Continuous Monitoring in Evolving Business Networks

Marco Comuzzi, Jochem Vonk, Paul Grefen

Eindhoven University of Technology, Eindhoven, The Netherlands
{m.comuzzi, j.vonk, p.w.p.j.grefen}@tue.nl

Abstract. The literature on continuous monitoring of cross-organizational processes, executed within virtual enterprises or business networks, considers monitoring as an issue regarding the network formation, since what can be monitored during process execution is fixed when the network is established. In particular, the impact of evolving agreements in such networks on continuous monitoring is not considered. Also, monitoring is limited to process execution progress and simple process data. In this paper, we extend the possible monitoring options by linking monitoring requirements to generic clauses in agreements established across a network and focus on the problem of preserving the continuous monitorability of these clauses when the agreements evolve, i.e. they are introduced, dropped, or updated. We discuss mechanisms to preserve continuous monitorability in a business network for different types of agreement evolution and we design a conceptual and technical architecture for a continuous monitoring IT infrastructure that implements the requirements derived from such mechanisms.

Keywords: Monitoring, Business Network, Agreement Evolution.

1 Introduction

In collaborative settings, where autonomous parties perform part(s) of a business process governed by an established agreement/contract, continuous assurance of a process can be defined as the set of methodology and tools for issuing audit reports and assessing compliance to agreements simultaneously with, or within a reasonably short period after, the occurrence of relevant events in the process. Compared to ex-post assurance, which relies on ex-post audit trails, 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. Moreover, continuous assurance allows the application of recovery actions on-the-fly, further reducing the risks associated with deviations occurrence [1],[2].

Faster market dynamics and fiercer competition have pushed organizations to engage in complex, Internet-enabled, highly dynamic collaborations, referred to as virtual enterprises (organizations) or collaborative business networks [3],[4],[5]. Such

collaborations entail the enactment of cross-organizational business processes, which are regulated by agreements established between the participants constituting the business network. Because of the high variability of the environment in which they are situated, however, agreements in a network may evolve during the network lifetime. We argue that the evolving nature of business networks poses additional challenges for continuous assurance of cross-organizational business processes, since the IT infrastructure supporting assurance should adapt to enable and preserve continuous assurance in reaction to evolution.

Assurance is constituted by two phases, namely the *monitoring* and the *auditing* phases [2]. From an individual actor's perspective, monitoring concerns the collection of relevant information regarding the agreements established with other actors in the network. Examples of monitoring information are: the process execution progress and exceptions, data produced during the process execution, response times of invoking services, choices made and process paths taken in the process execution (and the argumentation leading to those choices), etc. Proper monitoring represents a prerequisite for correct and complete auditing, i.e. checking the compliance of process execution with constraints set by agreements. In this paper we focus on the monitoring phase, and, therefore, on continuous monitoring of cross-organizational business processes. Our objective is to study how to guarantee continuous monitoring in a business network in which agreements evolve during the network operation.

Research on continuous monitoring in cross-organizational processes shows three main limitations. First, monitoring is usually limited to the reporting of the status (progress and simple process-level variables) of a process execution to interested parties [4], [6]. Second, monitoring and the setup of the IT monitoring infrastructure is always considered in 1:1 settings, i.e. the monitoring by one consumer of the processes outsourced to one specific provider [6]. Third, the setup of the monitoring infrastructure is considered only in the network formation phase [3], [4], i.e., at the time an agreement is established and fixed (it cannot change anymore). In our approach, continuous 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. In addition, we consider the possibility for agreements in the network, and, consequently, the monitoring requirements of participants, to evolve, extending the scope of monitoring beyond the 1:1 setting. Eventually, because of evolving agreements, we consider the need for the IT monitoring infrastructure to be constantly updated to be able to facilitate the continuous monitoring in such evolving business networks.

In this paper, after having introduced a running example and the main concepts related to collaborations in business networks (in Section 2), we classify the types of evolution of business networks (Section 3) and discuss mechanisms to preserve the continuous monitorability of agreements in reaction to their evolution (Section 4). Then, we design the conceptual and technical architecture of the continuous monitorability IT infrastructure of one actor in a business network (Section 5). A prototype implementation of the architecture and mechanisms is presented next (Section 6) and the paper ends with related work (Section 7), conclusions and an outline of future work (Section 8).

2 Business networks

To introduce the concepts related to business networks, we first consider a running example of a Business Network (BN) in the healthcare domain. The example is a simplification of a real-world teleradiology process extensively described in [7]. A representation of the BN at a given moment in time is shown in Fig. 1. A set of business actors participate in the BN (GP, PC, HOS, and SIS in the example). In their internal processes, general practitioners (GPs) and private clinics (PCs) rely on a hospital (HOS) to provide a radiology service. Each process comprises a sequence of blocks, i.e. structured set of activities. A block may be either executed internally by a business actor or outsourced to another business actor in the BN. In the example, HOS runs internally the scan acquisition (sa) and the result transfer (rt) blocks, and outsources the scan interpretation (si) to a scan interpretation service provider (SIS).

The cross-organizational collaboration among actors in the BN is regulated by established contracts, e.g. on service levels, and by internal or industry-level policies. We use the generic term *agreement* to identify the artifacts regulating the provisioning of processes among business actors in the BN. Agreements are constituted by a set of *clauses*, which capture the individual cross-organizational constraints on the BN operation.

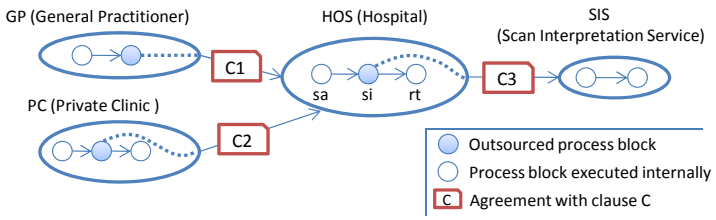


Fig. 1. Example BN in healthcare industry

As a sample, we consider the following three types of clauses that may appear in agreements between actors in the BN of Fig. 1:

- C1: HOS allows GP to get information about the quality of acquired scans;
- C2: HOS must guarantee that scan interpretation is performed by two different scan interpreters, and their identity should be traceable by PC;
- C3: SIS must guarantee that scan interpretation is performed by two different scan interpreters, and their identity should be traceable by HOS.

