A Declarative Framework for Specifying and Enforcing Purpose-aware Policies

Riccardo De Masellis[1], Chiara Ghidini[2], and Silvio Ranise[2]

[1] Trento RISE, Via Sommarive 18, 38123 Trento, Italy
r.demasellis@trentorise.eu
[2] Bruno Kessler Foundation, Via Sommarive 18, 38123 Trento, Italy
(ghidini|ranise)@fbk.eu

Abstract. Purpose is crucial for privacy protection as it makes users confident that their personal data are processed as intended. Available proposals for the specification and enforcement of purpose-aware policies are unsatisfactory for their ambiguous semantics of purposes and/or lack of support to the run-time enforcement of policies.
In this paper, we propose a declarative framework based on a first-order temporal logic that allows us to give a precise semantics to purpose-aware policies and to reuse algorithms for the design of a run-time monitor enforcing purpose-aware policies. We also show the complexity of the generation and use of the monitor which, to the best of our knowledge, is the first such a result in literature on purpose-aware policies.

1 Introduction

An important aspect of privacy protection is the specification and enforcement of purposes, i.e. users should be confident that their data are processed as intended. For instance, email addresses are used only for billing but not for marketing purposes. Unfortunately, as already observed several times in the literature (see, e.g., [21] for a thorough discussion), both specifying and enforcing purposes turn out to be difficult tasks.

Specification. Following the seminal paper [36], the specification of privacy constraints consists of establishing when, how, and to what extent information about people is communicated to others. In the context of IT systems, this amounts to define policies governing the release of personal data for a given purpose. From a technological point of view, these policies are usually mapped to access control policies augmented with purpose constraints, which we call *purpose-aware policies* (sometimes called privacy-aware access control policies in the literature, see, e.g., [11]). In this paper, we do not consider the problem of deriving purpose-aware policies from the high-level and heterogenous privacy requirements . We assume that this has been done and focus instead on the basic building blocks of the models and specification languages underlying the policies. As observed in [20,21], such building blocks are data-centric and rule-centric policies. In the former, every piece of information is associated with the purposes for which it can be used; examples are the policies in [9] or those

expressed in the XACML Privacy Profile.[3] In the latter, rules specify under which conditions subjects can perform some action on a given piece of information for some purpose; examples are those in [25,11] and those expressed in EPAL.[4] For expressivity reasons, both data- and rule-centric policies should be supported for the specification of purpose-aware policies.

One of the most serious problems in purpose-aware policies is the lack of semantics for purposes, which are usually considered as atomic identifiers. This gives rise to arbitrariness in the interpretation of purposes; e.g., if the policy of a company states that emails of users are collected for the purpose of communication, this allows the organization to use emails for both billing and marketing when the majority of users has a strong preference for the first interpretation only. To solve this problem, several works have observed that "*an action is for a purpose if it is part of a plan for achieving that purpose*" [33]. Among the many possible ways to describe plans, one of the most popular is to use workflows, i.e. collections of activities (called tasks) together with their causal relationships, so that the successful termination of a workflow corresponds to achieving the purpose which it is associated to. We embrace this interpretation of purpose and avoid ambiguities in its specification by using a temporal logic which allows us to easily express the causal relationships among actions in workflows associated to purposes. Additionally, the use of a logic-based framework allows us to express in a uniform way, besides purpose specifications, also authorization (namely, data- and rule-centric) policies together with authorization constraints, such as Separation/Bound of Duties (SoD/BoD). While temporal logics have been used before for the specification of authorization policies (see, e.g., [26]) and of workflows (see, e.g., [13]), it is the first time—to the best of our knowledge—that this is done for both in the context of purpose-aware policies. In particular, the capability of specifying SoD or bod constraints—which are crucial to capture company best practices and legal requirements—seems to be left as future work in the comprehensive framework recently proposed in [21].

Enforcement. Enforcing purpose-aware policies amounts to check if (**C1**) a user can peform an action on a certain data for a given purpose and (**C2**) the purpose for which a user has accessed the data can be achieved.

(**C1**) is relatively easy and well-understood being an extension of mechanisms for the enforcement of access control policies (see, e.g., [14] for an overview) by considering the combined effect of rule- and data-centric policies.

(**C2**) is much more complex than (**C1**) as it requires to foresee if there exists an assignment of users to tasks that allows for the successful termination of the workflow. This is so because—as discussed above—a purpose is associated to a workflow so that its successful execution implies the achievement of the purpose. The problem of checking (offline) if a workflow can successfully terminate, known as the Workflow Satisfiability Problem (WSP), is already computationally expensive with one SoD [34], and moreover, the on-line monitoring of authorization constraints requires to solve several instances of the WSP [12].

