

# Towards Forensic Data Flow Analysis of Business Process Logs

Rafael Accorsi, Claus Wonnemann, Thomas Stocker  
*Department of Telematics*  
*University of Freiburg, Germany*  
{accorsi,wonnemann,stocker}@iig.uni-freiburg.de

## Abstract

This paper presents RECIF, a forensic technique for the analysis of business process logs to detect illegal data flows. RECIF uses *propagation graphs* to formally capture the data flow within a process execution. Abstracting away from the concrete traces, propagation graphs are analyzed with *extensional data flow policies* that denote what – instead of how – relevant industrial requirements, e.g. Chinese Wall and separation of duty constraints, are to be achieved. An example and the corresponding runtime figures demonstrate the feasibility of the approach.

## Keywords

Business process forensics; Data flow reconstruction; Data flow policies.

## I. INTRODUCTION

### A. Problem Context and Contribution

A business process specification (so-called *workflow*) is a high-level piece of software that describes an automated procedure within an enterprise, such as open a bank account, process a loan application, or update a patient record. A recent, representative survey conducted in major European companies shows that up to 70% of business processes are automated [35]. The adoption of workflows is expected to grow even further with the advent of cloud computing and the provision of distributed, configurable processes to attend to clients' needs "on-demand".

Driven by the development towards so-called "process-aware enterprise information systems" and the simultaneous need to adhere to regulatory, security, and contractual requirements, tools for detecting violations of these properties are gaining on momentum. A particularly relevant kind of incident concerns information leaks [14], which roughly speaking correspond to the violation of a secrecy property. Here, information meant to be available to only a specific domain (set of subjects) is transferred, in the course of the business process execution, to another domain.

There are generally two kinds of leaks: *explicit* leaks denote direct flows of data among (subjects in the) domains, e.g. a message is sent from one subject to another. *Implicit* leaks denote indirect flows of information among the domains, in that a domain can infer information about another over "covert-channels", such as time and control flow [22]. While both kinds of leaks are relevant, due to their probative force, evidence with regard to explicit leaks appear to be more relevant for forensic investigation of enterprise systems [12].

Despite of this relevance, the current forensic toolkit provides only support for investigation and evidence generation at the level of infrastructure, e.g. services, operating systems, virtual machines, and databases (see Section I-B). In consequence, the forensic analysis of business process executions is not supported. We firmly believe that by addressing this level – as opposed to only the infrastructure – one provides complementary indication as to how a leak comes to happen and which activity and system subject are to be accounted for in case of non-compliance.

This paper presents RECIF, a forensic technique for the analysis of workflow logs. RECIF (standing for “reconstruct information flows”) combines *data flow reconstruction* and *extensional policies* to identify possible data leaks that occurred in workflow executions, thereby substantiating or refuting a suspicion. Specifically, RECIF employs propagation graphs (PG) as an abstraction to represent the data flows. PG are directed labeled graphs that are extracted from business processes logs: each node corresponds to an event (denoting a workflow activity) and an edge stands for the flow of a piece of data (label) from one activity to another. Extensional policies are based on mandatory access control policies [16] which define general data flow constraints. In contrast to intensional specification styles [28], e.g. access control lists and other discretionary models, extensional policies provide a high-level but powerful method to specify analysis requirements. In particular, this specification styles provides for *complete* description of the allowed/forbidden flows [7].

Given that, the main contribution of this paper is an approach for a posteriori analysis of logs to uncover data transfers across domains that violate a security policy. Specifically, this paper:

- Brings forward RECIF, a forensic approach for automated detection of data leaks in workflows. To model data flows, we introduce a data flow abstraction called propagation graphs, presenting the algorithms and data structures needed the reconstruction of flows and analysis.
- Proposes a preliminary policy language to capture data flow requirements, i.e. isolation requirements for data items. In particular, we show that it can capture a number of industrially relevant access control requirements, e.g. separation of duties and conflict of interest.
- Focusing on separation of duty constraints, illustrates and evaluate RECIF in an example and report on the runtime figures and insights obtained with the prototypical realization. Being an “a posteriori” analysis technique, the runtime figures serve only to indicate the feasibility of tool-support.

Overall, our goal is to advance business process forensics with the development of sound, well-founded, and expressive analysis techniques and corresponding tool-support for enterprise computing and workflows. In this setting, the presented version of RECIF is just a first step towards more complex analysis techniques. In particular, while we believe that PG are sufficient for the identification of explicit leaks and analysis of usual enterprise properties, the formal foundation might be too coarse to cope with, e.g., implicit information leaks. For this kind of analysis, program dependency graphs [19] and Petri nets [3] might be more suitable formalisms.

## B. Related Work

The enterprise meta-model provides a suitable abstraction to classify and compare the related work with RECIF. This meta-model encompasses three layers: the *business* layer contains the

business-oriented artifacts, such as business processes and information objects. The *application* layer contains, among others, the web services and other logical components necessary to execute the business processes. The *technical* layer contain nodes, i.e. a hardware and software set providing the functionality necessary to process a particular service.

Forensic techniques focus on the application and technical layers. Techniques for the application layer fall into the scope of network forensics [27], focusing on the choreography and usage of distributed web services [25]. Gunestas et al. introduce “forensic web services” that can securely maintain transaction records between web services [18]. Chandrasekaran et al. develop techniques for inferring sources of data leaks in document management systems [13]. The technical layer is the traditional terrain of computer forensics. In database systems, forensics attempt to detect, e.g., leaks in query answering [10], tampering attempts [26] and measure retention [29]. New aspects arise from the increasing virtualization of operating system abstractions [8], as well as the live evidence acquisition [20].

The analysis of business process is by now a mature research field in business informatics. The focus lie, however, on preventive methods for detecting flows [3], [5], [6], [17], [30], [31]. Detective methods build upon *process mining*, i.e. the reconstruction of business process models from log data [4]. Process mining delivers models of a process’s functionality, for instance, to check the structural conformance of the implemented business process with the intended business process or to identify performance bottlenecks. In contrast to the propagation graphs used by RECIF, reconstructed process models do not account for data flows. In fact, auditing in the context of process mining one means checking whether a set of business processes is compatible in that it does not exhibit, e.g., deadlocks and livelocks [32]. Forensic analysis is not readily supported.

The use of propagation graphs to model dataflows stems from [23]. Livshits et al. focus on the inference of specifications by means of PG. In contrast, we take PG as an abstraction to reconstruct the data flows and automatically analyze a process. Compared to trees, an alternative abstraction for data flows, PG can capture activities in which several pieces of data are combined into one composite. The use of extensional policies to analyze business process executions has not been previously considered in previous work. RECIF proposes a multi-level security system in which policies build a lattice of security levels, as in [16]. This makes the approach similar to [15]. However, the proposed model focuses on the sounds delegation of tasks in business process and not on the verification of business process.