For continuous monitorability, C1 requires HOS to make available information regarding the quality of scan acquisitions to GP. C3 requires SIS to disclose information on the identity of the interpreters of the scan to HOS. Similarly, C2 requires HOS to disclose information on the identity of the scan interpreters to PC. The continuous monitoring of the clauses defined for the BN in Fig. 1, therefore, can be guaranteed if service providers, i.e. HOS and SIS, expose suitable monitoring capabilities to match the monitoring information requirements of their consumers, i.e. GP, PC, and HOS, respectively. We define a monitoring capability of a provider in the network as the ability to provide the specific monitoring information required by

consumers for continuously monitor a given clause. Monitoring capabilities exposed by a business actor are queried by consumers to obtain the information required for checking the compliance of process execution with clauses that apply to it.

Note that in our example, a dependency exists between clauses C3 and C2, since HOS cannot guarantee the continuous monitoring of C2 if it cannot access the information required to monitor C3, i.e., the identity of scan interpreters. The monitoring capabilities to match the information requirements of PC is built by HOS using the information obtained through the monitoring capabilities exposed by SIS to let HOS monitor C2. Hence, monitoring capabilities may be either *native* or *aggregated*. Native capabilities refer to information that can be captured within the business domain of the provider of the monitored clause, e.g. SIS exposes a native monitoring capability to HOS for monitoring C3 and HOS exposes a native capability to GP for monitoring C1. Aggregated monitoring capabilities are built by the provider of a clause through information obtained by the monitoring capabilities of other providers to which it outsources part of its process, e.g. HOS exposes an aggregated capability to PC for monitoring C2.

3 Modeling evolving business networks

In this section we introduce a formal model of business networks and their evolution. The model allows us, in Section 4, to describe algorithmically the mechanisms for the preservation of continuous monitorability in reaction to evolution.

3.1 Modeling business networks

Our model of a business network relies on a set of simplifying assumptions. With these assumptions, we limit the complexity of our notation without losing focus on the main principles behind our monitorability preservation mechanisms. We discuss later in Section 4 how such assumptions can be relaxed in more complex scenarios:

1. One actor contributes only one process to the network;
2. Any pair of actors in the network, i.e. a provider and a consumer, can establish at most one agreement;
3. We consider single-entry, single-exit, block-structured processes [8]; blocks are not hierarchically structured, i.e. a process is constituted by a flat set of blocks structured according to common business process patterns, such as sequential, conditional or parallel execution, and loops.
4. For the provider, a clause in an agreement always refers to one specific block in the process contributed to the BN;
5. A block in a process can be outsourced to at most one actor;

A business network *BN* is defined as:

$$BN = \langle ACT, PRO, AGR \rangle$$

where *ACT* is the set of actors, *PRO* the set of processes, and *AGR* the set of agreements. *BN* has *A* actors act_a:

$$ACT = \{act_a\}_{a=1,\dots,A}.$$

Because of assumption 1, we can write:

$$PRO = \{pro_a\}_{a=1,\dots,A}$$

where pro_a is the process contributed by the actor act_a.

For a process pro_a we define the set BLK_a of its K_a blocks:

$$BLK_a = \left\{blk_{a,k_a}^{bt}\right\}_{\substack{bt \in \{IN,OUT\} \\ k_a=1,\dots,K_a}}$$

A block *blk* may be either executed internally by act_a (block type $bt = IN$), e.g. the block sa in HOS's process in Fig. 1, or outsourced to another actor act_b ($bt = OUT$), e.g. block si for HOS in Fig. 1. In the remainder, we will make the block type *bt* explicit only when necessary.

We capture the outsourcing relation through the predicate:

$$executedBy\left(blk_{a,k_a}^{OUT}, act_b\right)$$

which evaluates to true if the block blk_{a,k_a}^{OUT} is outsourced by act_a to act_b, and to false otherwise.

An agreement always regulates the outsourcing relationship between act_c (consumer) and act_p (provider). Hence, given the set of all possible agreements *P*, we define the set of agreements *AGR* in place in the *BN* as:

$$AGR = \left\{agr_{c,p} \in P: \exists blk_{c,k_c}^{OUT} \in BLK_c \wedge executedBy\left(blk_{c,k_c}^{OUT}, act_p\right) \wedge act_c, act_p \in ACT\right\}$$

For each actor act_a, we define the provider set $PS(act_a)$ as the set of actors to which act_a outsources part of its process, and the consumer set $CS(act_a)$ as the set of actors that have outsourced part of their processes to act_a:

$$PS(act_a) = \left\{act_j: \exists agr_{a,j} \in AGR \wedge act_j \in ACT\right\}$$
$$CS(act_a) = \left\{act_j: \exists agr_{j,a} \in AGR \wedge act_j \in ACT\right\}$$

An agreement $agr_{c,p}$ is constituted by $L_{c,p}$ clauses. We define the set of $L_{c,p}$ clauses $CLA_{c,p}$ of the agreement $agr_{c,p}$ as:

$$CLA_{c,p} = \left\{cla_{c,p,l_{c,p}}\right\}_{l_{c,p}=1,\dots,L_{c,p}}$$

Note that, because of assumption 4, from the point of view of the provider act_p, a clause $cla_{c,p,l_{c,p}}$ always refers to one specific block blk_{p,k_p}. This association between clauses and blocks is captured by the predicate:

$$refersTo\left(cla_{c,p,l_{c,p}}, blk_{p,k_p}^{bt}\right)$$

which evaluates to true if $cla_{c,p,l_{c,p}}$ refers to blk_{p,k_p}, and to false otherwise. In Fig. 1, for actor HOS, C1 refers to the block sa, whereas C2 refers to the block si outsourced to SIS. Similarly, for SIS, C3 refers to a block executed internally.