[3] `docs.oasis-open.org/xacml/3.0/xacml-3.0-privacy-v1-spec-cd-03-en.pdf`
[4] `www.w3.org/Submission/2003/SUBM-EPAL-20031110`

For purpose-aware policies, this implies that it is necessary to solve an instance of the WSP per user request of executing a task in the workflow associated to a given purpose.

Contributions. The paper provides the following contributions:

- The *specification* of a comprehensive framework for expressing purpose-aware policies which are a combination of data- and rule-centric policies together with workflows augmented with authorization constraints (Section 3 and, in particular, Figure 1). To the best of our knowledge, this is the first time authorization constraints are considered and naturally integrated in a purpose-aware setting.
- The *semantic formalization* of purpose-aware policies as formulas in first-order temporal logic (Section 3.1).
- The provision of *formal techniques* not only for the (on-line) enforcement of purpose-aware policies, but also for their (off-line) analysis, together with decidability and complexity results (Section 4). Proofs of theorems can be found in the full-length version of the work [17].

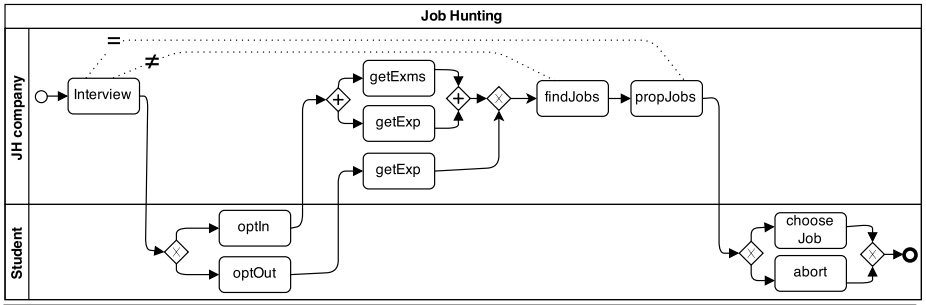
A running example (Section 2) introducing the main issues related to purpose-aware policies is used throught the paper to illustrate the main concepts of our framework. Related work and conclusions are also discussed (Section 5).

2 Running example

We describe a running example, based on *Smart campus*[5], which will be used throughout the paper. Smart campus is a platform in which citizens, institutions and companies can communicate with each other by exchanging data and services. It provides functionalities to access information about transportations, social services, education, and user profiles. These services allow companies to build new applications and thus offer new services to citizens. In this kind of scenario, users should be confident that the services access only the data required to their needs and use them for the right purposes. Additionally, service providers should comply with laws, regulations, and best practices in handling data mandated by local governments, the European Union, and enterprises. In other words, access to personal data must be mediated by appropriate authorization policies augmented with purpose constraints so that only authorized subjects have the right to access certain data for a given purpose. Following an established line of works (see, e.g., [21] for an overview), we assume that the purpose of an action is determined by its relationships with other, interrelated, actions.

For concreteness, we illustrate these ideas by considering the situation in which some personal data of users in the Smart campus platform (namely, the work experience and the academic transcripts) is accessed by JH, a job hunting company, for the purpose of finding jobs to students. JH has deployed in the Smart Campus platform the service depicted of Figure 1 (upper half), specified

[5] `http://www.smartcampuslab.it`



interview	$\Diamond \mathsf{optIn} \vee \Diamond \mathsf{optOut} \wedge \neg(\Diamond \mathsf{optIn} \wedge \Diamond \mathsf{optOut})$	$(\neg(\mathsf{getExms} \vee \mathsf{getExp}) \mathcal{U} (\mathsf{optIn} \vee \mathsf{optOut})) \wedge (\neg \mathsf{getExsm} \mathcal{U} \mathsf{optIn})$	$(\neg \mathsf{findJobs} \mathcal{U} (\mathsf{getExms} \vee \mathsf{getExp})) \wedge (\Diamond \mathsf{findJobs} \wedge \mathsf{findJobs} \rightarrow \bigcirc \mathsf{propJobs})$	$\Diamond(\mathsf{chooseJob} \vee \mathsf{abort})$

Fig. 1: The JobHunting workflow expressed in BPMN (upper half) and as a (partial) set of LTL formulae (lower half).