### C. Paper Structure

Section II introduces the abstract system model assumed in RECIF and Section III defines the language for and the semantics of data flow policies. Section IV presents the analysis algorithms and provides a formal definition of propagation graphs. Section V exemplifies the use of RECIF by means of a simple example on separation of duty constraints. Section VI discusses a number of issues with regard to RECIF. Section VII concludes the paper and outlines a generalization of the approach towards more expressive requirements.

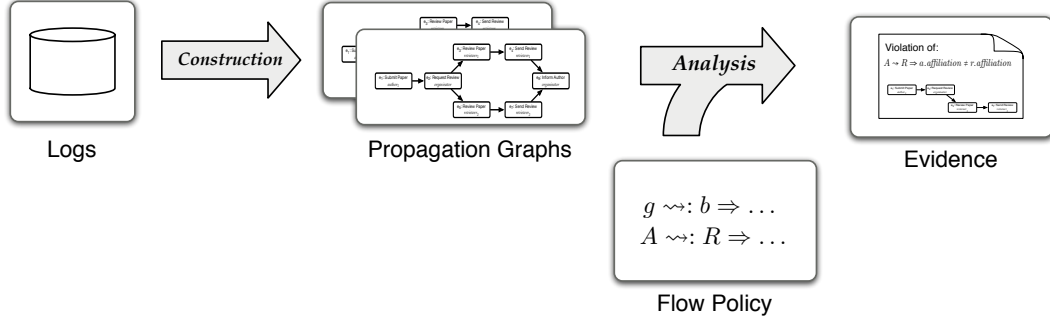


Figure 1. Overview of RECIF.

## II. RECIF OVERVIEW AND CHARACTERIZATION OF DATA FLOWS

RECIF follows the steps in Fig. 1. The logs contain the execution *traces* of a business process and provide the basis for the analysis. The data in the logs is organized entry-wise, where every entry stands for the execution of a business process activity and contains associated attributes such as the execution time, the processed data items and a subject identifier. For the analysis, RECIF extracts propagation graphs from the logs which represent the data flows that occurred within a particular execution. The resultant graphs are then analyzed against data flow policies, generating evidence (and counterexamples) as to whether or not the policies have been violated. Here, “evidence” should be seen as an indication of a violation (or its absence); it does not carry the connotation of a legal evidence in the sense of [21].

### A. Log File Format and Security

RECIF uses logs that are compatible with MXML, a standardized log format for business processes [33]. Each entry of a MXML exhibits the following fields (see Table II for an excerpt)

```
Case Timestamp Activity_ID Originator Input Output
```

where *Case* denotes the particular workflow instance; *Timestamp* the timepoint in which the activity happens; *Activity\_ID* the name of the activity; *Originator* is the system subject that triggers the activity; *Input* is a set of data consumed by the activity; *Output* is the set of data produce by the activity.

Besides the compact notation, the advantage of using MXML is that there are mappings from the majority of enterprise log formats used by business process systems, such as SAP, Oracle and Sage, thereby enabling RECIF to be deployed with a large variety of process systems. Furthermore, MXML serves as a basis for other analysis tools, e.g. ProM, which we plan to employ in the future. XES is an alternative log format to eventually replace MXML [34]. However, its adoption is limited at the moment and, for the forensic analysis pursuit in RECIF, XES does not seem to provide tangible advantages.

To be meaningful and contribute to the overall probative force of an evidence in court, the evidence generated by RECIF must be ideally drawn from authentic, complete logs. Hence, an

underlying assumption is that the enterprise systems enact the use of secure logging to provide for provably authentic logs. To this end, several secure logging exist [1], whereas the BBox is designed for highly-distributed enterprise systems [2].

### B. Characterizing Data Flows

Workflows are structured sequence of *activities*, whereas each activity can be either an automated task (e.g. applications and services) or require explicit human interaction (e.g. the creation of documents and approval of loan). The execution of an activity by the business process management system generates an *event*, which is in turn recorded as an *entry* in the log. While the distinction between activities and events is necessary in order to distinguish the business from the technical layers of an enterprise system, below we use these terms interchangeably.

The different formalisms that are used for workflow specification, such as BPEL, BPMN and Event-driven Process Chains, share a “core” of basic control flow operators, including parallelism, conditionals, and synchronization. These specification languages are, in essence, equally powerful (i.e. Turing-complete). The execution of a workflow is coordinated by an execution engine which triggers activities, synchronizes their interaction and communication, and writes log data. The log data encompasses the set of entries that builds the basis for the analysis considered in Section IV.

While the shape and formalism used to capture a workflow is, from the viewpoint of analysis, irrelevant, it is important to define the visible pieces of information (entries) stored in the log, i.e. the *abstract system model*. A workflow is a tuple  $\mathcal{W} = (A, F)$ , where  $A$  is the set of *activities* and  $F$  a specification of the control flow, i.e. the way in which the individual activities  $a_i \in A$  are carried out. During an execution instance (so-called *case*), a finite sequence of *events* is generated by the execution engine and stored in the log file as a trace  $T_{\mathcal{W}} = \{e_1, \dots, e_n\}$ . The log file that contains set of all executed traces of a workflow  $\mathcal{W}$  is denoted as  $L_{\mathcal{W}}$ .

Assuming a set  $S = \{s_1, \dots, s_n\}$  of subjects and a set  $D$  of data items, each activity execution  $e_i \in T$  in a case is associated with exactly one *originator*  $s \in S$  and takes an *input*  $D_i \subseteq D$  to produce an *output*  $D_o \subseteq D$ .<sup>1</sup> Given an event  $e$ , corresponding attributes are accessed via dot-notation, which can be seen as projection functions:  $e.originator \rightarrow S$  denotes the originator of the activity execution  $e$ ;  $e.input \rightarrow \wp(D)$  the input data item;  $e.output \rightarrow \wp(D)$  the output data item;  $e.activity \rightarrow A$  the an activity identifier; and  $e.timestamp \rightarrow \mathbb{N}$ .<sup>2</sup>

Carrying it over to a security terminology, events stand for accesses *rights* to *objects* (*Input*, *Output*), where the originator denotes the corresponding *subject*. The distinction between input and output is central for the reconstruction of the data flows produced by the sequence of activities: given a log file, with the event data flow relation it is possible to trace the relationship between the inputs and outputs of the activities and build the corresponding propagation graph denoting the data flow between events.