From the provider point of view, outsourcing entails also a relation between clauses. Specifically, if a provider act_p has outsourced a block of its process pro_p to another provider act_d, and act_p has also established a clause $cla_{c,p,l_{c,p}}$ with an actor act_c, with $act_c \in CS(act_p)$, which refers to the outsourced block, then a link exists

between the clauses $cla_{c,p,l_{c,p}}$ and the clause $cla_{p,d,l_{p,d}}$ that regulates the provisioning of the process pro_d to act_p by act_d. We say in this case that $cla_{p,d,l_{p,d}}$ is a projection of $cla_{c,p,l_{c,p}}$. The projection relation between clauses is captured by the predicate:

$$projectsTo\left(cla_{c,p,l_{c,p}}, cla_{p,d,l_{p,d}}\right)$$

In Fig. 1, the predicate *projectsTo* can be used to capture the relation between the clauses C2 and C3. Specifically, with an abuse of notation, if we write C2 as $cla_{PC,HOS,2}$ and C3 as $cla_{HOS,SIS,3}$, then the predicate $projectsTo\left(cla_{PC,HOS,2}, cla_{HOS,SIS,3}\right)$ holds for the shown *BN*. Although not explicitly modeled, we hypothesize that clause projection can only occur between semantically related clauses. In our example, both C2 and C3 refer to the four eyes principle on scan interpretation.

Providers expose monitoring capabilities to match monitoring requirements, derived from the establishment of clauses. The actor act_a exposes a set of M_a monitoring capabilities MCP_a:

$$MCP_a = \left\{mcp_{a,m_a}^{mt}\right\}_{\substack{mt\in\{NAT,AGG\}\\ m_a=1,\ldots,M_a}}$$

A monitoring capability may be either native (monitoring capability type *mt*=*NAT*) or aggregated (*mt*=*AGG*). In the remainder, we will make the monitoring capability type *mt* explicit only when necessary.

3.2 Modeling evolution of business networks

Since we aim at designing the continuous monitoring IT infrastructure of one actor in the BN, in modeling evolution of BNs we take the vantage point of an individual actor act_a in the BN. Thus, evolution concerns the modifications of the agreements in which act_a is involved, as either a provider or a consumer. We distinguish between two evolution categories: *clause-* and *agreement-level* evolution; within each category, we identify several evolution types. An evolution type is modeled by listing the changes that it implies on the BN. Specifically, the BN in the *as-is* situation is compared to the BN in the *to-be* situation, i.e. the BN resulting from the application of the evolution type.

Clause-level evolution. This concerns the insertion or deletion of clauses in an existing agreement in which act_a is involved (an update of a clause is a sequential combination of a deletion and insertion). We identify four types of clause-level evolution:

1. *Insert clause in provider-side agreement* (INS_PSC): it occurs when a new clause $cla_{j,a,n}$, with $n=L_{j,a}+1$, is added to an existing agreement with a consumer act_j, with $act_j \in CS(act_a)$:
$$CLA_{j,a}^{to-be} = CLA_{j,a}^{as-is} \cup \left\{cla_{j,a,n}\right\}$$
2. *Insert clause in consumer-side agreement* (INS_CSC): it occurs when a new clause $cla_{a,j,n}$, with $n=L_{a,j}+1$, is added to an existing agreement with a provider act_j, with $act_j \in PS(act_a)$:

$$CLA_{a,j}^{to-be} = CLA_{a,j}^{as-is} \cup \{cla_{a,j,n}\}$$

3. *Delete clause in provider-side clause* (DEL_PSC): it occurs when a clause $cla_{j,a,l}$, with $1 \leq l \leq L_{j,a}$, is removed from an existing agreement with a consumer act_j, with $act_j \in CS(act_a)$.
$$CLA_{j,a}^{to-be} = CLA_{j,a}^{as-is} \setminus \{cla_{j,a,l}\}$$

4. *Delete clause in consumer-side agreement* (DEL_CSC): it occurs when a clause $cla_{a,j,l}$, with $1 \leq l \leq L_{a,j}$ is removed from an existing agreement with a provider act_j, with $act_j \in PS(act_a)$.
$$CLA_{a,j}^{to-be} = CLA_{a,j}^{as-is} \setminus \{cla_{a,j,l}\}$$

Clause-level evolution is typical of highly regulated industries, such as the financial or healthcare industries. For what concerns the financial industry, for instance, changes of regulations, such as the introduction of the Sarbanes-Oxley act [9] in the US (and similar regulations in Europe), or the new lending policies for banks following the world economic crisis in 2009, introduce new constraints on BN processes without modifying, in most cases, the structure of the network and, therefore, without introducing new agreements.

Agreement-level evolution. Agreement-level evolution implies changes in the set of agreements *AGR*. In particular, we focus on the insertion and deletion of an empty agreement, i.e. an agreement $agr_{b,a}$ without clauses ($CLA_{b,a} = \emptyset$), in which clauses can be added through clause-level evolution. From the continuous monitorability of act_a perspective, the establishment of a new agreement $agr_{b,a}$ is constituted by the insertion of an empty agreement $agr_{b,a}$ followed by the insertion of a new clauses (i.e., INS_PSC evolution type for each clause to be included). Similarly, the deletion of an established agreement is a sequence of clause deletion for each clause in the agreement, followed by the deletion of the resulting empty agreement.

We identify four types of agreement-level evolution:

1. *Insert empty provider-side agreement* (INS_PSA): it occurs when the actor act_a becomes provider for a new consumer act_b:
$$AGR^{to-be} = AGR^{as-is} \cup \{agr_{b,a}\}$$

2. *Insert empty consumer-side agreement* (INS_CSA): it occurs when the actor act_a outsources a process segment to another actor act_b:
$$AGR^{to-be} = AGR^{as-is} \cup \{agr_{a,b}\}$$

3. *Delete empty provider-side agreement* (DEL_PSA): it occurs when act_a cancels a business relationship with a consumer act_b:
$$AGR^{to-be} = AGR^{as-is} \setminus \{agr_{b,a}\}$$

4. *Delete empty consumer-side agreement* (DEL_CSA): it occurs when act_a cancels a business relationship with a provider act_b:
$$AGR^{to-be} = AGR^{as-is} \setminus \{agr_{a,b}\}$$

Agreement-level evolution occurs in many traditional virtual enterprise scenarios; new consumers may be discovered, or partners in the network can decide, for cost or quality reasons, to outsource part of their processes to an external party [4], [6].