in the Business Process Model and Notation (BPMN). The swim lane labelled 'Student' contains the activities (also called tasks) that must be executed under the responsibility of the data owner and the swim lane labelled 'JH company' shows the activities that employees of JH are supposed to perform for the purpose of finding some jobs to the data owner. First of all, an employee of JH performs an interview to the student to understand his/her job preferences. Then, the student decides to give or not his/her consent for JH to access his/her academic transcripts by executing either task optIn or task optOut, respectively (the empty diamond before the two tasks in Figure 1 is an exclusive-or gateway). If he/she opts in, an employee of JH can access both his/her academic transcripts and past experience (by executing both getExms and getExp since the diamond containing the plus sign before the two tasks in Figure 1 is a parallel gateway); otherwise, only the student past experiences can be accessed. Based on the interview and the collected personal information, an employee of JH search for jobs the student can be interested in (task findJobs) and some other employee proposes him/her some of them (task propJobs). Finally, the students decides if choosing one of the jobs or to abort the process (tasks chooseJob and abort, respectively). Further authorization constraints are imposed on which employees can execute task interview, findJobs, and proposeJobs: the first two must be executed by different employees to keep the overall process unbiased—this is called a Separation of Duty (SoD) constraint—whereas the first and last tasks must be executed by the same employee so that the student gets in contact with the same person of JH—this is called a Bind of Duty (BoD) constraint. (In Figure 1, these constraints are shown as dotted lines connecting the tasks labelled by the distinct $\neq$ or equal $=$ sign in case of SoD or BoD, respectively.)

To summarize, the workflow specification is used to specify the purpose of an activity (task) with respect to all the others that must be executed for achieving the given purpose. From now on, we assume that a workflow is uniquely associated to a purpose or, equivalently, that the semantics of a purpose is its associated workflow.

Since the tasks in the workflow are executed under the responsibility of a user (e.g., an employee of JH), he/she must have the right to access such data. For instance, the task getExp takes as input the list of past job experiences of the student. The employee of JH executing this task must have the right to access such a list and the student should have given the consent to access this information to (an employee of) JH. In other words, every time an employee of JH asks to execute an activity for the purpose of finding jobs, he/she not only must have the right to do so according to the authorization policy of the company but also the student (data owner) should agree to release the information for the purpose of finding jobs. In other words, there are two types of policies that must be taken into account when granting the right to execute a task to an employee: one is called *rule-centric*, and constrains access by considering subjects, actions, and data objects while the other is called *data-centric*, and is such that data owners constrain access to their data objects for certain purposes only. For instance, in the job hunting scenario, the rule-centric policy specifies that employee bob has the right to read the list of job experiencens of students and the data-centric policy specifies that student sam's academic transcripts can be accessed for the purpose of JobHunting.

Notice the subtle interplay between purposes, described by workflows, and authorization policies. For instance, the execution of certain tasks can modify both rule- and data-centric policies as it is the case of optIn and optOut in Figure 1. When executing the latter, the execution of the task getExms is skipped in the current instance of the workflow despite the fact that sam has agreed to disclose such information according to the above data-centric policy (for which sam's academic transcripts can always be used for the purpose of jobHunting). This flexibility allows us to model certain data directive for privacy (see, e.g., [1]) in which the data owner must explicitly give his/her consent to access his/her personal data, every time it is requested.

Finally, observe that handling purposes in presence of authorization constraints (such as SoD or BoD) requires to solve, at run-time, the *Workflow Satisfiability Problem* (WSP) [12], i.e., to be able to answer the question: does there exist an assignment of authorized users (according to the rule- and data-centric policies) to workflow tasks that satisfies the authorization constraints (SoD or BoD)? The WSP is known to be a computationally expensive activity; it is already NP-hard with one SoD constraint [34]. To make things even more complex, at run-time, we need to solve several instances of the WSP, one per user request of executing a task in a workflow associated to a purpose. Indeed, the WSP returns one possible future execution sequence meeting the constraints and thus, each time the real execution diverges from that one, a new WSP problem taking into