**Definition 1:** Let  $T_{\mathcal{W}} \in L_{\mathcal{W}}$  be a trace of a workflow  $\mathcal{W}$ ,  $\prec$  denotes the order of events inside  $T$ :  $e_i \prec e_j$  iff  $e_j.timestamp < e_i.timestamp$ , i.e.  $e_i$  occurred strictly before  $e_j$ . An **event data flow** between  $e_i, e_j \in T$  is defined as  $e_i \rightarrow e_j \Leftrightarrow e_i \prec e_j$  and  $e_i.Output \cap e_j.Input \neq \emptyset$ .  $\dashv$

<sup>1</sup>In some situations  $D_i = D_o$ , e.g. when an activity declassifies a set of data items without producing new items.

<sup>2</sup>Similar to Unix systems, we assume the representation of timestamps as natural numbers  $t \in \mathbb{N}$ .

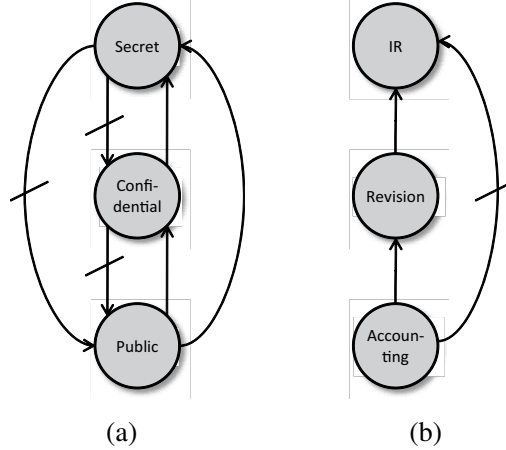


Figure 2. Exemplary flow policies.

A data flow from a *subject*  $s_1$  to a subject  $s_2$  happens if there exists a sequence of activities in  $T_i$  in which a)  $s_1$  and  $s_2$  act as originators and b) are connected over the corresponding data flow relation  $\rightarrow$ . Formally:

*Definition 2:* Let  $\rightarrow^+$  be the transitive closure of  $\rightarrow$ . A subject data flow, or simply **data flow**, from  $s_i$  to  $s_j$  is defined as  $s_i \rightsquigarrow s_j \Leftrightarrow \exists e_m, e_n \in T$  such that  $e_m \rightarrow^+ e_n$  and  $e_m.originator = s_i, e_n.originator = s_j$ . Extending this notion to subject groups, let  $S_i, S_j \subseteq S$  and  $S_i \cap S_j = \emptyset$ . The **group data flow** relation is defined as  $S_i \rightsquigarrow S_j \Leftrightarrow \exists s_1 \in S_i \wedge s_2 \in S_j, s_1 \rightsquigarrow s_2$ .  $\dashv$

Group data flows are particularly useful to model the notion of roles, thereby making it possible to reason about Chinese wall and separation of duty constraints. These kinds of requirements are very important in enterprise systems, as we report in Section III-C.

### III. EXPRESSING SECURITY REQUIREMENTS AS DATA FLOW POLICIES

A data flow policy  $P$  specifies which (sets of) data items can be exchanged among which (groups of) subjects. The underlying security model assumes that subjects are assigned security classifications from a set  $\mathcal{C}$  of security domains. A security domain denotes the rights and privileges of principals to obtain or manipulate information. In doing so, the actual policy specifies among which security domains information may or must not be exchanged, respectively. The function  $sc : S \rightarrow \mathcal{C}$  assigns subjects to security domains.

Fig. 2(a) shows an example of a multi-level security policy with three domains that are arranged according to a lattice  $Secret \succ Confidential \succ Public$ . A crossed arrow indicates that there must not flow information. Information may flow only upwards this lattice and thus cannot leak to principals with a lower clearance.

However, it is insufficient to state that some information may or may not flow from one principal to another. Rather, the exact path that some information has to take through the organization must be prescribed, for instance, to ensure that financial statements are cleared by the revision department before being published to investors. A corresponding information flow policy is depicted in Fig. 2(b): information coming from the *Accounting* domain must not flow directly to the *IR* department, but has to pass through *Revision*.

```

<Policy>      ::= <Rule> | <Rule>, <Policy>
<Rule>       ::= <Restriction>  $\Rightarrow$  <Exception>
<Restriction> ::= true | <FlowRel>
<Exception>  ::= false | <FR-DNF>
<FR-DNF>    ::= (<ConClause>) | (<ConClause>)  $\vee$  <FR-DNF>
<ConClause> ::= <FlowRel> | <FlowRel>  $\wedge$  <ConClause>
<FlowRel>   ::= <Domain> $\rightsquigarrow$ <Domain>
<Domain>    ::=  $c \in \mathcal{C}$ 

```

Figure 3. BNF grammar of the policy language.

With this security model at hand, the following defines the syntax and the semantics of a policy language to capture these requirements, and provides a few examples of industrial requirements.

### A. Policy Syntax

Intuitively, a data flow policy is a set of rules  $P = \{r_1, \dots, r_n\}$ , where each rule  $r \in P$  has the form  $Restriction \Rightarrow Exception$ . *Restriction* specifies a legitimate data flow between (groups of) subjects (see Definition 2). *Exception* is a logical concatenation of data flow relation and attribute evaluation in disjunctive normal form. An exception thus defines legitimate flows that contradict the specific *Restriction*. Attributes are accessed with the dot-notation used for activity, thereby allowing the expression of complex security requirements.

Fig. 3 gives the grammar for these policies in Backus-Naur-Form (BNF). A policy rule has the form  $Restriction \Rightarrow Exception$ . *Restriction* is a flow relation  $source \rightsquigarrow target$ , which specifies that information must not flow from domain *source* to domain *target* ( $source, target \in \mathcal{C}$ ). *Exception* is a logical combination of flow relations in disjunctive normal form which defines legitimate flows that might contradict *Restriction* (as in the example in Fig. 2(b)).<sup>3</sup>

Using the grammar in Fig. 3, the policy in Fig. 2(a) is formalized as

```

Secret $\rightsquigarrow$ Confidential  $\Rightarrow$  false,
Confidential $\rightsquigarrow$ Public  $\Rightarrow$  false,
Secret $\rightsquigarrow$ Public  $\Rightarrow$  false

```

while the policy in Fig. 2(b) is formalized as

```

Accounting $\rightsquigarrow$ IR  $\Rightarrow$ 
  (Accounting $\rightsquigarrow$ Revision  $\wedge$  Revision $\rightsquigarrow$ IR).

```

### B. Policy Semantics

Besides the extensional specification of flow relations, which denote whether data may flow between domains, a policy defines the patterns of system actions that constitute information transmission. We refer to this part of the policy as *flow semantics*.

