS Database*, 36(4):9–28, 2005.

[8] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of Web service compositions. In *Proceedings of the IEEE International Conference on Web Services*, pages 63–71, Washington, DC, USA, 2006. IEEE Computer Society.

[9] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Software*, 1(6):219–232, 2007.

[10] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti. Dynamo + Astro: An integrated approach for BPEL monitoring. In *Proc. IEEE International Conference on Web services*, 2009.
[11] A. Brown and G. Grant. Framing the frameworks: A review of IT governance research. *Communications of the AIS*, 15:696–712, 2005.
[12] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data & Knowledge Engineering*, 24(3):211–238, 1998.
[13] J. C. Cheng, K. H. Law, H. Bjornsson, and S. R. D. Jones, Albert. Modeling and monitoring of construction supply chains. *Advanced Engineering Informatics*, 24:435–455, 2010.
[14] D. K. W. Chiu, K. Karlapalem, Q. Li, and E. Kafeza. Workflow view based E-contracts in a cross-organizational E-services environment. *Distrib. Parallel. Dat.*, 12:193–216, 2002.
[15] D. Coderre. Continuous auditing: Implications for assurance monitoring and risk assessmen. Technical report, The Institute of Internal Auditors Research Foundation, 2005.
[16] M. Comuzzi, C. Kotsokalis, C. Rathfelder, W. Theilmann, U. Winkler, and G. Zacco. A framework for multi-level SLA management. In *ICSOC/ServiceWave Workshops*, pages 187–196, 2009.
[17] M. Comuzzi and B. Pernici. A framework for QoS-based Web service contracting. *ACM Transactions on the Web*, 3(3):1–52, 2009.
[18] M. Comuzzi, J. Vonk, and P. Grefen. Continuous Monitoring in Evolving Business Networks. In R. Meersman, T. Dillon, and P. Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2010*, volume 6426 of *Lecture Notes in Computer Science*, chapter 14, pages 168–185. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.
[19] P. Dadam and M. Reichert. The adept project: a decade of research and development for robust and flexible process support. *Computer Science - R&D*, 23(2):81–97, 2009.
[20] M. Daneva and R. Wieringa. A requirements engineering framework for cross-organizational ERP systems. *Requirements Engineering*, 11:194–204, 2006.
[21] N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, 30(12):859–872, Dec. 2004.
[22] N. Desi, A. K. Chopra, and M. P. Singh. Amoeba: A methodology for modeling and evolving cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology*, 29(2), 2009.
[23] S. L. Dimmick and K. D. Ignatova. The diffusion of a medical innovation: where teleradiology is and where it is going. *J Telemed Telecare*, 12(suppl_2):51–58, September 2006.
[24] K. M. Eisenhardt. Agency theory: An assessment and review. *The Academy of Management Review*, 14(1):57–74, 1989.
[25] R. Eshuis and P. Grefen. Constructing customized process views. *Data and Knowledge Engineering*, 64(2):419–438, 2008.
[26] W. Gaaloul, S. Bhiri, and M. Rouached. Event-based design and runtime verification of composite service transactional behavior. *IEEE Transactions on Services Computing*, 3(1):32–45, 2010.
[27] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: cross-organizational workflow management in dynamic virtual enterprises. *Comput. Syst. Sci. & Eng.*, 5:277–290, 2000.
[28] P. Grefen, R. Eshuis, N. Mehandijev, G. Kouvas, and G. Weichart. Internet-based support for process-oriented instant virtual enterpises. *IEEE Internet Comput.*, pages 30–38, 2009.
[29] 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.
[30] P. Grefen, H. Ludwig, A. Dan, and S. Angelov. An analysis of web services support for dynamic business process outsourcing. *Information & Software Technology*, 48(11):1115–1134, 2006.
[31] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan. Business process intelligence. *Computers in Industry*, 53:321–343, 2004.
[32] A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43:618–644, 2007.
[33] V. Kartseva, J. Hulstijn, J. Gordijn, and Y.-H. Tan. Control patterns in a health-care network. *European Journal of Information Systems*, 19:320–343, 2010.
[34] R. Lenz and M. Reichert. IT support for healthcare processes â“ premises, challenges, perspectives. *Data & Knowledge Engineering*, 61(1):39–58, April 2007.
[35] F. Leymann, D. Roller, and M. T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.
[36] C. Lia, M. Reichert, and A. Wombacher. Mining business process variants: Challenges, scenarios, algorithms. *Data Knowl. Eng.*, 70(5):409–434, 2011.
[37] P. F. Linington, Z. Milosevic, J. B. Cole, S. Gibson, S. Kulkarni, and S. W. Neal. A unified behavioural model and a contract language for extended enterprise. *Data and Knowledge Engineering*, 51(1):5–29, 2004.
[38] T. Malone, J. Yates, and R. Benjamin. Electronic markets and electronic hierarchies. *Commun. ACM*, 30(6):484–497, 1987.
[39] D. W. McCoy. Business activity monitoring. Technical report, Gartner Group Research Report ID LE-15-9727, 2002.
[40] N. D. Mehandjiev, I. D. Stalker, and M. R. Carpenter. Recursive construction and evolution of collaborative business processes. In *Proc. 2nd Int. Workshop on Collaborative Business Processes*, 2008.
[41] J. Mendling, H. Reijers, and W. van der Aalst. Seven process modeling guidelines (7PMG). *Information and Software Technology*, 52(2), 2010.
[42] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. *ICSE Workshop on Principles of Engineering Service Oriented Systems*, 0:18–25, 2009.
[43] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *Proc. World*