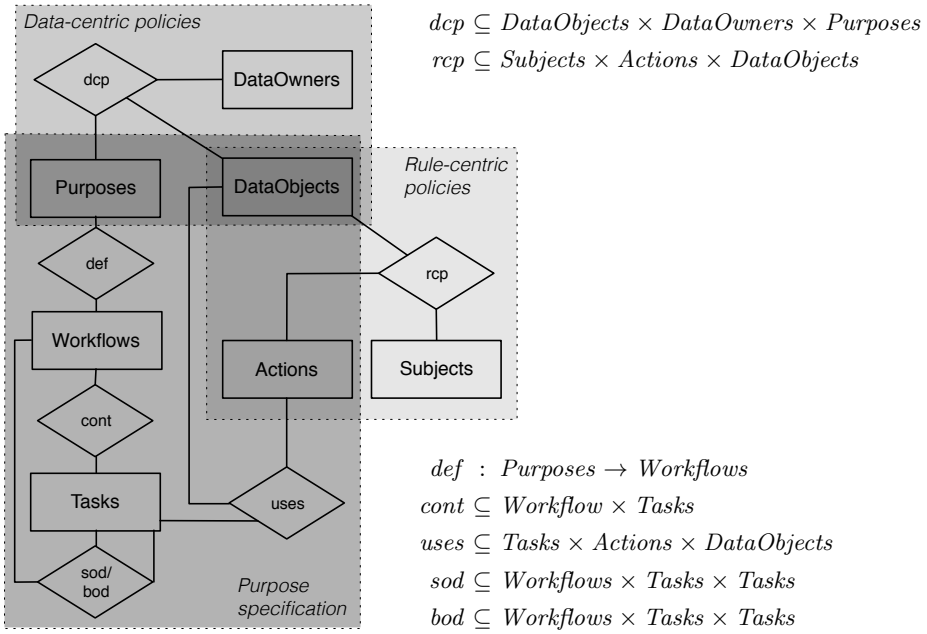


Fig. 2: Conceptual representation of the data-, rule- and purpose-centric policies.

account the real evolution, must be solved. This may happen each time a new request is presented.

3 A Declarative Framework for Purpose-aware Policies

On the left of Figure 2, it is shown an entity-relationship diagram describing the conceptual organization of our framework (rectangles represent sets of entities and diamonds relationships among them). We have *DataOwners* who own *DataObjects* and decide the *Purposes* for which these can be accessed by means of a data-centric policy (relation *dcp*). *Subjects* can perform certain *Actions* on *DataObjects* according to a rule-centric policy (relation *rcp*). *Purposes* are defined (relation *def*) in terms of *Workflows* which are composed of *Tasks* (relation *cont*) and SoD or BoD constraints; each task can perform some *Actions* on *DataObjects* (relation *uses*).

On the right of Figure 2, it is shown the formal characterization of the relationships as subsets of the cartesian products of the appropriate sets of entities. To illustrate, recall the running example in Section 2:

– the rule- and data-centric policies "bob has the right to read the list of job experiencens of students" and "sam's academic transcripts can

be accessed for the purpose of JobHunting" can be specified by relations *rcp* and *dcp* which are such that[6] *rcp*(bob, read, jobExpList) and *dcp*(academicTranscript, sam, jobHunting);
- if φ is the specification of the workflow in the upper half of Figure 1 (we explain below what is φ), then *def*(JobHunting) = φ (notice that *def* is a total function from *Purposes* to *Workflows*, i.e. every purpose is associated to a workflow);
- the SoD and BoD contraints in Figure 1 can be specified by relations *sod* and *bod* such that *sod*(φ, interview, findJobs) and *bod*(φ, interview, propJobs);
- the fact that, for example, tasks interview and optIn are part of the worflow specification φ can be specified by a relation *cont* such that *cont*(φ, interview) and *cont*(φ, optIn);
- the fact that the task interview reads the user profile can be specified by a relation *uses* such that *uses*(interview, read, UserProfile);

We now explain how we specify workflows in our framework. Following the declarative approach in [2], we have chosen Linear-time Temporal Logic (LTL) as the specification language. The main reason for this is two-fold. First, well-known techniques (see, e.g., [22]) are available to translate procedural descriptions of workflows (e.g., that in the upper half of Figure 1), and more in general concurrent systems, to LTL formulae. For instance, the lower half of Figure 1 shows an (incomplete) set of LTL formulae (to be read in conjunction) corresponding to the BPMN workflow in the upper half. The first conjunct on the left means that interview must be the first task to be executed, formula $\neg\mathsf{getExms}\,\mathcal{U}\,\mathsf{optIn}$ in the third conjunct of the figure means that the academic transcripts cannot be accessed if the student has opted out. Formula $\neg\mathsf{findJobs}\,\mathcal{U}(\mathsf{getExms} \vee \mathsf{getExp})$ in the fourth conjunct means that the execution of task findJobs must not happen before the execution of getExms or getExp.

