Purpose Control: did you process the data for the intended purpose?*

Milan Petković[1,3], Davide Prandi[2], and Nicola Zannone[3]

[1] Philips Research Eindhoven
milan.petkovic@philips.com
[2] Centre for Integrative Biology, University of Trento
prandi@science.unitn.it
[3] Eindhoven University of Technology
n.zannone@tue.nl

Abstract. Data protection legislation requires personal data to be collected and processed only for lawful and legitimate purposes. Unfortunately, existing protection mechanisms are not appropriate for purpose control: they only prevent unauthorized actions from occurring and do not guarantee that the data are actually used for the intended purpose. In this paper, we present a flexible framework for purpose control, which connects the intended purpose of data to the business model of an organization and detects privacy infringements by determining whether the data have been processed only for the intended purpose.

1 Introduction

In recent decades, many countries have enacted privacy laws and regulations that impose very stringent requirements on the collection and processing of personal data (e.g., EU Directive 95/46/EC, HIPAA). Purpose control plays a central role in such legislation [1]: personal data shall be collected for specified, lawful and legitimate purposes and not processed in ways that are incompatible with the purposes for which data have been collected. Purpose control requires the deployment of mechanisms that hold users accountable for their actions by verifying how data have actually been processed.

In contrast, current security and data protection mechanisms do not provide appropriate support for purpose control. They are preventive and, more importantly, they do not check for which purpose data are processed after access to data has been granted. Protection of personal information is often implemented by augmenting access control systems with the concept of purpose [2–5] (hereafter, we call access control policies augmented with purpose *data protection policies*). Here, protecting data implies guaranteeing that data are disclosed solely to authorized users with the additional condition that data are requested for the intended purpose. The access purpose is usually specified by the requester [4], implying complete trust on users. This poses risks of re-purposing the data [6, 7] as users might process the data for purposes other than those for which the data were originally obtained. Therefore, to ensure compliance to data protection

* This work has been partially funded by the EU-IST-IP-216287 TAS[3] project.

and purpose control, it is necessary to extend the current preventive approach by implementing mechanisms for verifying the actual use of data.

Some privacy enhancing technologies (e.g., [2, 8]) partially address this issue. They collect and maintain *audit trails*, which record the actual user behavior, for external privacy audits. These auditing activities, however, are usually manual; the auditors sample and inspect the audit trails recorded by the system. The lack of systematic methods for determining how data are used makes auditing activities time-consuming and costly. For instance, at the Geneva University Hospitals, more than 20,000 records are opened every day [9]. In this setting, it would be infeasible to verify every data usage manually, leading to situations in which privacy breaches remain undetected.

In this paper, we present a framework for purpose control which detects privacy infringements by determining whether data are processed in ways that are incompatible with the intended purpose of data. To this end, we need a purpose representation model that connects the intended purpose of data to the business activities performed by an organization and methods able to determine whether the data are actually processed in accordance with purpose specifications.

Organizations often make use of business processes to define how organizational goals should be fulfilled. These *organizational processes* define the expected user behavior for achieving a certain organizational goal. The idea underlying our approach is to link the purposes specified in data protection policies to organizational goals and, therefore, to the business processes used to achieve such goals. Thus, the problem of verifying the compliance of data usage with the intended purpose consists in determining whether the audit trail is a valid execution of the organizational processes representing the purposes for which data are meant to be used. Intuitively, if the audit trail does not correspond to a valid execution of those processes, the actual data usage is not compliant with the purpose specification.

To enable automated analysis, we use the Calculus of Orchestration of Web Services (COWS) [10] for the representation of organizational processes. COWS is a foundational calculus strongly inspired by WS-BPEL [11]. It is based on a very small set of primitives associated with a formal operational semantics that can be exploited for the automated derivation of the dynamic evolution of the process. The COWS semantics makes it possible to construct a labeled transition system that generates the set of traces equivalent to the set produced by all possible executions of the process.

A naïve approach for purpose control would be to generate the transition system of the COWS process model and then verify if the audit trail corresponds to a valid trace of the transition system. Unfortunately, the number of possible traces can be infinite, for instance when the process has a loop, making this approach not feasible. Therefore, in this paper we propose an algorithm that replays the audit trail in the process model to detect deviations from the expected behavior. We demonstrate that the algorithm terminates and is sound and complete.

The structure of the paper is as follows. Next section presents our running example. (§2). Then, we introduce the building blocks of our framework and analyze their alignment (§3). We present an algorithm for purpose control (§4) and demonstrate the termination, soundness and completeness of the algorithm (§5). Finally, we discuss related work (§6) and conclude the paper, providing directions for future work (§7).

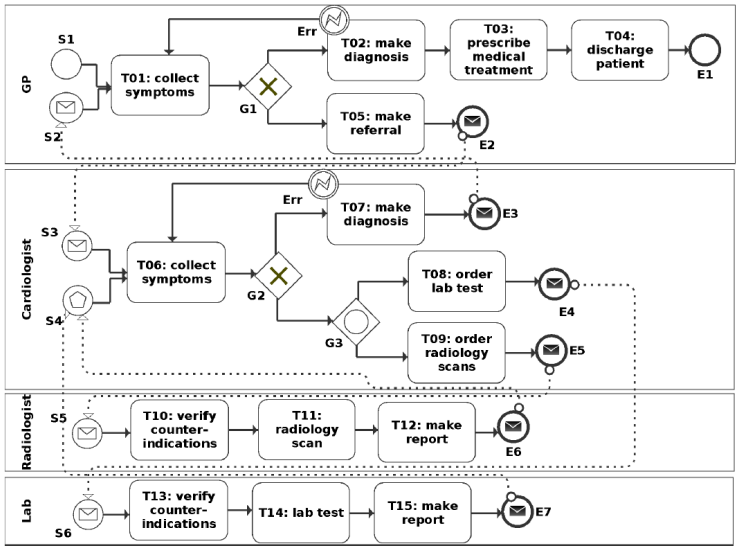


Fig. 1. Healthcare Treatment Process

2 Running example

This section presents a simple scenario in the healthcare domain to illustrate our approach. Consider a patient who goes to a hospital to see a doctor. The hospital is equipped with its own Hospital Information System (HIS) to manage the administrative, financial, and clinical aspects of patient information. In particular, patient information are stored in an electronic patient record (EPR); here, we assume that EPRs are organized in sections; each of them contains a certain type of information. Members of the hospital staff can access specific sections of the EPR (or parts of them) depending on their position within the hospital and for well defined purposes. Suppose that a patient, Jane, did not give the hospital consent to process her information for research purposes. Then, the hospital staff cannot access Jane's information for clinical trials.

The provision of healthcare treatments can be seen as a process that involves several parties. Fig. 1 describes an example of a healthcare treatment process specified in the Business Process Modeling Notation (BPMN) [12]. Here, every BPMN pool[4] represents the visit of the patient to a member of the clinical staff at the local hospital. The process starts when a patient visits the general practitioner (GP) at the hospital (S1). The GP accesses the HIS to retrieve the patient's EPR and makes a physical examination to collect the symptoms (T01). Based on the gathered information, the GP may suspect that the patient is affected by a certain disease. He can either make a diagnosis (T02) or refer to a specialist if the case is more complex (T05). For instance, in case the GP suspects a cardio-vascular disease, he can refer the patient to a cardiologist.

[4] In BPMN a pool is used to specify the boundaries of the activities to be performed by a participant in the process and is graphically represented as a container enclosing those activities.

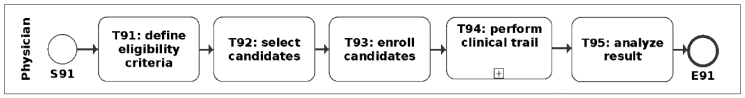


Fig. 2. Clinical Trial Process

If the patient is referred to a cardiologist (S3), the cardiologist accesses patient medical history in the HIS and makes a medical examination to collect the symptoms (T06). Based on this information, the cardiologist can either make a diagnosis directly (T07), or request lab tests or radiology scans (T08 and T09, respectively). If the resulting tests and scans are not good or a diagnosis cannot be made based on them, further tests can be required. When the lab or the radiology department receive the request for tests from the cardiologist (S5 and S6, respectively), they check the EPR for allergies or other counter-indications (T10 and T03, respectively). Then, they do the lab exam (T11 and T14) and export the results to the HIS (T12 and T15). A notification is sent to the cardiologist when the tests or the scans have been completed (E6 and E7).

When the cardiologist receives a notification for all the ordered tests and scans (S4), he retrieves the test results or the scans from the HIS (T06) and, based on them, makes a diagnosis (T07). After the cardiologist enters the diagnosis into the HIS, a notification is sent to the GP (E4). Upon the notification (S2), the GP extracts the diagnosis made by the cardiologist, scans and tests from the patient's EPR (T01). Based on them, the GP prescribes medical treatments to the patient (T03) and discharges her (T04).