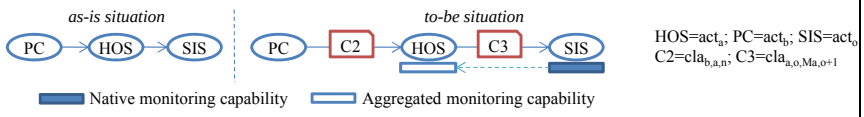
4 Algorithms for preserving continuous monitorability

When evolution occurs, the continuous monitorability of new and existing agreements may be disrupted. Hence, the continuous monitorability IT infrastructure of the actor act_a involved in an evolution type needs to undertake some corrective actions to reestablish the continuous monitorability of the agreements in which act_a is involved. In the following we describe algorithmically and by example such corrective actions for each evolution type.

01. $PROCEDURE :: INS\_PSC(cla_{b,a,n}, act_b, act_a)$ // $n=L_{b,a}+1$
02. $mcp_{a,m_a} = BuildCapability(cla_{b,a,n}, act_b, act_a)$
03. $SetMonitor(mcp_{a,m_a}, cla_{b,a,n}, act_a)$ // link the new capability to the new clause
04. END PROCEDURE

05. PROCEDURE:: $BuildCapability(cla_{c,p,l}, act_c, act_p)$
06. $blk^{bt}_{p,k_p} = \{blk: blk \in BLK_p \wedge refersTo(cla_{c,p,l}, blk)\}$ // identify the block to which clause refers
07. IF $bt = IN$ // block is executed internally
08. $mcp^{NAT}_{p,m_p} = CreateNatMcp(cla_{c,p,l}, act_p)$ //act_p (provider) creates native capability
09. ELSE // block is outsourced
10. $act_o = \{act: act \in ACT \wedge executedBy(blk_{p,k_p}, act)\}$ // identify outsourcer
11. $cla_{p,o,M_{p,o}+1} = Project(cla_{c,p,l}, act_p, act_o)$ // project the clause
12. $mcp_{o,m_o} = BuildCapability(cla_{p,o,M_{p,o}+1}, act_p, act_o)$ // call to external provider (recursive)
13. $mcp^{AGG}_{p,m_p} = CreateAggMcp(cla_{c,p,l}, mcp_{o,m_o}, act_p)$ // aggregate returned monitoring capability
14. END ELSE
15. return mcp_{p,m_p} // return the capability
16. END PROCEDURE

(a) algorithm



(b) example

Fig. 2. Monitorability preservation algorithm for evolution INS_PSC

Insert clause in provider-side agreement (INS_PSC). When a new clause $cla_{b,a,n}$, with $n=L_{b,a}+1$, is introduced, act_a must check whether the monitoring capability to match act_b's new monitoring requirements can be built on the fly. If the creation of a monitoring capability for a new clause is not successful, the actor can either decide to include the new clause in the agreement (the clause will not be continuously monitorable), or not to include the new clause in the agreement. The decision depends on the risk exposure policy of the actor, which is out of scope in this paper. In this paper, we assume that the creation of monitoring capabilities is always successful.

The steps to be followed by the provider act_a to build a monitoring capability for a new clause $cla_{b,a,n}$ are shown in the procedures in Fig. 2a. These use the four primitives *CreateNatMcp*(), *Project*(), *CreateAggMcp*(), and *SetMonitor*(), which capture the functionality of the continuous monitorability IT infrastructure as follows:

- *CreateNatMcp*(*cla, act*) represents the creation by the provider *act* of a new native monitoring capability for the continuous monitoring of the clause *cla* by its consumers. This involves the retrieval of the monitoring information required by consumers, e.g. through a native monitoring API or the instrumentation of the IT infrastructure on which the block to which the clause *cla* refers is executed;
- *Project*(*cla*, *act*, act_c) represents the projection of a clause *cla* towards an actor act_c made by the consumer *act* of a process provided by act_c. The projection triggers the establishment of a new clause cla_p, which is returned by the execution of the primitive. From the modeling perspective, after the execution of this primitive the predicate *projectsTo*(*cla*, cla_p) will evaluate to true;
- *CreateAggMcp*(*cla*, *mcp, act*) represents the creation by the provider *act* of a new monitoring capability mcp_n^{AGG} built as the aggregation of information retrieved from the capability *mcp*, exposed by one of the *act*'s providers;
- *SetMonitor*(*mcp*,*cla*,*act*) represents the creation made by the provider *act* of the mechanism that enables the consumer of the clause *cla*, to use the monitoring capability *mcp* to obtain the information required for monitoring the clause *cla*.

In Fig. 2a (line 5 onward), first act_a identifies the block to which the new clause refers. If such block is not outsourced, then the (native) monitoring capability to match the consumer's new monitoring information requirements is built by the provider, using the *CreateNatMcp*() primitive. If the block is outsourced, then act_a must (i) project the new clause towards the provider of the block and (ii) request the creation of the capability, which will be aggregated by act_a for guaranteeing the monitoring of the new clause to act_b. Note that the mechanism is recursively iterated till the outsourcer that can natively provide the information required for monitoring the new clause is found.

In our example (see Fig. 2b) PC and HOS may agree on a new clause, such as C2 (traceability of scan interpreters). PC, for instance, may be required by a new regulation for improving transparency towards patients to provide the identity of two different scan interpreters for every scan request. HOS identifies that the new clause refers to a process block that is outsourced to SIS and, therefore, projects the clause C2 on C3 and forwards the request for the new monitoring capability to SIS. The identity of the scan interpreters can be natively captured and made available by SIS to HOS. Therefore SIS exposes a native monitoring capability to HOS, who may apply a domain specific translation or integrate it with internal information before making it available, as an aggregated monitoring capability, to PC.

Insert clause in consumer-side agreement (INS_CSC). This case is not relevant from the point of view of act_a. The consumer act_a will rely, in fact, on the provider act_j to provide the monitoring capability that matches act_a's monitoring information requirements derived from the insertion of a new clause $cla_{a,j,n}$.

Delete clause in provider-side agreement(DEL_PSC). This case is also not relevant from the point of view of act_a, since the deletion of a clause does not introduce new monitoring information requirements for the actors in *CS*(act_a). In other words, there is no garbage collection made by the provider of the monitoring capabilities that are no longer used by consumers for monitoring established clauses. We argue, in fact,

that a monitoring capability may be reused in the future to satisfy the monitoring information requirements of new consumers.