Wide Web Conference, 2008.
[44] J. Motwani, A. Kumar, J. Jiang, and M. Youssef. Business process reengineering: A theoretical framework and an integrated model. *International Journal of Operations & Production Management*, 18(9/10):964–977, 1998.
[45] S. K. Mun, W. G. Tohme, R. C. Platenberg, and I. Choi. Teleradiology and emerging business models. *J Telemed Telecare*, 11(6):271–275, September 2005.
[46] OASIS. Web Services Distributed Management (WSDM) Technical Committee. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm`, 2011.
[47] OASIS. Web Services Transaction (WS-TX) technical committee. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx`, 2009.
[48] OASIS. ebXML Collaboration Protocol Profile and Agreement (CPPA). `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-cppa`, 2011.
[49] Object Management Group (OMG). Business Process Model and Notation (BPMN), January 2011.
[50] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-Agreement partner selection. In *Proceedings of the 15th international conference on World Wide Web*, pages 607–706, 2006.
[51] Oracle Inc. OpenESB: The open enterprise service bus, 2011. `http://wiki.open-esb.java.net/Wiki.jsp/`.
[52] B. Orriens, W.-J. van den Heuvel, and M. Papazoglou. On the risk management and auditing of SOA based business processes. In *Proc. 3rd Int. ISoLA Symposium*, 2008.
[53] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *Int. J. Cooperative Inf. Syst*, 17(2):223–255, 2008.
[54] P. Radha Krishna, K. Karlapalem, and D. Chiu. An ER^{EC} framework for e-contract modeling, enactment, and monitoring. *Data and Knowledge Engineering*, 51:31–58, 2004.
[55] W. Robinson. A requirements monitoring framework for enterprise systems. *Requirements Engineering*, 11:17–41, 2006.
[56] W. Robinson and S. Purao. Monitoring service systems from a language-action perspective. *IEEE Transactions on Services Computing*, (forthcoming), 2011.
[57] C. Ruggiero. Teleradiology: a review. *J Telemed Telecare*, 4(1):25–35, March 1998.
[58] S. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In *Proc. 5th Business Process Management Conference*, pages 149–164, 2007.
[59] M. Sailer and M. Morciniec. Monitoring and execution for contract compliance. Technical Report HPL-2001-161, E-Service Markets Department, HP Labs Bristol, 2005.
[60] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst. Process flexibility: A survey of contemporary approaches. In J. L. G. Dietz, A. Albani, and J. Barjis, editors, *CIAO! / EOMAS*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer, 2008.
[61] R. Seguel, R. Eshuis, and P. Grefen. Generating minimal protocol adaptors for loosely coupled services. In *Proc. 8th IEEE International Conference on Web Services*, pages 417–424, 2010.
[62] J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse. Runtime monitoring of Web service conversations. *IEEE Trans. Serv. Comput.*, 2:223–244, July 2009.
[63] J. Skene, F. Raimondi, and W. Emmerich. Service-level agreements for electronic services. *IEEE Transactions on Software Engineering*, 36(2):288–304, 2010.
[64] X. Song, B. Hwong, G. Matos, A. Rudorfer, C. Nelson, M. Han, and A. Girenkov. Understanding requirements for computer-aided healthcare workflows: experiences and challenges. In *ICSE*, pages 930–934, 2006.
[65] A. ter Hofstede, W. van der Aalst, M. Adamns, and N. Russell, editors. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010.
[66] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance checking of service behavior. *ACM Transactions on Internet Technology*, 8(3), 2008.
[67] P. Van Gorp and P. Grefen. Supporting the internet-based evaluation of research software with cloud infrastructure. *Software & Systems Modeling*, May 2010.
[68] E. van Heck and P. Vervest. Smart business networks: How the network wins. *Communications of the ACM*, 50:28–37, 2007.
[69] P. Vicente and M. Mira da Silva. A conceptual model for integrated governance, risk and compliance. In *Proc. 23rd Int. Conf. on Advanced Information Systems Engineering*, pages 199–213, 2011.
[70] J. Vonk and M. Comuzzi. Proxe v3.0: PROcess eXecution Environment with monitoring, March 2011. `http://is.ieis.tue.nl/research/share.html`.
[71] J. Vonk, T. Wang, and P. Grefen. A Dual View to Facilitate Transactional QoS. *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, 0:381–383, 2007.
[72] J. Vonk, T. Wang, P. Grefen, and M. Swennenhuis. An analysis of contractual and transactional aspects of a teleradiology process. Beta Technical Report WP-263, Eindhoven University of Technology, December 2008.
[73] B. Wetzstein, D. Karastoyanova, O. Kopp, F. Leymann, and D. Zwink. Cross-organizational process monitoring based on service choreographies. In *Proc. 25th ACM Symposium on Applied Computing*, pages 2485–2490, 2010.
[74] E. Withana, B. Pale, R. Barga, and N. Araujo. Versioning for workflow evolution. In *Proc. 19th ACM International Symposium on High Performance Distributed Computing*, pages 756–765, 2010.
[75] WS-AGREEMENT. WS-Agreement Framework. `https://forge.gridforum.org/projects/graap-wg`, September 2003.
[76] zur Muehlen. *Workflow-based Process Controlling*. Springer-Verlag, 2005.