Suppose now that the same cardiologist mentioned above is involved in a clinical trial. Fig. 2 shows the part of the clinical trial process in which a physician is involved. The physician has to define eligibility criteria for the trial (T91). Then, he accesses EPRs to find patients with a specific condition that meet the eligibility criteria (T92). In the next step, the physician asks the selected candidates whether they want to participate in the trial (T93). The trial is then performed (T94) and the results are analyzed (T95). Note that the cardiologist can get access to patient information for the legitimate purpose (i.e., claiming that it is for healthcare treatment) and then use the data for research purposes (i.e., clinical trial). Preventive mechanisms are not able to cope with these situations. In particular, they cannot prevent a user to process data for other purposes after the same user has legitimately got access to them.

To address this issue, we need mechanisms that make users accountable for their actions by determining how data were actually used. In this paper, we propose an approach that enables purpose control by determining if an audit trail is a valid execution of the process used by the organization to implement the intended purposes. In our scenario, this implies verifying that every usage of patient information is part of the sequence of tasks that the cardiologist and the other parties involved in the provision of healthcare treatments have to perform in order to accomplish the goal.

3 A framework for purpose control

In this section, we propose a formal framework for purpose control. The main goal of the framework is to verify if a user processed the data only for the intended purpose. The proposed approach uses and interlinks the following three components:

- *Data protection policies* that define who can access the data and for which purpose.
- *Organizational processes* that describe the business processes and procedures of an organization.
- *Audit trails* that record the sequence of actions performed by users.

In the remainder of this section, we present the formal model components of the framework and discuss their alignment.

3.1 Basic concepts and notation

Our framework includes sets of five basic elements: users ($\mathcal{U}$), roles ($\mathcal{R}$), objects ($\mathcal{O}$), actions ($\mathcal{A}$), and organizational processes ($\mathcal{P}$). A *user* denotes any entity capable to perform actions on an object, and a *role* denotes a job function describing the authority and responsibility conferred on a user assigned to that role. We assume that roles are organized into a hierarchical structure under partial ordering $\geq_R$. Such an order reflects the principles of generalization and specialization. Let $r_1, r_2 \in \mathcal{R}$ be roles. We say that r_1 is a specialization of r_2 (or r_2 is a generalization of r_1) if $r_1 \geq_R r_2$.

Objects denote resources (i.e., data, services, or system components) under protection. We use a directory-like notation to represent hierarchical resources like file systems and EPRs. This implies a partial ordering $\geq_O$ on resources, which reflects the data structure. In addition, we make explicit the name of the data subject when appropriate. For instance, Jane's EPR is denoted as $[Jane]EPR$, whereas $[Jane]EPR/Clinical$ denotes the section containing medical information in her EPR, with $[Jane]EPR \geq_O [Jane]EPR/Clinical$. We use $[\cdot]EPR$ to denote EPRs, regardless a specific patient.

An *action* is an operation that can be performed on a resource. The set of actions $\mathcal{A}$ includes "read", "write", and "execute".

Organizational processes specifies the sequences of *tasks* that have to be performed to achieve organizational goals. We use $\mathcal{Q}$ to denote the set of tasks belonging to any process in $\mathcal{P}$. Process models can specify the entity that is expected to perform a certain task (e.g., using pools in BPMN). However, processes usually are not specified in terms of identifiable users. For example, a hospital would not define a specific clinical trial process for each physician. Business processes are intended to be general in order to cover a large number of scenarios. Therefore, we assume that every BPMN pool corresponds to a role in $\mathcal{R}$. Finally, a given task can be part of more than one process, and several instances of the same process can be executed concurrently. To apply conformance checking techniques, it is necessary to distinguish the process instance (the so called *case*) in which tasks are performed [13]. Hereafter, $\mathcal{C}$ denotes the set of cases.

In data protection, the concept of *purpose* plays a central role [1, 2, 4]. The purpose denotes the reason for which data can be collected and processed. In particular, we refer to purposes that regulate data access as *intended purposes*, and to purposes

$(Physician, read, [\cdot]EPR/Clinical, treatment)$ $(Physician, write, [\cdot]EPR/Clinical, treatment)$ $(Physician, read, [\cdot]EPR/Demographics, treatment)$
$(MedicalTech, read, [\cdot]EPR/Clinical, treatment)$ $(MedicalTech, read, [\cdot]EPR/Demographics, treatment)$ $(MedicalLabTech, write, [\cdot]EPR/Clinical/Tests, treatment)$
$(Physician, read, [X]EPR, clinicaltrial)$

Fig. 3. Sample Data Protection Policy

for which data access is requested as *access purposed*. Purposes are related to the business activities of an organization and can be identified with organizational goals. For instance, [14] defines a list of purposes for healthcare, which includes healthcare treatment, payment, research and marketing. Thus, in this paper, we represent purposes by the organizational process implemented by an organization to achieve the corresponding organizational goal.

3.2 Data protection policies

The aim of data protection policies is to protect an individual's right to privacy by keeping personal data secure and by regulating its processing. Several languages for the specification of data protection policies have been proposed in literature [3, 4]. The objective of this paper is not to propose yet another policy specification language. Thus, here we present a simple language that suffices for the purpose of this paper.

A data protection policy specifies the access rights: who is authorized to access the system, what actions he is allowed to perform on a resource, and for which purpose.

Definition 1. *A* data protection statement *is a tuple* (s, a, o, p) *where* $s \in \mathcal{U} \cup \mathcal{R}$*,* $a \in \mathcal{A}$*,* $o \in \mathcal{O}$*, and* $p \in \mathcal{P}$*. A* data protection policy Pol *is a set of data protection statements.*

Fig. 3 presents the data protection policy governing our scenario. The first block of Fig. 3 states that physicians can read and write patient medical information in EPRs for treatment. Moreover, physicians can also read patient demographics for treatment. Note that roles GP, radiologist, and cardiologist are specializations of role physician. The second block of statements targets medical technicians. They can read patient medical information for treatment. Moreover, medical lab technicians (which is a specialization of medical technicians) have write permission on the EPR section concerning test results. The last block of Fig. 3 represents the hospital policy that allows physicians to access the EPR of those patients (X) who give consent to use their data for clinical trial.

Users can request access to system's resources by means of access request.

Definition 2. *An* access request *is a tuple* (u, a, o, q, c) *where* $u \in \mathcal{U}$*,* $a \in \mathcal{A}$*,* $o \in \mathcal{O}$*,* $q \in \mathcal{Q}$*, and* $c \in \mathcal{C}$*.*

An access request specifies the user who makes the request, the object to be accessed, and the action to be performed on the object along with information about the purpose for which the request is made. In particular, the access purpose is represented by the task for which the access to the object is requested and by the process instance.

When a user requests permission to execute an action on an object for a certain purpose, the access request is evaluated against the data protection policy. Access is

granted if there exists a data protection statement that matches the access request directly or through a hierarchy.

Definition 3. *Let Pol be a data protection policy and (u, a, o, q, c) an access request. We say that the* access request is authorized *if there exists a statement $(s, a', o', p) \in Pol$ such that (i) $s = u$, or $s = r_1$, u has role r_2 active,[5] and $r_2 \geq_R r_1$; (ii) $a = a'$; (iii) $o' \geq_O o$; (iv) c is an instance of p, and q is a task in p.*

3.3 Organizational processes

Organizational processes specify the activities that users are expected to take in order to accomplish a certain goal. Organizational processes are often modeled in BPMN [12], the de-facto standard modeling notation in industry. Although BPMN provides a standard visualization mechanism, it is informal and therefore not suitable for formal verification. We rely on COWS [10], a foundational language for service-oriented computing that combines constructs from process calculi with constructs inspired by WS-BPEL [11], for the formalization of BPMN processes. Here, we present a minimal version of COWS that suffices for representing organizational processes.

COWS basic entities are *services*, i.e., structured activities built by combining basic activities. COWS relies on three countable and pairwise disjoint sets: *names*, *variables*, and *killer labels*. Basic activities take place at *endpoints*, identified by both a *partner* and an *operation* name. The grammar of COWS is defined as follows:

$$s ::= p \cdot o!\,\langle w \rangle \mid [\,d\,]s \mid g \mid s \mid s \mid \{\!|s|\!\} \mid \mathbf{kill}(k) \mid * \; s$$
$$g ::= \mathbf{0} \mid p \cdot o?\,\langle w \rangle .\, s \mid g + g$$

Intuitively, the basic activities a service can perform are: the empty activity $\mathbf{0}$; $p \cdot o!\,\langle w \rangle$, an *invoke* (sending) activity over endpoint $p \cdot o$ with parameter w; $p \cdot o?\,\langle w \rangle$, a *request* (receiving) activity over endpoint $p \cdot o$ with parameter w; $\mathbf{kill}(k)$, a *block* activity that prevents services within the scope of a killer label k to proceed. The scope for names, variables, and killer labels is denoted by $[\,d\,]s$. The construct $\{\!|s|\!\}$, when not covered by an action, saves a service s from a killer signal sent out by a $\mathbf{kill}(\_)$.