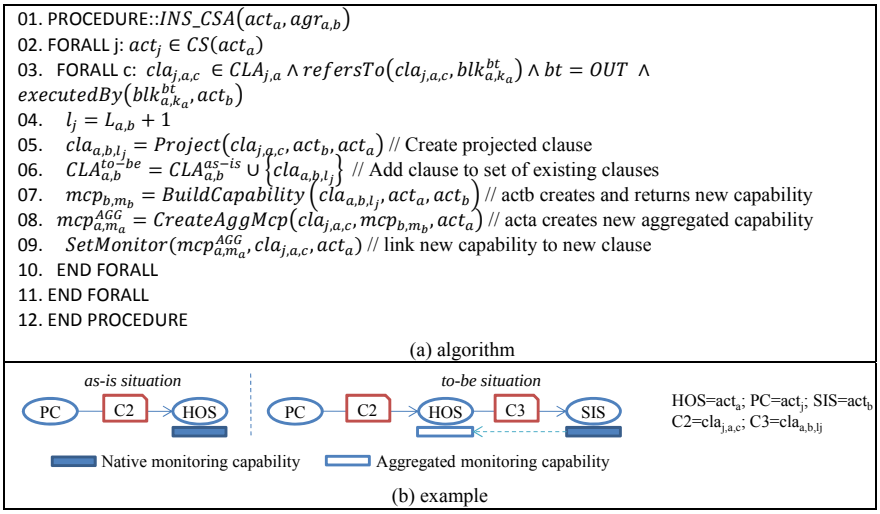


Fig. 3. Monitorability preservation algorithm for INS_CSA evolution

Delete clause in consumer-side agreement (DEL_CSC). The deletion of a clause $cla_{a,j,l_{a,j}}$ in which the actor acta appears as a consumer is allowed only if $cla_{a,j,l_{a,j}}$ is not a projection of any already existing clause. In other words, the deletion of $cla_{a,j,l_{a,j}}$ is allowed only if the set $\{cla: cla \in CLA_{a,j} \wedge projectsTo(cla, cla_{a,j,l_{a,j}}) \wedge act_j \in CS(act_a)\}$ is not empty.

In our example of Fig. 1, clause C3 cannot be removed, since it is the projection of C2 and HOS will not be able to let PC monitor C2 without the information obtained by SIS for monitoring C2. Clauses that represent a projection of existing clauses that act_a has with its consumers can only be deleted when the agreement in which they appear is removed afterwards (see evolution type DEL_CSA).

Insert empty consumer-side agreement (INS_CSA). When an actor act_a outsources part of a process to a new provider act_b, then act_a should coherently project the existing clauses in agreements with actors in $CS(act_a)$ towards act_b and modify its monitoring capability accordingly. In Fig. 3b, PC and HOS have agreed in the as-is situation on a clause similar to C2 (four-eyes principle on scan interpretations). Starting from the situation in which HOS executes internally the whole radiology process, HOS outsources part of the process to SIS. HOS needs to project C2 to regulate its relationship with SIS, i.e. creating the clause C3 that guarantees the four-eyes principle on the service provided by SIS. In the to-be situation, HOS has new information requirements for the continuous monitorability of C3, which should be matched by a suitable monitoring capability exposed by SIS, i.e. for the traceability of the scan interpreters. HOS will use such capability to update the capability exposed to GP, which now becomes aggregated.

The monitorability preservation algorithm for evolution type INS_CSA is shown in Fig. 3a. The algorithm uses the procedure *BuildCapability()* already defined for evolution type INS_PSC.

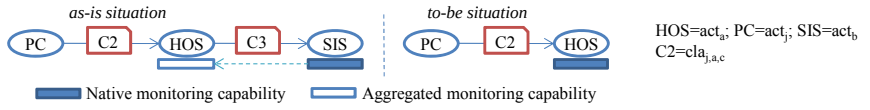
Insert empty provider-side agreement (INS_PSA). From the point of view of preserving continuous monitorability, this type of evolution does not imply any corrective actions by act_a. The creation of suitable monitoring capabilities is required from act_a only when the empty agreement will be filled in with new clauses (see INS_PSC evolution type).

Delete provider-side agreement (DEL_PSA). In this case an agreement between the provider act_a and one of its consumers act_b is cancelled. No specific actions should be taken in this case by act_a to preserve continuous monitorability.

Delete consumer-side agreement (DEL_CSA). In this case an agreement between the provider act_a and one of its providers act_b is cancelled. If the agreement is empty, then no specific actions should be taken by act_a to preserve continuous monitorability.

01. PROCEDURE::$DEL\_CSA(, agr_{a,b}, act_a, act_b)$
02. FORALL l: $cla_{a,b,l} \in CLA_{a,b}$ // for every clause still in the agreement $agr_{a,b}$
03. FORALL j: $act_j \in CS(act_a)$
04. FORALL c: $projectsTo(cla_{j,a,c}, cla_{a,b,l})$
05. $mcp^{NAT}_{a,m_a} = CreateNatMcp(cla_{j,a,c})$ // Create new native capability
06. $SetMonitor(mcp^{NAT}_{a,m_a}, cla_{j,a,c})$ // Link clause to new native capability
07. END FORALL
08. END FORALL
09. $blk^{OUT}_{a,k_a} = \{blk: blk \in BLK_a \wedge refersTo(cla_{a,b,l}, blk) \wedge executedBy(blk, act_b)\}$ //select outsourced block
10. $BLK^{to-be}_a = \{BLK^{as-is}_a \setminus blk^{OUT}_{a,k_a}\} \cup \{blk^{IN}_{a,k_a}\}$ // mark originally outsourced block as internal
11. END FORALL
12. END PROCEDURE

(a) algorithm



(b) example

Fig. 4. – Monitorability preservation algorithm for evolution DEL_CSA

If the agreement is not empty, the it contains clauses that are the projection of one or more clauses established between act_a and its consumers (see evolution type DEL_CSC). For each of this clauses $cla_{a,b,l_{a,b}}$, established by act_a with a provider act_b, the preservation of continuous monitorability is made possible only if act_a re-sources internally the block that was originally outsourced to act_b (and to which $cla_{a,b,l_{a,b}}$ refers). In our example (see Fig. 4b), when the agreement between SIS and HOS is deleted, it will still contain the clause C2, since it represents the projection of C3. HOS will first re-source internally the scan interpretation process. Then, HOS needs to check whether some of the clauses that it has established with its own consumers, e.g. C2 with PC, require information made available by SIS's monitoring capabilities for continuous monitoring. If this is the case, then HOS creates a new native monitoring capability that substitutes the capability originally obtained through

the aggregation of SIS's capability. The monitorability preservation algorithm for evolution type DEL_CSA in the generic case is shown in Fig. 4a.