The RECIF system currently supports the specification and verification of data flow requirements, i.e. constraints on the explicit information flow. Here, the flow semantics is trace-based and fairly straightforward: there is an data flow from domain  $c_1$  to domain  $c_2$ , if, and only if,

<sup>3</sup>Note that we overload the  $\rightsquigarrow$  relation, in that now relates security domains rather than subjects. However, since domains are in fact characterized by a (possibly empty) set of subjects, we do not expect it to cause a problem.

there is a data item which has been modified by an activity from  $c_1$  and is subsequently read by an activity from  $c_2$ . Assuming that the set of activities  $A$  corresponds to the events  $e_i$  recorded in the log traces, the flow semantics is defined as follows.

*Definition 3:* Let  $\mathcal{C}$  be a set of security domains and  $\mathcal{W} = (A, F)$  be a workflow,

$$\begin{aligned} \forall c_1, c_2 \in \mathcal{C} : c_1 \rightsquigarrow c_2 \Leftrightarrow & \exists (e_1, e_2 \in A), \exists (s_1, s_2 \in S) : \\ & (e_1.originator = s_1) \wedge (e_2.originator = s_2) \wedge \\ & (sc(s_1) = c_1) \wedge (sc(s_2) = c_2) \wedge \\ & ((c_1.output \cap c_2.input) \neq \{\}) \wedge (e_1 \prec e_2), \end{aligned}$$

where  $sc$  assigns a security domain to a subject (see above),  $c.input$  is a set of data items required by a domain  $c_i \in \mathcal{C}$ , and  $c.output$  is a set of data items produced by a domain  $c_i \in \mathcal{C}$ . Formally,  $d \in c_i.input$  iff  $\exists e \in A, \exists s \in S : d \in e.input \wedge sc(s) = c_i \wedge e.originator = s$ . The set  $c.output$  is defined in a similarly for the set of data items produced by a domain.  $\dashv$

The resultant policy language is similar to the tainting mode in the Perl programming language. It is sufficiently expressive to capture violations, such as failure to “clear” information or illicit propagation of some data item. For a more fine-grained analysis and to capture other kinds of information flows, more sophisticated semantics and analysis algorithms are needed and are subject of future work (see Section VII).

### C. Policy Examples

In abstracting away from access right and focusing on the flow of data between subjects, data flow policies capture security requirements in an extensional manner [28]. To illustrate its expressive power, the following formalizes two relevant requirements as data flow policies.

**Separation of duties.** Generally, separation of duties (short SoD) is the concept of having more than one subject required to fulfill a task, thereby helping to reduce the potential damage from the actions of one person. SoD is already well-known and vastly employed in financial accounting systems [9]. Companies in all sizes understand not to combine roles such as, for instance, receiving checks (payment on account) and approving write-offs, depositing cash and reconciling bank statements, approving time cards and have custody of pay checks. In contrast, SoD is fairly new in computer systems and a consequence of compliance requirements, such as SOX and HIPAA, which to a large extent rely on SoD.

To formalize SoD constraints as data flow policies, subjects are grouped according to their acting role inside the workflows. A simple policy would forbid flow from one group  $G_1$  to an other group  $G_2$  if the subjects  $s_i \in G_1$  and  $s_j \in G_2$  actually creating the flow are identical. Formally, the resultant policy  $P = \{r_1\}$ , such that  $r_1 = G_1 \rightsquigarrow G_2 \Rightarrow s_i \neq s_j$ . A more sophisticate policy could also consider the affiliation of subjects, represented by the corresponding attribute, and forbid data flow between two groups when the subjects are part of the same affiliation:  $G_1 \rightsquigarrow: G_2 \Rightarrow s_1.affiliation \neq s_2.affiliation$ .

**Conflict of interest:** In business, a Chinese wall is designed as an information barrier put within an enterprise to separate and isolate subjects who make investment decisions from persons who

```

1 CONSTRUCT( $T_i^{\mathcal{W}}$ ):
2 MOD := {};
3 for each ( $e$  in  $T_i^{\mathcal{W}}$ ) do
4   Add  $e$  as a vertex to  $PG^{\mathcal{W}}$ ;
5   for each ( $d$  in ( $e.input \cap \text{DOM}(\text{MOD})$ )) do
6     Add edge ( $\text{MOD}(d), e$ ) to  $PG^{\mathcal{W}}$ ;
7   for each ( $d$  in  $e.output$ ) do
8     MOD := MOD  $\cup d \rightarrow e$ ;

```

Figure 4. Pseudo-code for the CONSTRUCT-procedure.

are privy to undisclosed material information which may influence those decisions. This is a way of avoiding conflict of interest problems.

In computer systems, the Chinese Wall policy was identified by Brewer and Nash [11]. Its goal is to enforce a policy where read/write access to files is governed by membership of data in conflict-of-interest classes and datasets. Specifically, the Chinese Wall policy is build upon the classification of conflict classes ( $x$ -attribute) and company set ( $y$ -attribute). The company set is similar to the affiliation-attribute. The conflict classes are dynamically built by examing the previous accesses of an subject. The concrete mapping is context-sensitive and has to be provided separately. The corresponding data flow policy thus captures the simple security property<sup>4</sup> and is formalized, for two subjects  $s_1, s_2 \in S$  as  $s_1 \rightsquigarrow s_2 \Rightarrow s_1.y = s_2.y \vee s_2.y \notin s_1.x$ .

#### IV. ANALYSIS BASED ON GRAPH TRAVERSAL

Analyzing a log file against a data flow policy is a two-step procedure. The first step generates a set of propagation graphs (PG) which denote the data flows that have occurred between subjects during an execution. The second step analyzes PG for data flows according to a given policy. The following describes these steps in detail.

##### A. Construction of Propagation Graphs

Intuitively, a propagation graph (PG) for a trace  $T$  represents the flow of data between the activities (events) that occurred during the execution. Hence, the nodes are activities and the labeled vertices are the data flows.