The temporal execution order of the basic activities is described by a restricted set of operators: $p \cdot o?\,\langle w \rangle .\, s$ executes request $p \cdot o?\,\langle w \rangle$ and *then* service s; services s_1 and s_2 running in *parallel* are represented as $s_1 | s_2$; a *choice* between two request activities g_1 and g_2 is written as $g_1 + g_2$. Finally, recursion is modeled with the replication operator $*$: the service $* \, s$ behaves as $* \, s \mid s$, namely $* \, s$ spawns as many copies of s as needed.

COWS is equipped with a structural operational semantics [15], i.e., a set of syntax-driven axioms and rules which describes the dynamics of the system at hand. Specifically, rules allow the definition of a *Labeled Transition System* (LTS) $(s_0, S, L, \rightarrow)$, where S is a set of COWS services or *states*, $s_0 \in S$ is the *initial state*, L is a set of *labels*, and $\rightarrow \subseteq S \times L \times S$ is a *labeled transition relation* among COWS services, such that $(s, l, s') \in \rightarrow$ iff COWS semantics allows one to infer the labeled transition. We use $s \xrightarrow{l} s'$ as a shortcut for $(s, l, s') \in \rightarrow$.

[5] We assume users have to authenticate within the system before performing any action. During the authentication process, the role membership of users is determined by the system.

Labels in L are generated by the following grammar:

$$l ::= (p \cdot o) \triangleleft w \mid (p \cdot o) \triangleright w \mid p \cdot o\,(v) \mid \dagger k \mid \dagger$$

Invoke label $(p \cdot o) \triangleleft w$ and request label $(p \cdot o) \triangleright w$ are for invoke and request activities, respectively. Label $p \cdot o\,(v)$ represents a communication between an invoke label $(p \cdot o) \triangleleft v$ and a request label $(p \cdot o) \triangleright w$. If the communication is indeed a synchronization, the label is simplified as $p \cdot o$. Finally, labels $\dagger k$ and $\dagger$ manage, respectively, ongoing and already executed killing activities. We refer to [10] for a detailed description of the COWS structural operational semantics.

The encoding of BPMN processes into COWS specifications founds on the idea of representing every BPMN element as a distinct COWS service. For the lack of space, here we only present the intuition of the encoding; examples of the encoding are given in Appendix A. In [16], we have defined elementary and parametric COWS services for a core set of BPMN elements. For example, a start event (e.g., $S1$ in Fig. 1) is modeled in COWS as $x \cdot y!\,\langle\rangle$ where $x \cdot y$ is the endpoint triggering the next BPMN element in the flow (x is the pool that contains the element and y is the name of the element); a task (e.g., $T01$ in Fig. 1) is modeled as $x \cdot y?\,\langle\rangle.\,Act$, where $x \cdot y$ is the endpoint to trigger the execution of the task, and Act is the activity performed by the task (Act eventually specifies the next BPMN element). Parameters are instantiated to connect the BPMN elements forming the BPMN process. The COWS service implementing $S1$ and $T01$, denoted by $[[S1]]$ and $[[T01]]$ respectively, are $[[S1]] = GP \cdot T01!\,\langle\rangle$ and $[[T01]] = GP \cdot T01?\,\langle\rangle.\,[[Act]]$, where $[[Act]]$ is the COWS service implementing activity Act and GP stands for general practitioner. The overall organizational process results from the parallel composition of the elementary services implementing the single BPMN elements. The process composed by services $[[S1]]$ and $[[T01]]$ is $[[S1]] \mid [[T01]]$.

The sequence flow, which describes the execution order of process activities by tracking the path(s) of a token through the process, is rendered as a sequence of communications between two services. For instance, the sequence flow between event $S1$ and task $T01$ is defined by the labeled transition

$$[[S1]] \mid [[T01]] \xrightarrow{GP \cdot T01} \mathbf{0} \mid [[Act]]$$

Intuitively, the label $GP \cdot T01$ allows one to "observe" on the COWS transition system that the task received the token. The same idea applies to message flow as well as to other BPMN elements like event handlers and gateways. However, in case of event handlers and gateways, some "internal computation" is required to determine the next BPMN element to be triggered. For instance, exclusive gateway $G1$ in Fig. 1 is connected to $T02$ and $T05$; from $G1$, the token can flow either to $T02$ or $T05$, but not to both. The act of deciding which task should be triggered does not represent a flow of the token. In this case we use the private name sys as the partner in the label. Similarly, label $sys \cdot Err$ is used to represent error signals. In general, it is not known in advance how many times a service is invoked during the execution of a process. An example of this is given by cycles (e.g., $T01$, $G1$ and $T02$ in Fig. 1). To this end, we prefix COWS services with the replication operator $*$. This operator makes multiple copies of the COWS service; each copy corresponds to an invocation of the service.

3.4 Audit trails

Auditing involves observing the actions performed by users to ensure that policies and procedures are working as intended or to identify violations that might have occurred. Audit trails are used to capture the history of system activities by representing events referring to the actions performed by users. Every event is recorded in a log entry.

Definition 4. *A* log entry *is a tuple* (u, r, a, o, q, c, t, s) *where* $u \in \mathcal{U}$*,* $r \in \mathcal{R}$*,* $a \in \mathcal{A}$*,* $o \in \mathcal{O}$*,* $q \in \mathcal{Q}$*,* $c \in \mathcal{C}$*,* t *ties an event to a specific time, and* s *is the task status indicator.*

The field *user* represents the user who performed the *action* on the *object*. *Role* represents the role held by the user at the *time* the action was performed. The *task status indicator* specifies whether the task succeeded or not (i.e., $s \in \{success, failure\}$). We assume that the failure of a task makes the task completed; therefore, no actions within the task are possible after the task has failed. In addition, the process can proceed only if there is in place a mechanism to handle the failure. Log entries also contain information about the purpose for which the action was performed. In particular, the purpose is described by the *task* in which the action was performed and the *case* that identifies the process instance in which the action took place.

An audit trail consists of the chronological sequence of events that happen within the system.

Definition 5. *A* audit trail *is an ordered sequence of log entries where given two entries* $e_i = (u_i, r_i, a_i, o_i, q_i, c_i, t_i, s_i)$ *and* $e_j = (u_j, r_j, a_j, o_j, q_j, c_j, t_j, s_j)$ *we say that* e_i *is before* e_j *(denoted by* $e_i < e_j$*) if* $t_i < t_j$*.*

Recent data protection regulations in the US (see [17]) impose healthcare providers to record all actions related to health information. Accordingly, we assume that audit trails record every action performed by users, and these logs are collected from all applications in a single database with the structure given in Def. 4. In addition, audit trails need to be protected from breaches of their integrity. A discussion on secure logging is orthogonal to the scope of this paper. Here, we just mention that there exist well-established techniques [18, 19], which guarantee the integrity of logs.

Fig. 4 presents a possible audit trail for the scenario of Section 2. The audit trail describes the accesses to Jane's EPR made by the GP (John), the cardiologist (Bob), and the radiologist (Charlie) in the process of providing her medical treatments (we assume that Bob did not order lab tests). It also shows that a number of instances of the process can be executed concurrently. Time is in the form year-month-day-hour-minute. Tasks are denoted by a code as defined in Fig. 1. Case HT-1 represents the instance of the process being executed. In particular, HT stands for the healthcare treatment process and the number indicates the instance of the process.

The last part of Fig. 4 presents the log entries generated during the execution of the clinical trial (CT) process. Here, Bob specified healthcare treatment as the purpose in order to retrieve a larger number of EPRs.[6] Note that preventive mechanisms cannot detect the infringement. Only the patients selected for the trial might suspect a privacy breach. Certainly, the infringement remains covered for those patients who were not selected for the trial and did not allow the usage of their data for research purposes.