The second reason for chosing LTL to specify workflows is that it allows us to derive a precise semantics of purpose-aware policies and to reuse available techniques for the off-line and on-line verification of formulae to support the analysis of policies at design-time and their enforcement at run-time. Such verification tasks are presented in Section 4. Here we focus on the semantics of purpose-aware policies, starting with the meaning of LTL formulae, which are expressions of the following grammar:

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{O}\varphi \mid \varphi_1\,\mathcal{U}\,\varphi_2 \mid \Box\varphi \mid \Diamond\varphi \quad \text{with } a \in \mathit{Prop}$$

where *Prop* is a set of Boolean variables representing tasks. Intuitively, $\mathsf{O}\varphi$ means that φ holds at the *next* instant, $\varphi_1\,\mathcal{U}\,\varphi_2$ means that at some future instant φ_2 will hold and *until* that point φ_1 holds, $\Box\varphi$ means that φ always holds, and its dual $\Diamond\varphi$ that φ eventually holds. Since we assume workflows to *eventually terminate*, we adopt the finite-trace semantics in [15,16]. The only notable aspect of this semantics (with respect to the standard semantics as given in, e.g., [22]) is that $\mathsf{O}\varphi$ is true iff a next state actually exists *and* it satisfies

[6] Given an n-ary relation R, we write $R(e_1, ..., e_n)$ for $(e_1, ..., e_n) \in R$.

φ. The models of LTL formulae are finite sequences of Boolean assignments to the variables in $Prop$ indexed over natural numbers, which represent instants of a linear and discrete time. The idea is that at a certain time instant, the Boolean variable representing a task is assigned to *true* iff the corresponding task has been executed. As customary in workflow specifications, we assume that one task only is executed at a time. By looking at a sequence of Boolean assignments satisfying a formula, we can thus understand which tasks have been executed and which are not. In other words, a sequence Π of Boolean assignments satisfying a formula φ, in symbols $\Pi \models \varphi$, correspond to a possible execution of the workflow described by φ (see [15,16] for a precise definition). The set of all sequences satisfying a formula φ, i.e. the set of all successful executions of the workflow described by φ, is called its *language* and is denoted by $\mathcal{L}(\varphi)$. It is possible to build an *automaton* (i.e., a finite-state machine) from a formula φ accepting exactly all the traces belonging to $\mathcal{L}(\varphi)$; see again [15,16] for the description of the procedure for doing this.

We can now define the notion of purpose-aware policy as a tuple

$$\mathcal{P} = (DataOwners, Subjects, DataObjects, Actions, Tasks, \\ Workflows, Purposes, dcp, rcp, sod, bod, cont, def)$$

whose components are as explained above.

As it is standard in access control models, we introduce the notion of a request as a tuple (wid, sub, tsk, do, p) where $sub \in Subjects$, $tsk \in Tasks$, $do \in DataOwners$, $p \in Purposes$, and wid belongs to the set Wid of workflow identifiers (allowing us to distinguish among different executions of possibly the same workflow). Intuitively, (wid, sub, tsk, do, p) means that subject sub asks the permission to execute task tsk on the data objects owned by the data owner do in the workflow instance wid for the purpose p. The relation $req \subseteq Wid \times Subjects \times Tasks \times DataOwner \times Purposes$ contains all possible requests.

3.1 Semantics of purpose-aware policies

We explain how a request (wid, sub, tsk, do, p) is granted or denied according to a purpose-aware policy $\mathcal{P}$. The idea is to derive a first-order LTL formula from the LTL formula $def(p)$ constraining the ordering of requests such that the rule-, data-centric policies and SoD and BoD constraints are satisfied. Thus, instead of sequences of Boolean assignments, we consider first-order models which differ for the interpretation of requests only. For the sake of brevity, we do not give a formal semantics of first-order LTL on finite-traces but only some intuitions and refer the interested reader to [19] for the details.

First of all, we observe that every workflow instance can be considered in isolation since the framework presented above allows one to specify only constraints within a workflow instance and not accross instances. For this reason, we introduce an operator to identify requests referring to the same workflow instance wid out of a trace Π containing requests referring to arbitrary workflow instances, i.e.

$\Pi|_{\mathsf{wid}} = req_1(\mathsf{wid}, sub_1, tsk_1, do_1, \mathsf{p}), \ldots, req_n(\mathsf{wid}, sub_n, tsk_n, do_n, \mathsf{p})$ is the trace representing the evolution of the specific workflow instance wid. Notice that requests in $\Pi|_{\mathsf{wid}}$ share the same workflow identifier wid and purpose p whereas subjects, tasks, and data owners may be different. Given a purpose-aware policy $\mathcal{P}$, for each purpose $\mathsf{p} \in Purposes$ such that $def(\mathsf{p}) = \varphi$, we build a (first-order) LTL formula $\Phi_{\mathsf{p}} := \varphi \wedge \Lambda \wedge \Sigma \wedge B$ where

$$\Lambda := \bigwedge_{cont(\varphi,t)} t \leftrightarrow \exists sub, do. \begin{pmatrix} req(\mathsf{wid}, sub, t, do, \mathsf{p}) \wedge \\ \bigwedge_{uses(act,t,obj)} dcp(do, obj, \mathsf{p}) \wedge rcp(sub, act, obj) \end{pmatrix}$$