After having discussed the mechanisms for the preservation of monitorability, we can now go back to the simplifying assumptions adopted in the modeling of business networks made in Section 3.1. We argue that the relaxation of assumptions 1 and 2 leads to a more complex notation, but it does not change the rationale behind our monitorability preservation mechanisms. Relaxing assumptions 3, 4, and 5 should result in more complex primitives for the creation of aggregated monitoring capabilities and projections of clauses. As a sample, the relaxation of assumption 4 implies that a block can be outsourced to more than one actor. In our example, in order to maintain the four-eyes principles, the scan interpretation may be outsourced by HOS to two different scan interpretation services, each of which provides a single scan interpretation. The projection of clauses needs to be updated is such a way that every clause of type C2 established by HOS with a consumer triggers the establishment of two different clauses, one for each scan interpretation service. The aggregation of capabilities should be modified in such a way that the capability exposed by HOS to its consumers combines the identity of the scan interpreters retrieved from the capabilities exposed by the scan interpretation services to which the service is outsourced. Improved mechanisms that account for the relaxation of our simplifying assumptions are target of future research.

5 Architecture of continuous monitoring infrastructure

From the definition of the continuous monitoring problem in BNs and the discussion of the mechanisms for preserving continuous monitorability in reaction to evolution, we derive the list of functional requirements for the Continuous Monitoring Infrastructure (CMon-I) of a business actor act_a in a BN (see Table 1). We distinguish between requirements relevant when act_a is a provider in the BN (see PRO in the second column of Table 1), a consumer in the BN (CON), or both. Requirements REQ1-2 derive from the need for act_a to provision monitoring capabilities to consumers and to access the capabilities of providers, whereas REQ3-8 derive from the need to preserve continuous monitorability in reaction to evolution of the agreements that involve act_a.

From the list of requirements, we derive a conceptual architecture for CMon-I. We present a two-level decomposition of such an architecture. The level-1 decomposition of the architecture is shown in Fig. 5. The required functionality to support continuous monitoring is clustered in several modules. Fig. 5 shows the requirements that are implemented by each module. The monitoring client (MC) retrieves the capabilities and monitoring data from the actors in the provider set. Monitoring data are provided by the monitoring service (MS) module. The evolution manager (EM) detects changes in the BN and instructs the monitoring capability builder (MCB) to create a new monitoring capability, using, if necessary, the provider capabilities retrieved through MC. Note that a business actor in a BN, i.e. HOS in Fig. 1, acts at the same time as a provider and a consumer, whereas business actors at the edges of the network, i.e. PC

or SIS in Fig. 1, participate only as consumers or providers of business processes. In the latter case, the architecture, shown in Fig. 5, can be simplified to include only those modules, which satisfy the requirement that are relevant for that role (as indicated in Table 1).

Table 1. – Functional requirements for CMon-I

REQ1	PRO	CMon-I allows the provisioning of the monitoring capabilities referring to the process that act_a is contributing to the actors in $CS(act_a)$, covered by the primitive *SetMonitor*() of Section 4.
REQ2	CON	CMon-I allows act_a to access the monitoring capabilities of actors in $PS(act_a)$
REQ3	both	CMon-I allows act_a to detect the evolution of the agreements in the BN, either at agreement- or clause-level
REQ4	both	CMon-I has access to the agreements (including process specifications) that act_a has established with other actors in the BN, either as a consumer or a provider
REQ5	PRO	CMon-I allows act_a to build native monitoring capabilities to match the monitoring information requirements of actors in $CS(act_a)$ [see the primitive *CreateNatMcp*() of Section 4]
REQ6	PRO	CMon-I allows act_a to build aggregated monitoring capabilities to match the monitoring information requirements of actors in $CS(act_a)$ as aggregation of monitoring capabilities of actors in $PS(act_a)$ [see the primitive *CreateAggMcp*() of Section 4]
REQ7	both	CMon-I allows act_a to project clauses across the network in reaction to evolution [see the primitive *Project*()of Section 4]
REQ8	both	CMon-I allows act_a to detect if a new monitoring capability needs to be created in reaction to evolution. In this case, CMon-I starts the creation of a new, aggregated or native, monitoring capability

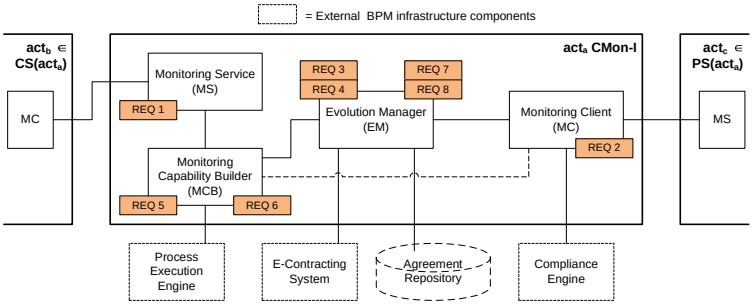


Fig. 5. Level 1 decomposition of CMon-I architecture

We make the assumption that CMon-I is situated within a generic Business Process Management (BPM) infrastructure, on which the actor runs its processes. Such BPM infrastructure is constituted by a process execution engine, an E-Contracting system, which can be triggered for the projection of clauses, an Agreement Repository, which stores the agreements established by an actor, either as a provider or a consumer, and a Compliance (Auditing) Engine, which checks the satisfaction of clauses during process execution according to the information captured through monitoring capabilities of actors in $PS(act_a)$.

Our decomposition of the CMon-I architecture is iterated until we identify only modules that implement at most one of the requirements, so that a clear separation of

concerns is reached in which each module provides functionality to satisfy one specific requirement. Hence, the level 2 decomposition, i.e. the internal conceptual architecture of the CMon-I modules, is shown only for those modules that implement two or more requirements in the level 1 decomposition, i.e. MCB and EM (see Fig. 6).