[6] Note that, if the physician specifies clinical trial as purpose, the HIS would only return the EPRs of those patients who gave their consent to use their information for research purposes.

user	role	action	object	task	case	time	status
John	GP	read	[Jane]EPR/Clinical	T01	HT-1	201003121210	success
John	GP	write	[Jane]EPR/Clinical	T02	HT-1	201003121212	success
John	GP	cancel	N/A	T02	HT-1	201003121216	failure
John	GP	read	[Jane]EPR/Clinical	T01	HT-1	201003121218	success
John	GP	write	[Jane]EPR/Clinical	T05	HT-1	201003121220	success
John	GP	read	[David]EPR/Demographics	T01	HT-2	201003121230	success
			...				
Bob	Cardiologist	read	[Jane]EPR/Clinical	T06	HT-1	201003141010	success
Bob	Cardiologist	write	[Jane]EPR/Clinical	T09	HT-1	201003141025	success
Charlie	Radiologist	read	[Jane]EPR/Clinical	T10	HT-1	201003201640	success
Charlie	Radiologist	execute	ScanSoftware	T11	HT-1	201003201645	success
Charlie	Radiologist	write	[Jane]EPR/Clinical/Scan	T12	HT-1	201003201730	success
Bob	Cardiologist	read	[Jane]EPR/Clinical	T06	HT-1	201003301010	success
Bob	Cardiologist	write	[Jane]EPR/Clinical	T07	HT-1	201003301020	success
John	GP	read	[Jane]EPR/Clinical	T01	HT-1	201004151210	success
John	GP	write	[Jane]EPR/Clinical	T02	HT-1	201004151210	success
John	GP	write	[Jane]EPR/Clinical	T03	HT-1	201004151215	success
John	GP	write	[Jane]EPR/Clinical	T04	HT-1	201004151220	success
Bob	Cardiologist	write	ClinicalTrial/Criteria	T91	CT-1	201004151450	success
Bob	Cardiologist	read	[Alice]EPR/Clinical	T06	HT-10	201004151500	success
Bob	Cardiologist	read	[Jane]EPR/Clinical	T06	HT-11	201004151501	success
			...				
Bob	Cardiologist	read	[David]EPR/Clinical	T06	HT-20	201004151515	success
Bob	Cardiologist	write	ClinicalTrial/ListOfSelCand	T92	CT-1	201004151520	success
Bob	Cardiologist	read	[Alice]EPR/Demographics	T06	HT-21	201004151530	success
			...				
Bob	Cardiologist	read	[David]EPR/Demographics	T06	HT-30	201004151550	success
Bob	Cardiologist	write	ClinicalTrial/ListOfEnrCand	T93	CT-1	201004201200	success
Bob	Cardiologist	write	ClinicalTrial/Measurements	T94	CT-1	201004221600	success
			...				
Bob	Cardiologist	write	ClinicalTrial/Measurements	T94	CT-1	201004291600	success
Bob	Cardiologist	write	ClinicalTrial/Results	T95	CT-1	201004301200	success

Fig. 4. Audit Trail

3.5 Alignment

In the previous sections, we introduced data protection policies, organizational processes, and audit trails. Although these components make use of the same concepts, such concepts are often specified at a different level of abstraction. In this section, we discuss how they are related to each other.

Audit trails usually capture and store information at a lower level of abstraction than organizational processes. While the basic components of organizational processes are tasks, the basic components of audit trails are actions. Accomplishing a task may require a user to execute a number of actions. Thereby, there is a 1-to-n mapping between tasks and log entries: one task can be associated with multiple log entries. One can think to bring organizational processes and audit trails at a comparable level of abstraction, for instance, by specifying a process in terms of actions or by annotating each task in the process with the actions that can be executed within the task. However, these approaches would require several efforts in the specification of organizational processes as well as affect their readability and understanding. To address this issue, we allow for every action executed within the tasks active at a certain time (when checking the compliance of the actual data usage with the intended purpose as described in Section 4). However, this leaves risks of unauthorized access to data. To prevent unauthorized access while keeping the management of organizational processes simple, a mechanism for purpose

control should be complemented with a preventive enforcement mechanism that verifies access requests in isolation (i.e., independently from other requests).

Languages for representing business processes often rely on information that is not available in audit trails. For instance, the COWS representation of organizational processes founds on the idea that the evolution of the transition system completely characterizes the evolution of the process. Accordingly, transition systems contain information about the management of gateways and the occurrence of non-observable events, which is not recorded in audit trails. To define a suitable mapping between log entries and COWS labels, we distinguish the information that is IT observable by defining the set of *observable labels* $\overline{L}$ as a subset of the set of labels L, i.e., $\overline{L} \subset L$. In particular, labels in $\overline{L}$ specify labels representing the execution of a task q by a partner r (i.e., synchronization labels of the form $r \cdot q$) and labels representing errors (i.e., $sys \cdot Err$). Summing up, the set of observable labels is

$$\overline{L} = \{r \cdot q \mid r \in \mathcal{R} \text{ and } q \in \mathcal{Q}\} \cup \{sys \cdot Err\}.$$

Our definition of observable labels reflects the fact that we assume that only the execution of tasks and error events are IT observable. In case other activities can be logged by the system (e.g., message flows), the definition above should be extended to include the corresponding labels.

How the system determines the purpose for which an action was performed (i.e., the task and case in a log entry) is a critical issue as it is necessary to link the actions to the appropriate process instance. We assume that information about the purpose is available. Different solutions can be adopted to collect this information. For instance, most IT systems based on transactional systems such as WFM, ERP, CRM and B2B systems are able to record the task and the instance of the process [13]. Here, the system itself is responsible to determine the context of an action and store it in the log entry. A different approach is proposed in [2–4] where users are required to specify the purpose along with the access request. This approach is also adopted in existing EPR systems like DocuLive which require users to provide the reason for the access and record that reason in audit trails [20]. We assume that the purpose specified in the access request (i.e., q and c in Def. 2), which can be determined either by the user or by the system, is recorded in the corresponding log entry. In the next section, we present an algorithm for verifying if data have actually been processed for that purpose.

4 Compliance with purpose specification

The aim of purpose control is to guarantee that personal data are not processed in ways that are incompatible with the intended purpose of data. We identify two main issues to be addressed in order to ensure compliance to purpose control: (a) data access should be authorized, and (b) access purposes should be specified correctly and legally. The first issue is addressed by Def. 3 which provides a way to evaluate access request against data protection policies. In particular, this definition guarantees that the data requester has the permission necessary to access the data.

However, the real challenge in ensuring purpose control is to validate the access purpose specified in the access request, i.e. determining whether data were in fact processed for that purpose. In our scenario, the cardiologist legitimately accessed patient

Algorithm 1: Compliance procedure

input : a state s, an audit trail l
output: bool

let $conf\_set = \{(s, empty, \mathsf{WeakNext}(s))\}$;
let $next\_conf\_set = null$;
while $l \neq null$ **do**
  let $l = e * l'$;
  let $r \in \mathsf{R}$ s.t. $e.role \leq_R r$;
  let $found = false$;
  forall $conf \in conf\_set$ **do**
    if $((r, e.task) \notin conf.active\_tasks) \vee (e.status = failure))$ **then**
      forall $(label, state, active\_task) \in conf.next$ **do**
        if $((label = r \cdot e.task) \wedge (e.status = success)) \vee ((label = sys \cdot Err) \wedge (e.status = failure))$ **then**
          $found = true$;
          $next\_conf\_set+ = (state, active\_task, \mathsf{WeakNext}(state))$;
      end
      else
        $found = true$;
        $next\_conf\_set+ = conf$;
    let $l = l'$;
    $conf\_set = next\_conf\_set$;
    $next\_conf\_set = null$;
  end
  if $\neg found$ **then** return false;
end
return true;

data for healthcare treatment and then used those data for clinical trial, which leads to a privacy infringement. To detect re-purposing of data, it is necessary to analyze the actual usage of data. Therefore, for each case in which the object under investigation was accessed, we determine if the portion of the audit trail related to that case is a valid execution of the process implemented by an organization to achieve the corresponding purpose using Algorithm 1.

The algorithm takes as input the COWS service representing the purpose and a finite (portion of) audit trail and determines whether the LTS associated to that service accepts the audit trail, i.e. the audit trail corresponds to a trace of the LTS. The key point of the algorithm is to determine if a log entry can be simulated at a certain point of the execution of the process. Hence we introduce the concept of configuration to represent the current state, the tasks active in that state, and the states reachable from the current state together with the set of active tasks in those states.

Definition 6. *Let S be the set of COWS services, $\mathcal{R}$ the set of roles, $\mathcal{Q}$ the set of tasks, and $\overline{L}$ the set of observable labels. A* configuration $conf$ *is a triple* $(state, active\_tasks, next)$ *where* $state \in S$ *represents the current state,* $active\_tasks \in 2^{(\mathcal{R} \times \mathcal{Q})}$ *represents the set of active tasks in the current state, and* $next \in 2^{(\overline{L} \times S \times 2^{(\mathcal{R} \times \mathcal{Q})})}$ *represents the possible states that can be reached from the current state executing* $l \in \overline{L}$ *together with the corresponding active tasks. Hereafter, we denote the components of a configuration as* $conf.state$, $conf.active\_tasks$, *and* $conf.next$, *respectively.*

The initial configuration consists of the state s representing a given process. Because a BPMN process is always triggered by a start event [12], the set of active tasks in the

initial configuration is empty. The $conf.next$ is computed using function WeakNext. This function takes state s as input and returns the set of states S reachable from s with exactly one observable label. Intuitively, this function explores the process and determines which activities can be executed and the states that are reachable executing such activities. Consider, for instance, the LTS in Fig. 5 where we use l for labels in $\overline{L}$ and $\neg l$ for labels in $L \setminus \overline{L}$. Function WeakNext(s) returns states s_1, s_2, and s_3. The reachable states are computed on the basis of the sequence and message flow. This requires analyzing the gateways which the token(s) goes through. For instance, parallel gateways (i.e., AND gateways) create parallel flow. Accordingly, WeakNext returns the states reachable following all the flows coming out from the gateway. On the other hand, inclusive decision gateways (i.e., OR gateways) are locations where the sequence flow can take one or more alternative paths. Therefore, the set of reachable states includes states that allow the execution of every possible combination of alternatives. For each reachable state, the function also computes the set of tasks active in that state. WeakNext can be implemented on top of CMC [21], an on-the-fly model checker and interpreter for COWS. This tool supports the derivation of all computations originating from a COWS process in automated way.