*Definition 4:* Let  $L_{\mathcal{W}}$  be the log file for a workflow  $\mathcal{W}$ . A *propagation graph* for a trace  $T$  is a tuple  $PG_T = (E, \rightarrow)$ , where  $E$  is the set of log events/activities in  $T$  and  $\rightarrow$  is the asymmetric relation introduced in Def. 1 representing the graph's (directed) edges. An edge  $e \rightarrow e'$  from  $PG_T$  indicates an event data flow from event  $e$  to  $e'$ .  $\dashv$

Fig. 4 shows the pseudo-code for the CONSTRUCT-procedure, which constructs the propagation graph  $PG_n^{\mathcal{W}}$  for a trace  $T_i^{\mathcal{W}}$ . During the construction process, the procedure maintains a mapping MOD that maps a data item to the activity that last modified it. Starting with the initial activity (Line 3), every activity from  $i_n^{\mathcal{W}}$  is added as a vertex to  $PG_n^{\mathcal{W}}$  (Line 4). It is checked

<sup>4</sup>The ss-property states that a subject cannot read *above* his/her security domain. The conflict of interest classes arise from the horizontal view of the lattice, in that information should not flow between classes at the same level.

whether an input datum of the activity that is currently under consideration has been modified by previous activities (Line 5). This is done by comparing the activities' *input*-attributes with the domain  $\text{DOM}(\text{MOD})$  of mapping MOD. If a data item  $d$  is found, an edge from the activity that last modified  $d$  to the current activity is inserted into  $PG^{\mathcal{W}}$  (Line 6). Afterwards, the mapping for each data item that is modified by the current activity is updated or added to MOD (Line 8). The result of the CONSTRUCT procedure is a propagation graph  $PG^{\mathcal{W}}$  which contains every activity in  $T_i^{\mathcal{W}}$  as a vertex. Not every activity in  $PG^{\mathcal{W}}$  can necessarily be reached from the initial activity, if there has not been a corresponding data flow. Therefore,  $PG^{\mathcal{W}}$  might have multiple nodes with an indegree of zero (i.e. there are no inbound data flows).

The construction procedure thus analyzes a log file in a trace-wise manner and generates a PG for each workflow execution. To this end, it chronologically scans the list of log events of an execution and adds each event as a vertex to the PG. During the scan, it maintains a mapping which stores for each data item the event that modified it last; if the currently inspected event modifies a data item, an edge is added to the PG from the event that previously modified the item to the current one. Hence, with  $|T|$  being the number of log events for an execution  $T$  and  $|D_T|$  being the number of data items processed by  $T$ , the worst-case execution time for the construction of a PG is in  $O(|T| \times |D_T|)$ . However, it is lower in the average case, as an activity usually processes only a fraction of the data items.

### B. Data Flow Analysis

Fig. 5 shows the pseudo-code for the analysis procedure of RECIF, which traverses a propagation graph  $PG_i$  in a depth-first manner and checks for violation of rules of a policy  $P$ . Assuming PATH as global variable, the CHECK is the main procedure which initializes the data structures and triggers the traversals. The starting point for the traversals are vertices (i.e. activity executions) which do not have incoming edges (Line 2). During traversal, the analysis routine maintains the list `triggeredRules` of currently *triggered* policy rules. A rule is triggered whenever there was an activity execution performed on behalf of a subject which is the source of a restricted data flow (Line 10). The relation PATH stores, for each rule  $r$ , the sequence of vertices from the vertex that triggered that rule to the one that closed the rule, i.e. the path that violates the restriction.

The VISIT procedure is recursively applied to each vertex of a  $PG_i$ . For each rule that has not yet been triggered (Line 9), it checks whether the originator of the current vertex matches the left side of a rule's restriction (`r.restriction.source`) and updates the list of triggered rules (Line 10). For all triggered rules, it adds the current vertex to the corresponding PATH (Line 12) and reports a violation if the updated PATH is prohibited by the rule (Line 15). This is the case whenever the current vertex is a sink of the rule's restriction (Line 13) and possible exceptions are not applicable. (The evaluation of the exception clause is omitted due to space reasons.) Subsequently, the traversal proceeds by applying VISIT to all direct successors of the current vertex (Line 16,17,18). Upon the return of the recursive calls to VISIT, the current vertex is removed from the PATH of all currently triggered rules (Line 20). All rules with an empty PATH are additionally marked as untriggered (Line 21).

```

1 CHECK( $PG_i, P$ ):
2 for each (Vertex  $v$  in  $PG_i$  with  $INDEGREE(v) = 0$ )
3   triggeredRules = {};
4   for each (Rule  $r \in Policy$ ) do
5     PATH( $r$ ) = [ ];
6     VISIT( $PG_i, v, triggeredRules, policy$ );
7
8 VISIT( $graph, vertex, triggeredRules, policy$ ):
9 for each (Rule  $r \in policy \wedge r \notin triggeredRules$ )
10  if ( $vertex.originator = r.restriction.source$ ) then  $triggeredRules = triggeredRules \cup r$ ;
11  for each (Rule  $r \in triggeredRules$ )
12    PATH( $r$ ) = PATH( $r$ ) +  $vertex$ ;
13    if ( $vertex.originator = r.restriction.sink$ ) then
14      Evaluate exception clause of rule  $r$  and
15      if false, Report violation of  $r$  through flow over PATH( $r$ );
16  Successors = { $v \mid v \in vertices\ of\ graph \wedge (vertex, v) \in edges\ of\ graph$ };
17  for each (Vertex  $v$  in Successors) do
18    VISIT( $graph, v, triggeredRules, policy$ );
19  for each (Rule  $r \in triggeredRules$ ) do
20    PATH( $r$ ) = PATH( $r$ ) -  $vertex$ ;
21  if (PATH( $r$ ) = [ ]) then  $triggeredRules = triggeredRules \setminus r$ ;

```

Figure 5. Pseudo-code for the analysis of a propagation graph  $PG_i$ .

**Complexity issues.** Let  $n$  denote the number of traces in the log file and  $m$  the maximal number of activities in the traces, the complexity of PG generation is linear and amounts to  $O(n \times m)$ . The analysis consists of a depth-first graph traversal. In consequence, searching each PG has the complexity of  $O(|T| + |\rightarrow|)$ . A more detailed analysis is being carried out to investigate the role that branches play in the analysis. Practically, experiments carried out with our prototypical implementation demonstrate the feasibility of RECIF to cope with large log files. Section V includes exemplary runtime figures for the analysis of SoD policies.

## V. EXAMPLE

The following example shows the application of RECIF to analyze a review process for research papers. This is clearly not an example of an industrial workflow, but insofar representative as it employs a process that shares the same characteristics and requirements as large workflow. The review process involves three different types of subjects, namely the review coordinators, the reviewers, and the contributing authors. Fig. 6 outlines the process in BPMN notation, where the activities of coordinators, reviewers and authors are each modeled in different swimlanes. As an annotation of standard BPMN, the need for two reviewers to complete a review is represented by the cardinality of outgoing and ingoing messages.