$$\Sigma := \bigwedge_{sod(\varphi,t_1,t_2)} \xi(t_1, t_2, =) \wedge \xi(t_2, t_1, =)$$

$$B := \bigwedge_{bod(\varphi,t_1,t_2)} \xi(t_1, t_2, \neq) \wedge \xi(t_2, t_1, \neq)$$

$$\xi(t, t', \bowtie) := \Box \forall sub, sub'. \begin{pmatrix} \forall do. req(\mathsf{wid}, sub, t, do, \mathsf{p}) \wedge \\ \Diamond \forall do. req(\mathsf{wid}, sub', t', do, \mathsf{p}) \end{pmatrix} \rightarrow sub \bowtie sub' .$$

Formula Λ says that in order to execute task t for purpose p, we need to check that subject *sub* who has requested to execute it is entitled to do so according to both the rule- and data-centric policies in $\mathcal{P}$, thus formalizing the check (**C1**) in the introduction. Formulae Σ and B encode the SoD and BoD constraints in $\mathcal{P}$, respectively, which are both derived from the same template formula $\xi(t, t', \bowtie)$, saying that if a request for executing t' is seen after that for executing t, then the two subjects performing such tasks must be either different (when $\bowtie$ is $\neq$, i.e., in case of a SoD) or equal (when $\bowtie$ is $=$, i.e., in case of a BoD). Formulae φ, Σ and B, thanks to their temporal characterization, formalize the check (**C2**) in the introduction. A sequence of requests $\Pi|_{\mathsf{wid}}$ for a purpose p and an instance wid of the workflow $def(\mathsf{p})$ *satisfies* the purpose-aware policy $\mathcal{P}$ iff $\Pi|_{\mathsf{id}} \models \Phi_{\mathsf{p}}$. By abusing notation, we write $\mathcal{L}(\varphi)$ for all such sequences. Given a sequence σ of (previous) requests, a (new) request $r = (\mathsf{wid}, sub, tsk, do, \mathsf{p})$ is *granted* by the purpose-aware policy $\mathcal{P}$ iff wid is an instance of the workflow $def(\mathsf{p})$ and there exists a sequence σ' of requests such that σ, r, σ' is in $\mathcal{L}(\varphi)$; otherwise, it is *denied*.

To illustrate some of the notions introduced above, let us consider the first-order LTL formula that can be derived from the example in Section 2. As already observed, the formula φ associated to the purpose JobHunting is the conjunction of the formulae in the lower half of Figure 1. The conjunct in Λ for interview is

$$\mathsf{interview} \leftrightarrow \exists sub, do. \begin{pmatrix} req(\mathsf{wid}, sub, \mathsf{interview}, do, \mathsf{jobHunting}) \wedge \\ dcp(do, \mathsf{userProfile}, \mathsf{jobHunting}) \wedge \\ rcp(sub, \mathsf{read}, \mathsf{userProfile}) \end{pmatrix} .$$

The formula representing the SoD constraint between interview and findJobs is

$$\Box\forall sub, sub'.\begin{pmatrix}\forall do.req(wid, sub, \mathsf{interview}, do, \mathsf{jobHunting})\wedge\\ \Diamond\forall do.req(wid, sub', \mathsf{findJobs}, do, \mathsf{jobHunting})\end{pmatrix}\rightarrow sub \neq sub' \wedge$$
$$\Box\forall sub, sub'.\begin{pmatrix}\forall do.req(wid, sub, \mathsf{findJobs}, do, \mathsf{jobHunting})\wedge\\ \Diamond\forall do.req(wid, sub', \mathsf{interview}, do, \mathsf{jobHunting})\end{pmatrix}\rightarrow sub \neq sub' \ .$$

Notice that the second conjunct above can be dropped without loss of generality since, from $def(\mathsf{jobHunting})$, it is possible to derive that it is never the case that task findJobs is executed before task interview. From a complete specification of the purpose-aware policy $\mathcal{P}$ for the running example, it is not difficult to see that the request $r_0 = (\mathsf{wid}, \mathsf{bob}, \mathsf{interview}, \mathsf{sam}, \mathsf{jobHunting})$ is granted by applying the definition given above as follows. First, take σ to be the empty sequence as r_0 is the first request. Second, we can derive that task interview can be executed from the formula for interview above and the fact that $\mathcal{P}$ is such that $dcp(\mathsf{sam}, \mathsf{userProfile}, \mathsf{jobHunting})$, i.e. sam decided to release his user profile for the purpose of job hunting, and $rcp(\mathsf{bob}, \mathsf{read}, \mathsf{userProfile})$, i.e. bob has the right to read user profiles. Third, take $\sigma' = r_1, r_2, r_3, r_4, r_5$ for