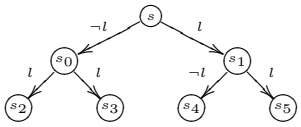


Fig. 5. WeakNext

Algorithm 1 extracts an entry e from the audit trail (line 4), and, for each configuration $conf$ in $next\_conf\_set$, it verifies whether the executed activity (i.e., $(r, e.task)$ where role r is a generalization of $e.role$ according to the role hierarchy) is active and succeeded. If it is the case, the task remains active and the configuration is added to the set of configurations to be considered in the next iteration ($next\_conf\_set$) (line 16). Otherwise, if the action succeeded and the execution of the task makes it possible to reach a reachable state, a new configuration for that state is created and added to $next\_conf\_set$. Similarly, if the activity failed and the failure leads to a reachable state, a new configuration for that state is created and added to $next\_conf\_set$.

The computation terminates with *false* (i.e., an infringement is detected), if the entry cannot be simulated by the process (line 21). Otherwise, the computation proceeds until the audit trail has been completely analyzed. If infringements are not detected, the computation terminates with *true* (line 23). Note that the analysis of the audit trail may lead the computation to a state for which further activities are still possible. In this case the analysis should be resumed when new actions within the process instance are recorded. However, if a maximum duration for the process is defined, an infringement can be raised in the case where this temporal constraint is violated.

We show now the application of Algorithm 1 to the process in Fig. 1 and the sequence of audit entries of Fig. 4 with case HT-1. Fig. 6 presents a portion of the transition system generated by the algorithm. Here, nodes represent the states visited by the algorithm together with the set of active tasks; edges represent observable transitions from the states. The number on the edge indicates the path of the transition system (i.e., the order of log entries).

It is worth noting that the failure of task $T02$ (step 3) leads to a state in which no tasks are active ($St4$). This state corresponds to a "suspension" of the process awaiting

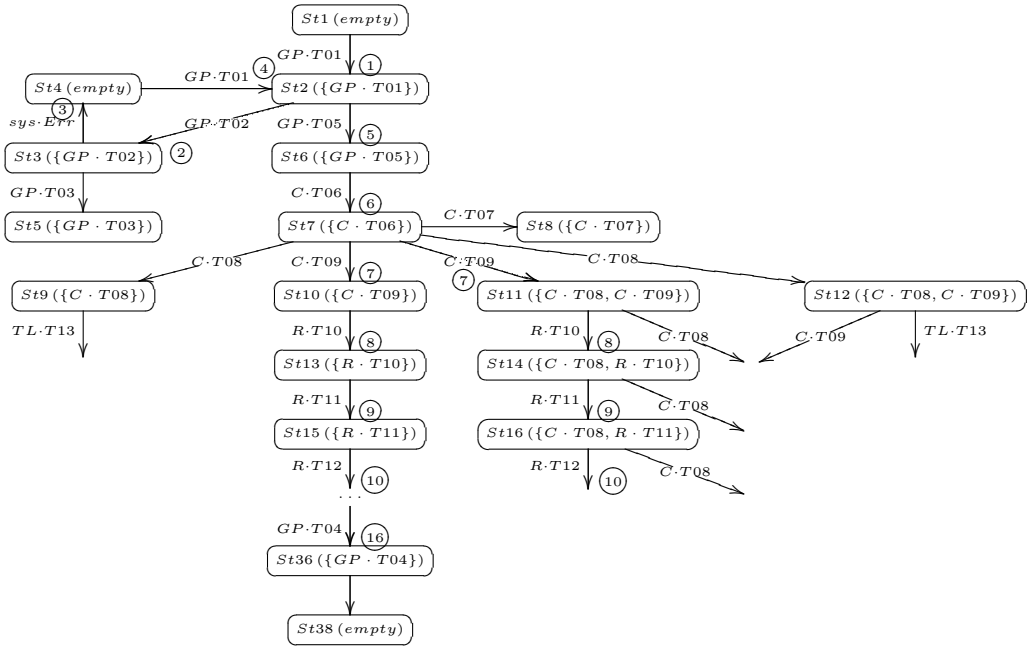


Fig. 6. Portion of the transition system visited by Algorithm 1

the proper activities ($GP \cdot T01$) to restore it. Moreover, one can notice that five states are reachable from state $St7$. This situation is due to the combination of exclusive decision gateway $G2$ and inclusive decision gateway $G3$ (see Fig. 1). In particular, $G3$ allows four possible states, one in which the cardiologist (C in Fig. 6) ordered only lab tests ($St9$), one in which he ordered only radiology scans ($St10$), and two in which he ordered both ($St11$ and $St12$). The difference between these two states lies in the activity executed to reach them. The next log entry refers to the radiologist verifying counter-indications ($R \cdot T10$). States $St10$ and $St11$ allow the execution of that activity; therefore, both states are considered in the next iteration. The algorithm proceeds until the audit trail is fully simulated by the process (step 17). As the portion of the audit trail corresponding to HT-1 is completely analyzed without deviations from the expected behavior, no infringement is detected by the algorithm.

Besides for HT-1, Jane's EPR has been accessed for case HT-11 (see Fig. 4). If we apply the algorithm to the portion of the audit log corresponding to that case (only one entry), we can immediately see that it does not correspond to a valid execution of the HT process. Therefore, the algorithm returns *false* indicating the presence of an infringement: Jane's EPR was accessed for healthcare treatment (HT-11), but it was not processed for that purpose. The cardiologist might have launched a mimicry attack to cover the infringement. However, achieving a purpose requires the execution of a number of tasks; each task should be executed by a user who is allowed to hold the role to which the task is assigned. Therefore, a single user cannot simulate the whole pro-

cess alone, but he has to collude with other users to make a mimicry attack successful. Moreover, if the cardiologist reuses a previous case as the reason for the request (i.e., HT1 instead of HT11), the attack would succeed only in very restricted time windows: the unlawful access has to be in conjunction with a legitimate access, otherwise Algorithm 1 recognizes that the audit trace is not a valid execution of the process. This threat can be partially mitigated by limiting multi-tasking, i.e. a user have to complete an activity before starting a new activity.

5 Properties of the Algorithm

In the previous section, we have proposed an algorithm that can either confirm that the data were processed for the intended purpose or detect privacy infringements. In this section, we discuss the termination, soundness and completeness of the algorithm. Proofs are presented in Appendix B.

The critical part of Algorithm 1 is function WeakNext, because, given a COWS service s, it has to generate and explore (part of) the possibly infinite transition system $LTS(s)$ associated to s; thereby, we have to guarantee the termination of WeakNext. We start by giving some basic definitions.

Given a LTS $\Omega = (s, S, L, \rightarrow)$, a trace is a possibly infinite sequence of $S \times L$ pairs $\sigma \equiv (s, l_0), (s_0, l_1) \ldots (s_n, l_{n+1}) \ldots$ describing a trajectory of the LTS, also denoted as $s \xrightarrow{l_0} s_0 \xrightarrow{l_1} \ldots s_n \xrightarrow{l_{n+1}} \ldots$. The possibly infinite set of traces of Ω is $\Sigma(\Omega)$.

A formal definition of the set of states computed by function WeakNext follows.

Definition 7. *Let s be a COWS service and $\overline{L}$ the set of observable labels for s. Then,* $\mathsf{WeakNext}(s) = \{s' \mid \exists k < \infty . s \xrightarrow{l_0} \ldots \xrightarrow{l_k} s_k \xrightarrow{l} s' \wedge \forall i \leq k \,.\, l_i \notin \overline{L} \wedge l \in \overline{L}\}$.

Given a COWS service s, function $\mathsf{WeakNext}(s)$ is decidable w.r.t. the set of observable labels $\overline{L}$ if, for each trace from s, it is not possible to perform an infinite number of transitions with label in $L \setminus \overline{L}$. The following generalizes this concept.

Definition 8. *Let $\Omega = (s, S, L, \rightarrow)$ be a LTS and $M \subseteq L$ a set of labels. A trace $\sigma \equiv (s, l_0), (s_0, l_1) \ldots (s_n, l_{n+1}) \ldots$ in $\Sigma(\Omega)$ is* finitely observable *w.r.t M iff $\exists n < \infty \,.\, l_n \in M$ and $\forall j > n \,.\, (l_j \in M \Rightarrow \exists k < \infty \,.\, l_{j+k} \in M)$. The set of finitely observable traces of Ω is denoted as $\Sigma^{FO}(\Omega)$. If $\Sigma(\Omega) = \Sigma^{FO}(\Omega)$, Ω is a* finitely observable LTS *w.r.t M.*

A finitely observable transition system w.r.t. a set of labels M could express infinite behaviors, but, within a finite time period it is possible to observe a label in M. This is the idea underlying Algorithm 1. In particular, given a task active at a certain step of the execution of the process, the algorithm determines what are the possible sets of active tasks in the next step. The definition of finitely observable transition system allows us to state a first important result.