We simulate the execution of the review process in a workflow execution environment, which

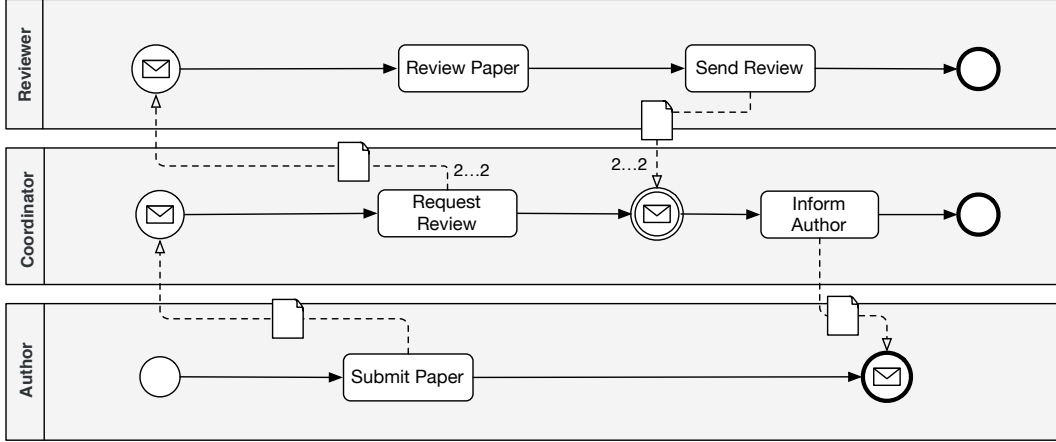


Figure 6. Review process in BPMN notation.

Table I  
AFFILIATION OF SUBJECTS.

Subject	Affiliation
$author_1$	$dept_A$
$author_2$	$dept_B$
$reviewer_1$	$dept_C$
$reviewer_2$	$dept_A$
$reviewer_3$	$dept_D$
$coordinator$	$dept_E$

produces the corresponding log data. The case study comprises six subjects, of which one is the coordinator of the review process ( $coordinator$ ), two are authors ( $author_1, author_2$ ) and three are reviewers ( $reviewer_1, reviewer_2, reviewer_3$ ). We assume that every subject is affiliated to an institution; the mapping is given in Table I.

**Policy.** The policy for the review process consists of one SoD rule  $r$  that demands that a reviewer must not review papers from an author that has the same affiliation as the reviewer. This rule is formalized as a data flow constraint. It prescribes that, for any data flow between an author and a reviewer, the author and the reviewer must have different affiliations. For this, the two groups  $A$  and  $R$ , denoting the set of authors and the set of reviewers, is defined:

$$A = \{author_1, author_2\}; R = \{reviewer_1, reviewer_2, reviewer_3\}$$

The data flow policy follows the general pattern introduced in Section III, being defined as  $r = A \rightsquigarrow R \Rightarrow a.affiliation \neq r.affiliation$ .

**Construction of propagation graphs.** For each instance of the review process, RECIF builds a propagation graph denoting the data flow between subjects that occurred during process execution. Since PG abstract from the actual content of data items, PG that are structurally equivalent to ones that have been generated previously are discarded, thereby reducing the runtime of the analysis procedure.

Table II shows an excerpt of the log file that has been generated during the simulation of the

Table II  
LOG FILE EXCERPT FOR AN EXECUTION OF THE REVIEW PROCESS.

Entry	Instance	Activity	Originator	Input	Output
$e_1$	1	Submit Paper	$author_1$	$\{paper\}$	$\{submission\}$
$e_2$	1	Request Review	$organisator$	$\{submission\}$	$\{issue_1, issue_2\}$
$e_3$	1	Review Paper	$reviewer_1$	$\{issue_1\}$	$\{review_1\}$
$e_4$	1	Send Review	$reviewer_1$	$\{review_1\}$	$\{result_1\}$
$e_5$	1	Review Paper	$reviewer_2$	$\{issue_2\}$	$\{review_2\}$
$e_6$	2	Submit Paper	$author_2$	$\{paper\}$	$\{submission\}$
$e_7$	1	Send Review	$reviewer_2$	$\{review_2\}$	$\{result_2\}$
$e_8$	1	Inform Author	$organisator$	$\{result_1, result_2\}$	$\{\}$
$e_9$	2	Request Review	$organisator$	$\{submission\}$	$\{issue_1, issue_2\}$
$e_{10}$	2	Review Paper	$reviewer_3$	$\{issue_1\}$	$\{review_1\}$

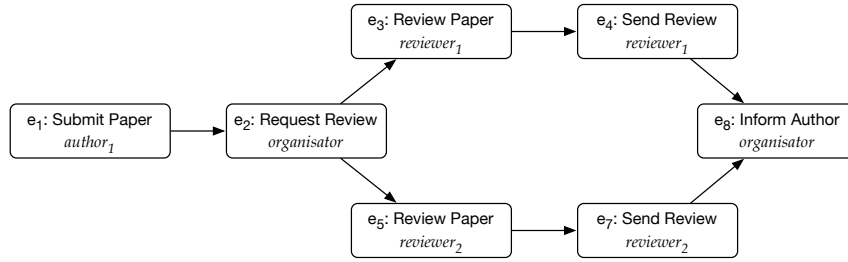


Figure 7. Propagation graph for log data from Table II.

review process. This excerpt accounts for two execution of the process where instance two is only fragmentary represented. For the analysis, we focus on instance one. In the corresponding trace  $T_1$ ,  $author_1$  submits a paper for which the *coordinator* requests a review from  $reviewer_1$  and  $reviewer_2$ . Fig. 7 depicts the corresponding propagation graph.

**Data flow analysis.** The data flow analysis procedure of RECIF traverses the propagation graphs in a depth-first fashion. For the policy used in this case study, noncompliance detection boils down to verifying whether there are paths from authors to reviewers with the same affiliation. The propagation graph in Fig. 7 contains a violation of this policy in path  $\langle e_1, e_2, e_5 \rangle$ , which is detected by RECIF. Here, a paper written by  $author_1$  is reviewed by  $reviewer_2$  who has the same affiliation as  $author_1$ .

Albeit simple, the case study illustrates the kind of analyses performed by RECIF. Currently, the approach is evaluated in further case studies which comprise more complex processes and policies, such as the Chinese Wall policy in a consultancy scenario with multiple processes and several dozens of subjects (consultants and customers).