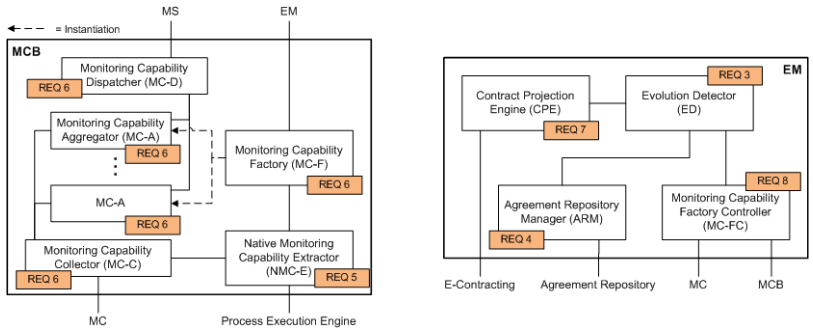


Fig. 6. Conceptual architecture of MCB and EM

Since new monitoring capabilities should be created when evolution occurs in the BN, EM commands the Monitoring Capability Factory (MC-F) to create a new capability when required from a detected evolution. In response to requests from EM, MC-F instantiates a new Monitoring Capability Aggregator (MC-A), which implements the aggregation of native or external, i.e. from actors in $PS(act_a)$, capabilities to provide the monitoring capability required by the actors in $CS(act_a)$. The instantiation of a new MC-A can be seen as the result of the execution of the primitives *CreateNatMcp*() or *CreateAggMcp*() introduced in Section 4.

A separate module (NMC-E) is required to extract native monitoring capabilities from the process engine on which act_a executes the processes contributed to the BN. Most of the commonly used workflow engines provide a native monitoring API for inspecting the execution of process instances. This is the case, for instance, of the BPELMonitor API for the Open-ESB BPEL Open source engine (see http://wiki.open-esb.java.net/Wiki.jsp?page=BPELMonitor), or the YAWL Observer interface for the YAWL engine [10]. Similar native monitoring interfaces are also available in commonly used ERP packages. NMC-E sits on top of such native API to extract the information required for monitor a clause.

For what concerns EM, the ARM provides access to the external Agreement Repository. ED is responsible for analyzing agreements, detecting the evolution of the BN. ED may either poll ARM for retrieving new agreements, or new agreements may be proactively pushed by ARM to ED. ED runs the business logic of the mechanisms discussed in Section 3 and may trigger the projection of contracts, implied by the execution of the *Project*() primitive introduced in Section 3, and the construction of new monitoring capabilities, e.g. in reaction to INS_PSC evolution. CPE is responsible of managing the projection of clauses and interacts, therefore, with the external E-contracting system. MCF-C controls the MC-F in MCB for requesting new monitoring capabilities. MCF-C is also connected to MC, in order to establish access to the monitoring capabilities provided by providers in $PS(act_a)$.

6 Implementation

As a proof of concept, the continuous monitoring approach has been implemented in the PROXE (PROcesses in a Cross-organizational Environment) system. PROXE is based on the Business Process Web Services (BP-WS) framework, the aim of which is to 'open-up' black-box Web Services. BP-WS services expose the enclosed business processes through a set of standard interfaces, so that service consumers can monitor, control, and synchronize with the service execution progress of the service provider [11].

Fig. 7 shows the implementation architecture of the PROXE system (excluding those components and interfaces that are not relevant for the work presented in this paper). The teleradiology process has been used as a test scenario to validate the continuous monitoring approach implemented in the system. As can be seen in the figure, three parties (GP, HOS, and SIS) are included and their systems are connected through the ACT and MON interfaces. The ACT interface is used to invoke the service and is handled by the invoker module. The MON interface is used to monitor service execution and is backed by the CMon-I modules.

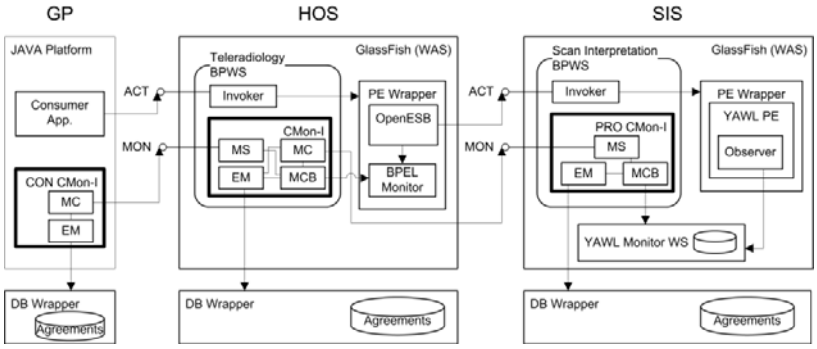


Fig. 7. Implementation Architecture

The process that is performed by HOS, i.e., the teleradiology process as shown in Fig. 1, is executed on a BPEL process engine (OpenESB BPEL Engine on the GlassFish Web Application Server). The scan interpretation part of this process is however outsourced and is executed on the YAWL workflow system [10] by SIS. Both process engines have their own mechanisms to expose monitoring information. The BPEL Monitor is an addition to the BPEL Engine and can be directly accessed through its API. YAWL exposes process events to a, so called, ObserverGateway. This ObserverGateway passes the events to the YAWL Monitor Web Service, which subsequently stores the relevant information contained in an event into a database. On request by the CMon-I, e.g., as a response to a query posed by HOS to retrieve the scan interpreters identity, the YAWL Monitor Web Service retrieves the desired monitoring information from the database.

Each party has its own database, in which the agreements and clauses are stored. Any component within the BPWS, i.e., Teleradiology BPWS and Scan Interpretation

BPWS, can access the database. This is required to validate calls to the BPWS against the contract. For the CMon-I component, access is required as explained in the previous section. The process engines and databases are part of the BPM infrastructure, as defined in Section 5. The e-contracting systems and compliance engines, which are also part of the BPM infrastructure, have not been included in the PROXE system as they do not address the core issues of continuous monitoring.

SIS is only a provider in the business network, so SIS has only native monitoring capabilities, indicated through the single connection of CMon-I/MCB to the YAWL Monitoring WS. HOS acts as both a consumer (of SIS) and provider (to GP). The CMon-I/MC of HOS is therefore connected to the MON interface of SIS (and will aggregate the monitoring information retrieved through this interface if required by GP) and CMon-I/MCB is connected to the BPEL Monitor (to provide native monitoring capabilities to GP. GP acts as a consumer only and retrieves the required monitoring information of HOS through the exposed MON interface. The connections to the databases are used to access the stored agreements and clauses.