$$r_1 := (\mathsf{wid}, \mathsf{sam}, \mathsf{optOut}, \mathsf{sam}, \mathsf{jobHunting}) \qquad r_2 := (\mathsf{wid}, \mathsf{bob}, \mathsf{getExp}, \mathsf{sam}, \mathsf{jobHunting})$$
$$r_3 := (\mathsf{wid}, \mathsf{adam}, \mathsf{findJobs}, \mathsf{sam}, \mathsf{jobHunting}) \qquad r_4 := (\mathsf{wid}, \mathsf{bob}, \mathsf{propJobs}, \mathsf{sam}, \mathsf{jobHunting})$$
$$r_5 := (\mathsf{wid}, \mathsf{sam}, \mathsf{choosJob}, \mathsf{sam}, \mathsf{jobHunting})$$

where r_1 corresponds to the fact that sam has opted out and only his past experiences can be released, r_2 to the fact that bob can retrive sam's past experiences (as said in Section 2), r_3 to the fact that the jobs for sam are found by adam, who is distinct from bob in order to satisfy the SoD constraint between interview and findJobs in Figure 1 (indeed, we assume that bob can execute getExp according to $\mathcal{P}$), r_4 to the fact that the list of found jobs is proposed to sam by bob in order to satisfy the BoD constraint between interview and propJobs in Figure 1, and r_5 to the fact that sam decides to pick a job from the proposed list.

We close the Section by remarking that this technique allows to discover inconsistencies as soon as they occur, i.e., at the earliest possible time. This is sometimes called *early detection* in the BPM literature, and it is a notable feature of temporal logics. To explain the concept, assume that, according to $\mathcal{P}$, *only* bob can perform the activities of jobHunting: when $r_0 = (\mathsf{wid}, \mathsf{bob}, \mathsf{interview}, \mathsf{sam}, \mathsf{jobHunting})$ (or actually any other request for purpose jobHunting) is presented to the system, we are able to understand that no execution can ever successfully complete the workflow, as sooner or later task findJobs must be executed by someone different from bob which however does not have the rights to do it. As a result, r_0 is denied and hence bob is not granted to access the data even if, by observing the current state, there is no evidence yet of any violation.

The decidability and complexity of answering requests is studied in the following section.

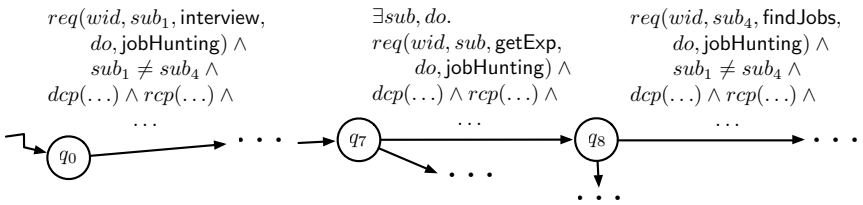


Fig. 3: An excerpt of the pre-automaton for formula $\Phi_{\mathsf{jobHunting}}$.

4 Policies verification

We now formalize, provide a solution and give complexity results to the following verification tasks:

Purpose achievement problem : given a purpose-aware policy $\mathcal{P}$ and a purpose p, is it possible to successfully execute workflow $def(\mathsf{p})$? That is to say: is it possible to assign tasks to subjects such that policy $\mathcal{P}$ is satisfied and the workflow successfully terminates?

Runtime policy enforcement : given a purpose-aware policy $\mathcal{P}$, the current workflow execution trace π, and a new request r_i, can r_i be granted or must be denied in order for the workflow to eventually terminate (check (**C2**) in the introduction) and such that the sequence of granted request always satisfies $\mathcal{P}$ (check (**C1**))?

Due to space constraints, proofs have been omitted. We refer the interested reader to [17] for the full-length version of this work.

4.1 Purpose achievement problem

Technically speaking, this problem amounts to check, given a purpose p in a purpose-aware policy $\mathcal{P}$, if Φ_{p} is satisfiable, i.e., if there exists a trace $\Pi|_{\mathsf{wid}}$ (where wid is a generic identifier for workflow $def(\mathsf{p})$) such that $\Pi|_{\mathsf{wid}} \models \Phi_{\mathsf{p}}$.