Proposition 1. *Given a COWS service s and the set of observable labels $\overline{L}$ for s, if $LTS(s)$ is finitely observable w.r.t. $\overline{L}$, then* $\mathsf{WeakNext}(s)$ *is decidable on $\overline{L}$.*

Although COWS is expressive enough to represent BPMN processes, in order to guarantee the decidability of WeakNext we have to restrict our target to the set of BPMN processes whose transition system is finitely observable w.r.t. the set of observable labels. Hereafter, we say that a BPMN process p is *well-founded* w.r.t. a set of labels M if $LTS(s)$ is finitely observable w.r.t. M, where s is the COWS encoding of p. Intuitively, a BPMN process p is well-founded if every cycle in p has at least one activity which is observable. Given the definition of observable labels $\overline{L}$ in Section 3.5, a BPMN process is therefore well-founded if every sequence flow path ending in a cycle contains at least a task or an event handling errors. Restricting the analysis to well-founded processes does not impose a serious limitation in practice. It avoids those degenerate cases where the process could get stuck because no task or event handler is performed but the process is not ended. An example is a BPMN process with a cycle formed only by gates. Note that non well-founded processes can be detected directly on the diagram describing the process.

From the above considerations, we can state the following corollary.

Corollary 1. *Let p be a well-founded BPMN process, s the COWS service encoding p, $LTS(s) = (s, S, L, \rightarrow)$, and $\overline{L} \subseteq L$ the set of observable labels for s. Then,* WeakNext *terminates for all $s' \in S$.*

We now prove that Algorithm 1 terminates for every COWS service s encoding a well-founded BPMN process. If we consider an audit trail l of length k, Algorithm 1 explores only a finite portion of the transition system of s to verify if an entry e of l is accepted, because of Corollary 1. The idea is that, being k finite, Algorithm 1 explores a finite portion of $LTS(s)$, and therefore terminates.

Theorem 1. *Let p be a well-founded BPMN process, s the COWS service encoding p, $\overline{L}$ the set of observable labels for s, and l an audit trail of length k. Then, Algorithm 1 on (s, l) terminates.*

The following result demonstrates the correctness of Algorithm 1.

Theorem 2. *Let s be a COWS service encoding a well-founded BPMN process and l an audit trail. Algorithm 1 on (s, l) returns $true$ iff there exists a trace $\sigma \in \Sigma(LTS(s))$ such that σ accepts l.*

The results presented in this section allow us to conclude that, within the boundaries defined by a well-founded BPMN process, Algorithm 1 can always decide if a finite audit trail raises concerns about infringement of purpose specification.

6 Related Work

It is largely recognized that traditional access control is insufficient to cope with privacy issues [1, 2, 22]. Karjoth et al. [23] identify the need of three additional elements (i.e., *purpose*, *condition*, and *obligation*), beside the basic authorization elements (i.e., subject, object, and action). Based on this observation, a number of models, languages and standards tailored to specify data protection policies have been proposed in the last decade [2, 4, 5, 8, 24–26]. In this section, we discuss existing proposals based on

the concept of purpose. Works on obligations are complementary to our work as obligations are intended to address different data protection requirements (e.g., retention period).

Existing purpose-based frameworks [2, 4, 5, 24] treat the intended purpose as a label attached to data. Upon receiving an access request, they match the access purpose against the label attached to the requested data. However, they rely on the fact that the requester specifies the purpose legally, implying complete trust on the requester. Few researchers have addressed the problem of validating the access purpose. For instance, Byun and Li [4] specify the roles that can make a request for a given purpose. However, this approach is preventive and does not solve the problem of re-purposing the data. In contrast, we propose to specify the operational behavior of purposes, which makes it possible to analyze the actual usage of data with respect to purpose specifications.

To best of our knowledge, there is only another framework, the Chain method [27], that attempts to define an operational model for purposes. Here, a privacy policy specifies the "chains of acts" that users are allowed to perform where each act is some form of information handling (i.e., creating, collecting, processing, disclosing); purposes are implicitly defined by the sequences of acts on personal information. Compared to our approach, this corresponds to specifying business processes in terms of actions, introducing an undesirable complexity into process models. Conversely, our solution provides a flexible way to align business processes, data protection policies and audit trails and allows an organization to reuse its business process models for purpose control. In addition, the Chain method has a preventive nature and lacks capability to reconstruct the sequence of acts (when chains are executed concurrently).

Some approaches propose methods for a-posteriori policy compliance [28, 29]. For instance, Cederquist et al. [29] present a framework that allows users to justify their actions. However, these frameworks can only deal with a limited range of access control policies and do not consider the purpose. Agrawal et al. [30] propose an auditing framework to verify whether a database system is compliant with data protection policies. However, their focus is mainly on minimizing the information to be disclosed and identifying suspicious queries rather than verifying data usage.

Techniques for detecting system behaviors that do not conform to an expected model have been proposed in process mining and intrusion detection. In intrusion detection, logs of system calls are analyzed either to detect deviations from normal behavior (anomaly detection) [31] or to identify precise sequences of events that damage the system (misuse detection) [32]. Accordingly, our method can be seen as an anomaly detection technique. Process mining [33] and, in particular, conformance checking [13] have been proposed to quantify the "fit" between an audit trail and a business process model. These techniques, however, work with logs in which events only refer to activities specified in the business process model. Consequently, they are not able to analyze the compliance with fine-grained data protection policies. Moreover, they are often based on Petri Nets. This formalism does not make it possible to capture the full complexity of business process modeling languages such as BPMN. Existing solutions based on Petri Nets either impose some restrictions on the syntax of BPMN (e.g., avoiding cycles), or define a formal semantics that deviate from the informal one. Conversely, we have adopted COWS [10] for the representation of organizational processes. This

language has been proved to be suitable for representing a large set of BPMN constructs and analyzing business processes quantitatively [16].

7 Conclusions

Organizations often make use of business process models to define how organizational goals should be achieved. In this paper, we proposed to associate the purpose defined in data protection policies to such process models. This representation makes it possible to verify the compliance of the actual data usage with purpose specifications by determining whether the audit trail represents a valid execution of the processes defined by the organization to achieve a certain purpose. We expect that the audit process is tractable and scales to real applications. Intuitively, the complexity of the approach is bound to the complexity of Algorithm 1. Indeed, Algorithm 1 is independent from the particular object under investigation so that it is not necessary to repeat the analysis of same process instance for different objects. In addition, the analysis of process instances is independent from each other, allowing for massive parallelization. A detailed complexity analysis of the algorithm is left for future work, but our first experiments show encouraging performances.

The work presented in this paper suggests some interesting directions for the future work. The proposed approach alone may not be sufficient when we consider the human component in organizational processes. Process specifications may contain human activities that cannot be logged by the IT system (e.g., a physician discussing patient data over the phone for second opinion). These silent activities make it not possible to determine if an audit trail corresponds to a valid execution of the organization process. Therefore, we need a method for analyzing user behavior and the purpose of data usage when audit trails are partial. In addition, many application domains like healthcare require dealing with exceptions. For instance, a physician can take actions that diverge from the procedures defined by a hospital to face emergency situations. On one side, preventing such actions may be critical for the life of patients. On the other side, checking every occurrence of emergency situations can be costly and time consuming. To narrow down the number of situations to be investigated, we are complementing the presented mechanism with metrics for measuring the severity of privacy infringements.

References

1. P. Guarda and N. Zannone, "Towards the Development of Privacy-Aware Systems," *Information and Software Technology*, vol. 51, no. 2, pp. 337–350, 2009.
2. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic Databases," in *Proceedings of the 28th International Conference on Very Large Data Bases*. Morgan Kaufmann, 2002, pp. 143–154.
3. P. Ashley, S. Hada, G. Karjoth, and M. Schunter, "E-P3P privacy policies and privacy authorization," in *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*. ACM, 2002, pp. 103–109.
4. J.-W. Byun and N. Li, "Purpose based access control for privacy protection in relational database systems," *VLDB J.*, vol. 17, no. 4, pp. 603–619, 2008.