As an example of agreement evolution, we use the outsourcing by HOS of the scan interpretation to SIS (evolution type INS_CSA, see Figure 3). For the implementation, this type evolution implies that the retrieval of the identities of scan interpreters is moved from the BPEL Monitor of HOS to the YAWL Monitor WS of SIS. YAWL provides two API methods to acquire the required information: getUserWhoIsExecutingThisItem():String and get_whoStartedMe():String. HOS's CMon-I/MCS transforms the returned results into terminology that is expected by GP.

In the current PROXE system, the creation of a monitoring capability related to a clause is done manually. For example, discovering that the four-eyes principle can be supported through the identification of the persons who performed the specific scan interpretation task (and that they should not be the same person) and that this information, in turn, can be retrieved through some specific API calls, is done manually. A (semi-)automatic translation is considered future research.

7 Related Work

Business process monitoring has been largely investigated under the labels of Business Process Intelligence (BPI) [12] and Business Activity Monitoring (BAM) [13]. 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 [14] and normative reasoning applied in the context of business process management [15]. Process mining does not represent a suitable 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.

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 [16] or the design of monitoring engines compliant with Web service technology [17]. An approach for the controlled evolution of Web service contracts is discussed in [18], 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 [19]. Such a methodology, however, is applied only within the domain of the orchestrator of the collaboration and considers the services invoked by a business process as black boxes.

Monitoring in the CrossFlow project [6] concerns only information on the progress of an outsourced process and basic process variables, which are accessible at specific monitoring points specified in the contract. Moreover, CrossFlow considers 1:1 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 [4]. 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 [20]. Monitoring requirements and evolution of a formed network are, however, not considered. The design of cross-organizational processes with evolving requirements is tackled in [21], but without explicit focus on monitorability requirements.

Similarly to the monitoring capabilities defined in this paper, the E-Adome workflow engine [3] 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.

8 Conclusions

This paper analyzes the issue of continuous monitorability of cross-organizational business processes. In particular, we discuss the case of the evolution of agreements in a business network and show how the continuous monitoring IT infrastructure should adapt to preserve the monitorability of agreements. The paper formally describes algorithms for restoring the continuous monitorability of agreements in reaction to their evolution. We also discussed the PROXE system, which implements the requirements for continuous monitorability derived from our modeling of evolving business networks.

Future work will concern the refinement of our model of business networks, relaxing the assumptions made in Section 3, and the analysis of alternative forms of evolution for BNs. We also plan to consider, within the PROXE system, template-based agreement lifecycles and to link monitoring with control actions to be undertaken when the compliance to existing clauses in not verified.

References

1. Alles, M. G., Kogan, A., Vasarhely, M. A.: Feasibility and Economics of Continuous Assurance. Auditing: Journal of Practice and Theory 21(1) (2002)
2. Coderre, D.: Continuous Auditing: Implications for Assurance Monitoring and Risk Assessment. (2005)
3. Chiu, D. K. W., Karlapalem, K., Li, Q., Kafeza, E.: Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment. Distrib. Parallel. Dat. 12, 193-216 (2002)
4. Grefen, P., Eshuis, R., Mehandijev, N., Kouvas, G., Weichart, G.: Internet-based support for Process-Oriented Instant Virtual Enterpises. IEEE Internet Comput Nov/Dec, 30-38 (2009)
5. van Heck, E., Vervest, P.: Smart Business Networks: How the network wins. Communications of the ACM 50, 28—37 (2007)
6. Grefen, P., Aberer, K., Hoffner, Y., Ludwig, H.: CrossFlow: cross-organizational workflow management in dynamic virtual enterprises. Comput. Syst. Sci. & Eng. 5, 277--290 (2000)
7. Vonk, J., Wang, T., Grefen, P., Swennenhuis, M.: An Analysis of Contractual and Transactional Aspects of a Teleradiology Process. Beta Technical Report 263, Eindhoven University of Technology, Eindhoven (2008)
8. Mendling, J., Reijers, H., van der Aalst, W. M. P.: Seven Process Modeling Guidelines (7PMG). Information and Software Technology 52(2) (2010)
9. Hall, J. A., Liedtka, S. T.: The Sarbanes-Oxley Act: Implication for large-scale IT outsourcing. Communications of the ACM 50(3), 95-100 (2007)
10. ter Hofstede, A., van der Aalst, W. M. P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its support environment. Springer (2010)
11. Grefen, P., Ludwig, H., Dan, A., Angelov, S.: An analysis of web services support for dynamic business process outsourcing. Informtation and Software Technology 48, 1115-1134 (2006)
12. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.-C.: Business Process Intelligence. Computers in Industry 53, 321-343 (2004)
13. McCoy, D. W.: Business Activity Monitoring., Gartner Group Research Report ID LE-15-9727 (2002)
14. van der Aalst, W. M. P., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, E.: Conformance checking of Service Behavior. ACM TOIT 8(3) (2008)
15. Sadiq, S., Governatori, G., Namiri, K.: Modeling control Objectives for Business Process Compliance. In : Proc. 5th BPM Conference, pp.149-164 (2007)
16. Skene, J., Raimondi, F., Emmerich, W.: Service-Level Agreements for Electronic Services.. IEEE Transactions on Software Engineering (forthcoming) (2010)
17. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for WS-BPEL. In : Proc. WWW 2008 (2008)
18. Andrikopoulos, V., Benbernou, S., Papazoglou, M.: Evolving Services from a Contractual Perspective. In : Proc. CAiSE 2009, pp.290-304
19. Orriens, B., van den Heuvel, W.-J., Papazoglou, M.: On The Risk Management and Auditing of SOA Based Business Processes. In : Proc. 3rd Int. ISoLA Symposium, pp.124-138 (2008)
20. Mehandjiev, N. D., Stalker, I. D., Carpenter, M. R.: Recursive Construction and Evolution of Collaborative Business Processes. In : BPM 2008 Workshops, pp.573-584 (2008)
21. Desi, N., Chopra, A. K., Singh, M. P.: Amoeba: A methodology for modeling and evolving cross-organizational business processes. ACM TOSEM 19(2), Article 6 (2009)