**Runtime figures.** A tool for the generation of large propagation graphs (PG) was built to empirically assess the runtime of RECIF. There are two main factors influencing the time taken for the PG traversal (analysis): first, the size of the PG (i.e. the number of nodes) and, second, its degree (i.e. the number of incoming and outgoing arcs of nodes). While the size of the PG is passed to the generation tool as a fixed number, the degree is specified as a interval of integers

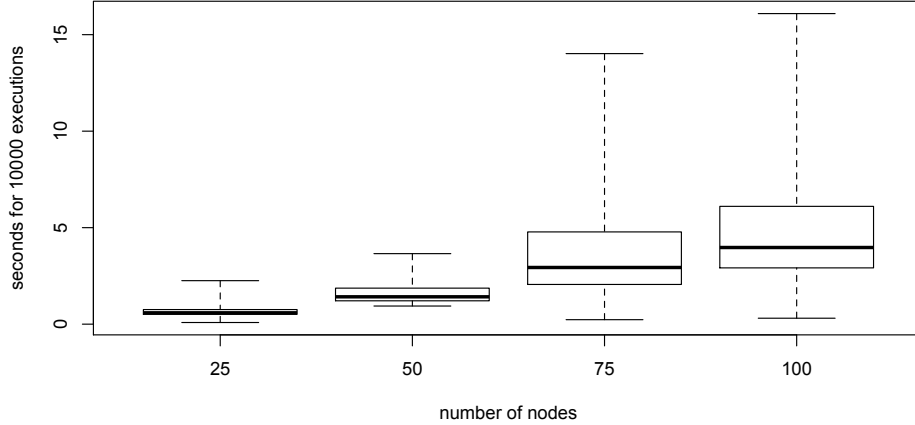


Figure 8. Runtime figure obtained with the prototypical RECIF realization.

[1;3], from which the tool randomly picks a value for each vertex: here, each node may process up to three data elements, which is a reasonable number in industry [35].

Runtime figures are drawn for PG with 25, 50, 75 and 100 nodes. For each size, a series of 100 PG was generated, which differ in the number and distribution of arcs (i.e. data flows). The policy used for the experiments is a simple data flow restriction between arbitrary nodes without an exception (the exact semantics of the policy is irrelevant; in any case, a PG is completely traversed once). Using an implementation of the analysis algorithm in Java, the search was carried out on a computer with a 2GHz clock rate using 4GB of main memory and executed 10K times for every generated graph with the same policy as input. Time was measured for all 10K executions together.

Fig. 8 depicts the runtime figures of this experiment. For every node cardinality the mean execution time (fat line) along with the distribution is presented (thin lines). The maximal elapsed time amounts to 16 seconds for 10K executions on a graph with 100 nodes. Furthermore, the mean runtime clearly increases with the number of nodes. This is mainly due to the distribution of the number of incoming nodes. The analysis procedure analyzes all paths of a given graph for possible violations and is, therefore, directly affected by the length of the path. To give meaningful estimations of the runtime, further classifications of typical PG will be analyzed with regard to the resulting path length.

Overall, given the prototypical character of the realization, these results underscore the feasibility of RECIF in industrial settings. We note though that the analysis happens “after the fact”. Hence, the actual runtime necessary to analyze a log file is not quite an issue, but rather the precision of the findings. While the policy language presented in Section III is expressive enough to capture the desired properties, ongoing work aims at demonstrating the correctness of the analysis algorithm, in that it detects all the violations expressed by the policies.

## VI. DISCUSSION

In the following we discuss pertinent issues regarding RECIF. Specifically, we address the interplay of RECIF with other forensic approaches and investigation steps; the dependency on the MXML format; and the need for the proposed policy language.

*RECIF and forensic investigation:* RECIF relies on authentic system records (logs) to generate correct evidence. To this end, tools are needed to attest the authenticity of log files in case secure logging mechanisms [1] are not in place. Furthermore, the analysis level at which RECIF operates is high: the execution traces generated by the workflow management system represent only a part of the system execution and in order to obtain further evidence as to whether an attack took place, other layers of abstraction, including databases and operating systems are necessary. This is the realm of traditional forensic techniques and, complementing them, RECIF is an approach to analyze workflow executions.

*MXML, the log format:* While the implementation and formal model presented in this paper explicitly build upon the structure of MXML log files, RECIF does not depend upon MXML. In fact, most of the formats used in practice include the same fields, so that we could as well use other log formats and reduce them to the MXML format. (Tool support for this steps is readily available.) Indeed, flexibility with regard to the source log format is essential. MXML provides thus a uniform basis for their analysis.

*Policy language:* The language presented in Section III is cannot be really considered a new policy language, but a language to express audit constraints. The idea is to provide a means to capture domain independent policy templates, such as for SoD constraints, and make these patterns available for auditors. Clearly, in order to apply RECIF, forensic investigators have to instantiate the policies accordingly, capturing the semantics of the claim they want to refute or corroborate. While this might involve training, we believe that the investigators would not face problems in instantiating policies. In particular, being aware of the set of vulnerabilities that can occur at process level [24] can help investigators to come up with relevant policies.

## VII. SUMMARY AND ONGOING WORK

This paper presents RECIF, an approach for the forensic analysis business processes logs against data flow policies expressed as data flow constraints. The formalization and checking of compliance requirements using RECIF is currently being evaluated with a proof-of-concept implementation. For this purpose, a simulation environment was built to execute workflows and generate the corresponding log files. Workflows are taken from a collection of industrial business processes and their executions are parametrized to exhibit illicit data flows with respect to isolation policies that were derived from common service-level agreements.

RECIF is part of a project on the design of well-founded mechanisms for workflow analysis. Central to the project is the development of a formal meta-model for workflows which is based on Petri nets and subsumes and supersedes the propagation graphs used in this paper. Corresponding workflow models are generated either from static business process descriptions (for instance in BPEL or BPMN) which are complemented with runtime information from log traces, or solely from log data, using reconstruction algorithms. As future work, we intend to apply information flow analysis techniques based on Petri nets to detect policy violations.