5. F. Massacci, J. Mylopoulos, and N. Zannone, "Hierarchical Hippocratic Databases with Minimal Disclosure for Virtual Organizations," *VLDB J.*, vol. 15, no. 4, pp. 370–387, 2006.
6. D. Catteddu and G. Hogben, "Cloud Computing – Benefits, risks and recommendations for information security," European Network and Information Security Agency (ENISA), Report, 2009.
7. B. Daskala, "Being diabetic in 2011 – Identifying Emerging and Future Risks in Remote Health Monitoring and Treatment," European Network and Information Security Agency (ENISA), Report, 2009.
8. G. Karjoth, M. Schunter, and M. Waidner, "Platform for Enterprise Privacy Practices: Privacy-enabled Management of Customer Data," in *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies*, ser. LNCS, vol. 2482. Springer, 2002, pp. 69–84.
9. C. Lovis, S. Spahni, N. Cassoni, and A. Geissbuhler, "Comprehensive management of the access to the electronic patient record: Towards trans-institutional networks," *Int. J. of Medical Informatics*, vol. 76, no. 5-6, pp. 466–470, 2007.
10. A. Lapadula, R. Pugliese, and F. Tiezzi, "Calculus for Orchestration of Web Services," in *Proceedings of the 16th European Symposium on Programming*, ser. LNCS, vol. 4421. Springer, 2007, pp. 33–47.
11. OASIS, "Web Services Business Process Execution Language – Version 2.0," OASIS Standard, 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
12. Object Management Group, "Business Process Modeling Notation (BPMN) Specification (version 1.2)," OMG document, 2009. [Online]. Available: http://www.omg.org/spec/BPMN/1.2/
13. A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, 2008.
14. "Enterprise Security and Privacy Authorization (XSPA) Profile of XACML v2.0 for Healthcare," Committee Draft, 2008. [Online]. Available: http://xml.coverpages.org/xspa-xacml-profile-CD01-29664.pdf
15. G. Plotkin, "The origins of structural operational semantics," *J. Log. Algebr. Program*, vol. 60, pp. 3–15, 2004.
16. D. Prandi, P. Quaglia, and N. Zannone, "Formal analysis of BPMN via a translation into COWS," in *Proceedings of the 10th International Conference on Coordination Models and Languages*, ser. LNCS, vol. 5052. Springer, 2008, pp. 249–263.
17. Office of the National Coordinator for Health Information Technology, "Electronic Health Records and Meaningful Use," 2010. [Online]. Available: http://healthit.hhs.gov/portal/server.pt/community/healthit_hhs_gov__meaningful_use_announcement/2996
18. D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Trans. Storage*, vol. 5, no. 1, pp. 1–21, 2009.
19. B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 2, pp. 159–176, 1999.
20. L. Rostad and O. Edsberg, "A study of access control requirements for healthcare systems based on audit trails from access logs," in *Proceedings of the 22nd Annual Computer Security Applications Conference*. IEEE Computer Society, 2006, pp. 175–186.
21. A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, and F. Tiezzi, "A model checking approach for verifying COWS specifications," in *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering*, ser. LNCS, vol. 4961. Springer, 2008, pp. 230–245.
22. Q. He and A. I. Antón, "A Framework for Modeling Privacy Requirements in Role Engineering," in *Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality*, 2003, pp. 137–146.

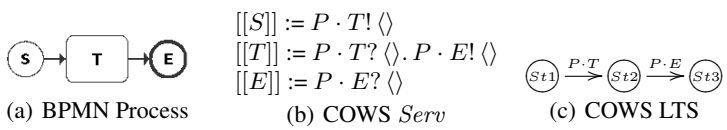
23. G. Karjoth and M. Schunter, "A Privacy Policy Model for Enterprises," in *Proceedings of the 15th IEEE Workshop on Computer Security Foundations*. IEEE Computer Society, 2002, pp. 271–281.
24. M. Backes, G. Karjoth, W. Bagga, and M. Schunter, "Efficient comparison of enterprise privacy policies," in *Proceedings of the 2004 ACM Symposium on Applied Computing*. ACM, 2004, pp. 375–382.
25. M. Hilty, D. A. Basin, and A. Pretschner, "On Obligations," in *Proceedings of the 10th European Symposium on Research in Computer Security*, ser. LNCS, vol. 3679. Springer, 2005, pp. 98–117.
26. OASIS, "eXtensible Access Control Markup Language (XACML) Version 2.0," OASIS Standard, 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
27. S. S. Al-Fedaghi, "Beyond purpose-based privacy access control," in *Proceedings of the 8th Conference on Australasian Database*. Australian Computer Society, Inc., 2007, pp. 23–32.
28. C. Fournet, N. Guts, and F. Z. Nardelli, "A formal implementation of value commitment," in *Proceedings of the 17th European Symposium on Programming*, ser. LNCS, vol. 4960. Springer, 2008, pp. 383–397.
29. J. G. Cederquist, R. J. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini, "Audit-based compliance control," *Int. J. Inf. Sec.*, vol. 6, no. 2-3, pp. 133–151, 2007.
30. R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant, "Auditing Compliance with a Hippocratic Database," in *Proceedings of the 30th International Conference on Very Large Data Bases*. VLDB Endowment, 2004, pp. 516–527.
31. H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, and W. Gong, "Anomaly detection using call stack information," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2003, pp. 62–75.
32. S. Kumar and E. H. Spafford, "A Pattern Matching Model for Misuse Intrusion Detection," in *Proceedings of the 17th National Computer Security Conference*, 1994, pp. 11–21.
33. W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

A Encoding of BPMN in COWS

In this section we present some examples that provide the intuition underlying the COWS semantics and the encoding of BPMN processes into COWS specification.

Consider the simple process in Fig. 7(a); it is composed by a start event S, a task T, and an end event E within a pool P. The corresponding COWS service is $Serv = [[S]] \mid [[T]] \mid [[E]]$, where services $[[S]]$, $[[T]]$, and $[[E]]$ are defined in Fig. 7(b). Service $[[S]]$ gives the control to task T in pool P, written as $P \cdot T! \langle\rangle$. Service $[[T]]$ receives the control ($P \cdot T? \langle\rangle$) and then (represented as infix dot ". ") gives the control to the end event E within pool P ($P \cdot E! \langle\rangle$). Finally, $[[E]]$ closes the flow receiving the control $P \cdot E? \langle\rangle$. The LTS associated with service $Serv$ (Fig. 7(c)) gives a compact representation of the possible paths of tokens within the process of Fig. 7(a). In this simple case, only a single path is possible.

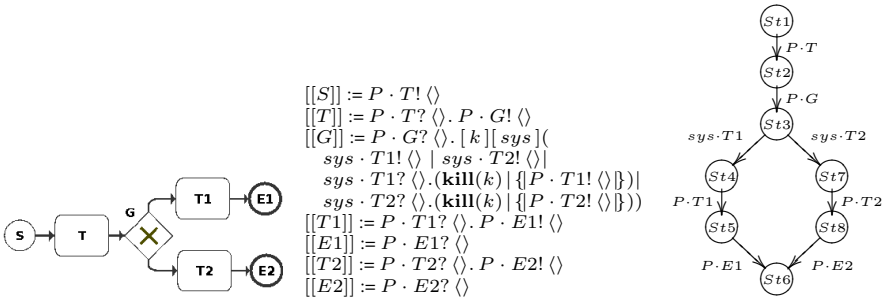
An example involving a gateway is presented in Fig. 8(a). Here, when reaching the exclusive gateway G, the token can follow only one flow, either through $T1$ or through $T2$. Fig. 8(b) shows the encoding of the process in COWS. Note that the encoding of G, $[[G]]$, makes use of $\mathbf{kill}(k)$: when an alternative is selected, a killer signal is sent



(a) BPMN Process (b) COWS $Serv$ (c) COWS LTS

Fig. 7. A Simple BPMN process in COWS

to prevent the other alternative to be executed. This is evident in Fig. 8(c) where state $St6$ is reached by running either $T1$ or $T2$, but there is no path where both $T1$ and $T2$ are executed. We use the private name sys to avoid interference between services or between different executions of the same service.



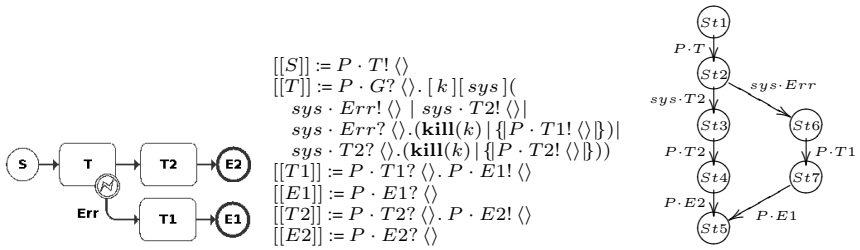
(a) BPMN Process (b) COWS $Serv$ (c) COWS LTS

Fig. 8. A BPMN Process with Exclusive Gateway

Tasks with an associated error event are another example of sequence flow. The example in Fig. 9(a) models a task T that can either proceed "correctly" to task $T2$ or incur an error *Err* managed by task $T1$. The encoding of the COWS services is reported in Fig. 9(b). Note that the encoding of the BPMN elements S, $T1$, $E1$, $T2$, and $E2$ are the same of the previous example (Fig. 8(b)). We only change the definition of $[[T]]$: when T takes the token, it can either proceed normally invoking $T2$ ($P \cdot T2! \langle\rangle$) or after signaling an error (label "$sys \cdot Err$") proceed invoking $T1$ ($P \cdot T1! \langle\rangle$). The LTS of Fig. 9(c) shows these two possible paths.

The three examples above give some hints about the representation of an organizational process through COWS. To have a complete insight in the COWS services used to model the process in Fig. 1 we have to discuss two further topics, namely *message flows* and *service replication*.

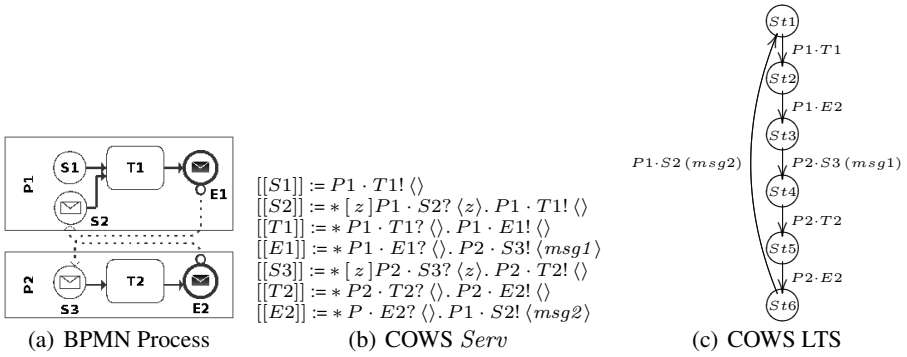
Message flows are represented as communications over endpoints belonging to different pools. An example is given in Fig. 10. Here, events $S2$ and $S3$ are message start events, i.e. events that trigger the start of the process upon the receipt of a message, and $E1$ and $E2$ are message end events, i.e. events that send a message to a participant at the conclusion of the process. When the end event $E1$ is triggered, it sends a message



(a) BPMN Process (b) COWS $Serv$ (c) COWS LTS

Fig. 9. A BPMN Process with Error Event

$msg1$ to the start event $S3$ in pool $P2$; similarly, end event $E2$ sends a message $msg2$ to the start event $S2$ in pool $P1$. Upon the receipt of the message, $S2$ and $S3$ start the corresponding process. The COWS encoding of the process in Fig. 10(a) is presented in Fig. 10(b) and the corresponding LTS in Fig. 10(c).



(a) BPMN Process (b) COWS $Serv$ (c) COWS LTS

Fig. 10. A BPMN Process with Message Flow and Cycles

In general, it is not known in advance how many times a service is invoked during the execution of a process. An example of this is given by cycles. Cycles are closed paths in the process; they can consist of sequence flows, as the cycle involving $T01$, $G1$, $T02$, and again $T01$ in Fig. 1, or combination of sequence and message flows, as the cycle involving $S2$, $T1$, $E1$, $S3$, $T2$, $E2$, and again $S2$ in Fig. 10(a). A cycle involves the restart of the process from a certain activity. Consider the example of Fig. 1: if the GP is not able to make a diagnosis (*Err* in task $T02$), the process has to restart from $T01$. To address this issue, we prefix COWS services with the replication operator $*$ (Fig. 10(b)). This operator makes multiple copies of the COWS service; each copy corresponds to an invocation of the service.

B Proofs

Proof of Proposition 1 If $LTS(s)$ is finitely observable w.r.t. $\overline{L}$, then for each trace

$$s \xrightarrow{l_0} s_0 \xrightarrow{l_1} s_1 \dots s_n \xrightarrow{l_{n+1}} s_{n+1} \dots$$

there exists a value $k < \infty$ such that $l_k \in \overline{L}$, by Def. 8. This implies that the set of states that can be reached from s with exactly one label in $\overline{L}$ can be computed in a finite number of steps, namely $\mathsf{WeakNext}(s)$ is decidable on $\overline{L}$. □

Proof of Theorem 1 The proof is by induction on the length k of $l = e_1 e_2 \dots e_k$.

Base Step: Let $l = e_1$. A BPMN process is always triggered by a start event [12]. Therefore, the initial configuration has the form $conf = (s, empty, \mathsf{WeakNext}(s))$. By definition $LTS(s)$ is a finitely observable LTS. Therefore, each trace $\sigma \equiv (s, l_0), (s_0, l_1) \dots (s_n, l_n) \dots$ in $\Sigma(LTS(s))$ is such that $\exists j < \infty . l_j \in \overline{L}$. If $\Sigma(LTS(s))$ is empty, $\mathsf{WeakNext}(s)$ returns an empty set. In this case, the algorithm does not enter into the forall of line 7 and the variable *found* remains false. Thereby, the algorithm exits at line 22 with *false*. If there exists $(l_j, s', active\_tasks) \in \mathsf{WeakNext}(s)$ such that $l_j = r \cdot e_1.task$ with $e_1.role \leq_R r$, or $l_j = sys \cdot Err$ and $e_1.status = failure$, Algorithm 1 returns *true* at line 24. Otherwise, Algorithm 1 returns *false* at line 22.

Inductive Step: Let $l = e_1 \dots e_k$ with $k > 1$. By the inductive hypothesis, Algorithm 1 terminates on the audit trail $l^{(k-1)} = e_1 e_2 \dots e_{k-1}$. Let $conf\_set$ be the set of actual configurations. If $conf\_set = \emptyset$, the variable *found* remains *false*, and Algorithm 1 returns *false* at line 22. If there exists a configuration $conf \in conf\_set$ such that $(r, e_k.task) \in conf.active\_tasks$ with $e_k.role \leq_R r$ and $e_k.status = success$, the configuration is added to the set of configurations to be considered in the next iteration (line 16). As l is completely analyzed and *found* is equal to *true* (line 15), Algorithm 1 returns $true$ at line 24. If there exists a configuration $conf \in conf\_set$ such that $(r \cdot e_k.task, s', at) \in conf.next$ with $e_k.role \leq_R r$ or $(sys \cdot Err, s', at) \in conf.next$ and $e_k.status = failure$, variable $found$ becomes $true$ at line 11. Then, Algorithm 1 returns $true$ at line 24. Otherwise, if the entry e_k does not correspond either to any task in $conf.active\_tasks$ or to any label in $conf.next$, Algorithm 1 returns $false$ on line 22. Therefore, by induction, Algorithm 1 terminates for l.

□

Proof of Theorem 2 The correctness is essentially given in term of soundness (forwards proof) and completeness (backwards proof). We show the implications separately. We only sketch the proof of the theorem, which requires a double induction on the length of l and on the structure of s.

($\Longrightarrow$) Let $l = e_1 e_2 \dots e_k$ be of length k. Algorithm 1 on (s,l) returns *true* only if the while cycle of line 3 ends and line 24 is executed. By Theorem 1, we know that the algorithm and, consequently, the cycle always terminates. Line 24 is executed only if for each $e_i \in l$ either condition on line 8 is not verified (i.e., $e_i.task$ is active and $e_i.status$

is not $failure$) or, by line 10, there exists a configuration $conf \in conf\_set$ that accepts e_i (i.e., $(r \cdot e_i.task, s', at) \in conf.next$ with $e_i.role \leq_R r$ or $(sys \cdot Err, s', at) \in conf.next$ and $e_k.status = failure$). This consideration makes it possible to prove that, at each iteration $i \in [1, k]$ of the while cycle, Algorithm 1 computes the set of traces of $LTS(s)$ that accept the prefix $e_1 \dots e_i$. Therefore, if line 24 is executed, there exists at least one trace σ in $\Sigma(LTS(s))$ that accepts l.

($\Longleftarrow$) To prove the completeness of Algorithm 1, we have to prove that if there is a trace from s that accepts l, then Algorithm 1 on (s, l) gives ***true***. Below, we demonstrate the contra-positive of the previous sentence, i.e. if Algorithm 1 on (s, l) returns ***false***, then there is not a trace from s that accepts l. Given $l = e_1 e_2 \dots e_k$, Algorithm 1 on (s, l) returns ***false*** if there exists an iteration $i \in [1, k]$ of the while cycle in line 3 such that the condition on line 21 is true. This is possible if during iteration i both line 11 and line 15 are not executed. The first condition is verified if, given a $conf \in conf\_set$ such that e_i does not correspond to any task in $conf.active\_tasks$ or it is a failure (i.e., condition on line 8 is true), there is not a triple $(l_m, s_m, at) \in conf.next$ that accepts e_i (condition on line 10 is false). In this case, by Proposition 1, there does not exist a finitely observable trace $(s_j, l_{j+1}) \dots (s_{m-1}, l_m)$ from the current state s_j to s_m such that e_i corresponds to l_m. Line 15 is executed only if e_i corresponds to a task in $conf.active\_tasks$ and it is not a failure (i.e., condition on line 8 is false). No execution of this line implies that there does not exist a $conf \in conf\_set$ where e_1 is active. This means that e_i does not correspond to an active task in the current state s_j. Suppose that at the $i\text{-}th$ iteration of the while cycle, Algorithm 1 has already built all the traces accepting $e_1 \dots e_{i-1}$. If Algorithm 1 cannot replay e_i in the process, we can conclude that there is not a trace in $\Sigma(LTS(s))$ accepting $e_1 \dots e_i$, and so l. $\square$