## REFERENCES

- [1] R. Accorsi, “Safe-keeping digital evidence with secure logging protocols: State of the art and challenges,” in *Proceedings the IEEE Conference on Incident Management and Forensics*, O. Goebel, R. Ehlert, S. Frings, D. Günther, H. Morgenstern, and D. Schadt, Eds. IEEE, 2009, pp. 94–110.
- [2] —, “BBox: A distributed secure log architecture.” in *European Workshop on Public Key Services, Applications and Infrastructures*, to appear ser. Lecture Notes in Computer Science. Springer, 2011.
- [3] R. Accorsi and C. Wonnemann, “Strong non-leak guarantees for workflow models,” to appear *ACM Symposium on Applied Computing*, 2011.
- [4] R. Agrawal, D. Gunopulos, and F. Leymann, “Mining process models from workflow logs,” in *Proceedings of the Conference on Advances in Database Technology*, ser. Lecture Notes in Computer Science, H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, Eds., vol. 1377. Springer, 1998, pp. 469–483.
- [5] A. Armando and S. E. Ponta, “Model checking of security-sensitive business processes,” in *Proceedings of the Workshop on Formal Aspects in Security and Trust*, ser. Lecture Notes in Computer Science, P. Degano and J. Guttman, Eds., vol. 5983. Springer, 2009, pp. 66–80.
- [6] M. Barletta, S. Ranise, and L. Viganò, “Verifying the interplay of authorization policies and workflow in service-oriented architectures,” in *Proceedings of the Conference on Computational Science (3)*, 2009, pp. 289–296.
- [7] D. Bell and L. LaPadula, *Secure Computer Systems: Mathematical Foundations*. MITRE Corporation, 1973.
- [8] D. Bem, “Virtual machine for computer forensics – the open source perspective,” in *Open Source Software for Digital Forensics*, E. Huebner and S. Zanero, Eds. Springer, 2010, pp. 25–42.
- [9] R. Botha and J. Eloff, “Separation of duties for access control enforcement in workflow environments,” *IBM Systems Journal*, vol. 40, no. 3, pp. 666–682, 2001.
- [10] S. Böttcher and R. Steinmetz, “Finding the leak: A privacy audit system for sensitive xml databases,” in *International Workshop on Privacy Data Management*. ACM, 2006.
- [11] D. Brewer and M. Nash, “The Chinese-wall security policy,” in *IEEE Symposium on Security and Privacy*, 1989, pp. 206–214.
- [12] K.-D. Bussmann, O. Krieg, C. Nestler, S. Salvenmoser, A. Schroth, A. Theile, and D. Trunk, “Wirtschaftskriminalitt 2009 – Sicherheitslage in deutschen Grounternehmen,” Report by Martin-Luther-Universität Halle-Wittenberg and PricewaterhouseCoopers AG, 2009, (in German).
- [13] M. Chandrasekaran, V. Sankaranarayanan, and S. J. Upadhyaya, “Inferring sources of leaks in document management systems,” in *IFIP Conference on Digital Forensics*, ser. IFIP, I. Ray and S. Sheno, Eds., vol. 285. Springer, 2008, pp. 291–306.

- [14] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, “Controlling data in the cloud: Outsourcing computation without outsourcing control,” in *Proceedings of the ACM Workshop on Cloud Computing Security*. ACM, 2009, pp. 85–90.
- [15] J. Crampton and H. Khambhammettu, “On delegation and workflow execution models,” in *ACM Symposium on Applied Computing*, R. L. Wainwright and H. Haddad, Eds. ACM, 2008, pp. 2137–2144.
- [16] D. Denning, “A lattice model of secure information flow,” *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.
- [17] A. Ghose and G. Koliadis, “Auditing business process compliance,” in *Conference on Service-Oriented Computing*, ser. Lecture Notes in Computer Science, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds., vol. 4749. Springer, 2007, pp. 169–180.
- [18] M. Gunestas, D. Wijesekera, and A. Singhal, “Forensic web services,” in *IFIP Conference on Digital Forensics*, ser. IFIP, I. Ray and S. Sheno, Eds., vol. 285. Springer, 2008, pp. 163–176.
- [19] C. Hammer, J. Krinke, and G. Snelting, “Information flow control for java based on path conditions in dependence graphs,” in *IEEE International Symposium on Secure Software Engineering*. IEEE Computer Press, 2006, pp. 87–96.
- [20] B. Hay, M. Bishop, and K. Nance, “Live analysis: Progress and challenges,” *IEEE Security & Privacy*, vol. 7, no. 2, pp. 30–37, 2009.
- [21] E. Kenneally, “Digital logs – Proof matters,” *Digital Investigation*, vol. 1, no. 2, pp. 94–101, 2004.
- [22] B. Lampson, “A note on the confinement problem,” *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, 1973.
- [23] B. Livshits, A. Nori, S. Rajamani, and A. Banerjee, “Merlin: Specification inference for explicit information flow problems,” in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, M. Hind and A. Diwan, Eds. ACM, 2009, pp. 75–86.
- [24] L. Lewis and R. Accorsi, “On a classification approach for soa vulnerabilities,” in *IEEE International Computer Software and Applications Conference*. IEEE Computer Society, 2009, pp. 439–444.
- [25] —, “Finding vulnerabilities in SOA-based business processes,” to appear *IEEE Transactions on Service Computing*, 2011.
- [26] K. E. Pavlou and R. T. Snodgrass, “Forensic analysis of database tampering,” *ACM Transactions on Database Systems*, vol. 33, no. 4, 2008.
- [27] M. Ponc, P. Giura, H. Brönnimann, and J. Wein, “Highly efficient techniques for network forensics,” in *ACM Conference on Computer and Communications Security*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007, pp. 150–160.

- [28] B. Roscoe, “Intensional specifications of security protocols,” in *IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996, pp. 28–38.
- [29] P. Stahlberg, G. Miklau, and B. Levine, “Threats to privacy in the forensic analysis of database systems,” in *Conference on Management of Data*, C. Y. Chan, B. C. Ooi, and A. Zhou, Eds. ACM Press, 2007, pp. 91–102.
- [30] S. Sun, L. Zhao, J. Nunamaker, and O. L. Sheng, “Formulating the data-flow perspective for business process management,” *Information Systems Research*, vol. 17, no. 4, pp. 374–391, 2006.
- [31] N. Trčka, W. van der Aalst, and N. Sidorova, “Data-flow anti-patterns: Discovering data-flow errors in workflows,” in *Conference on Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, P. van Eck, J. Gordijn, and R. Wieringa, Eds., vol. 5565. Springer, 2009, pp. 425–439.
- [32] W. van der Aalst, K. van Hee, J. M. van der Werf, and M. Verdonk, “Auditing 2.0: Using process mining to support tomorrow’s auditor,” *IEEE Computer*, vol. 43, no. 3, pp. 90–93, 2010.
- [33] B. van Dongen and W. van der Aalst, “A Meta Model for Process Mining Data,” in *CAiSE’05 Workshops*, J. Casto and E. Teniente, Eds., vol. 2, 2005, pp. 309–320.
- [34] H. M. Verbeek, C. A. M. Joos, and B. D. W. van der Aalst, “XES, XESame, and ProM 6,” in *Information Systems Evolution*, ser. Lecture Notes in Business Information Processing, W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, P. Soffer, and E. Proper, Eds. Springer, 2011, vol. 72, pp. 60–75.
- [35] C. Wolf and P. Harmon, “The state of business process management,” BPTrends Report, 2010, available at <http://www.bptrends.com/>.