We adopt an automata-based approach to solve the problem, which consists in building the automaton for Φ_{p}, which we call A_{p}, and check if there exists a path to a final state. Since Φ_{p} is first-order, we exploit the modularity of our framework and the of symbolic techniques in [19] to build the automaton with a reasonable complexity. Figure 3 shows an excerpt of the automaton for $\Phi_{\mathsf{jobHunting}}$, where for the sake of readability we abbreviated formulas on edges. Indeed, on each edge, along with formula $req(wid, sub, \mathsf{t}, do, \mathsf{p})$, taking care of the order of activities, constraint $\bigwedge_{act,obj \in uses(act,\mathsf{t},obj)}(dcp(do, obj, \mathsf{p}) \wedge rcp(sub, act, obj))$ is also present, checking that dcp and rcp policies are met. We notice that variables for subjects involved in SoD or BoD constraints are *free*, i.e., not bounded by any quantifier. Indeed, as tasks $\mathsf{interview}$ and $\mathsf{jobHunting}$ must be executed by different subjects (SoD), variables for such subjects, namely sub_1 and sub_4, are free and must be different.

Theorem 1. *Given a purpose-aware policy $\mathcal{P}$, and a purpose* p *with* $def(\mathsf{p}) = \varphi$*, the construction of the* automata A_{p} *requires exponential space in the number of temporal operators of φ, sod and bod.*

Automaton A_{p} is a symbolic structure where we check whether there is a way to reach a final state—thus solving a workflow satisfiability problem—by trying to satisfy formulas on edges. Indeed, satisfy a formula (w.r.t. $\mathcal{P}$) precisely means assigning a task to a subject which is authorized by $\mathcal{P}$ to perform it. Consider, e.g., formula $\exists sub, do.req(wid, sub, \mathsf{findJobs}, do, \mathsf{jobHunting}) \bigwedge_{uses(act,\mathsf{findJobs},obj)} dcp(do, obj, \mathsf{p}) \wedge rcp(sub, act, obj)$ on edge from q_7 to q_8 in Figure 3: assuming that bob has the rights to perform getExp, the substitution bob/sub satisfy the formula according to $\mathcal{P}$, and so that edge can be use to build a path to a final state. Analogously, the assignment bob/sub_1 and adam/sub_4 satisfies formulas on edges from q_0 and q_8 respectively. When no such an assignment can be found, the workflow cannot be successfully completed given policy $\mathcal{P}$.

Theorem 2. *Given a finite purpose-aware policy $\mathcal{P}$ and a purpose* p *with* $def(\mathsf{p}) = \varphi$ *the* purpose achievement problem *can be solved in exponential time in the size of φ, sod and bod.*

4.2 Runtime policies verification

Given a sequence π of (previous) requests and a new request r, should we allow r or not?

Traditional LTL semantics presented in the previous Section is not adequate for evaluating requests at runtime, as it considers the trace Π seen so far to be *complete*. Instead, we want to evaluate the current request by considering that the execution could still continue and this evolving aspect has a significant impact on the evaluation: at each step, indeed, the outcome may have a degree of uncertainty due to the fact that future executions are yet unknown.

Consider, e.g., that so far request r_0 has been granted, where $r_0 := (\mathsf{wid}, \mathsf{bob}, \mathsf{interview}, \mathsf{sam}, \mathsf{jobHunting})$ is as in the previous Section. Assume that now request $r_1 := (\mathsf{wid}, \mathsf{sam}, \mathsf{optOut}, \mathsf{sam}, \mathsf{jobHunting})$ is presented and must be evaluated: we may be tempted to use the traditional LTL semantics, which returns $\pi : r_0, r_1 \not\models \Phi_{\mathsf{p}}$ because some constraints have not been satisfied (such as: "eventually findJobs must be executed") but this is not a good reason to deny request r_1, as the workflow execution will not stop after r_1 (and actually any other single request different from r_1 would not have satisfied Φ_{p} anyway).

A more complex analysis is hence required, which assesses the capability of a partial trace to satisfy or violate a formula φ *in the future* by analyzing whether it belongs to the set of *prefixes* of $\mathcal{L}(\varphi)$ and/or the set of prefixes of $\mathcal{L}(\neg\varphi)$. Roughly speaking, let $\pi : r_0, r_1$ be as before: we want to check if there exists a possible sequence of future requests $\pi' : r_2, r_3, \ldots r_n$ such that $\pi, \pi' \models \Phi_{\mathsf{p}}$ and, if this is the case, we grant request r_1. We can actually be more precise, and evaluate the current request in four different ways.

Given a (partial) trace π, a formula Φ_{p} and a new request r, we adopt the runtime semantics in [15] which is such that:

- $\pi, r \models [\Phi_p]_{\text{RV}} = \textit{temp\_true}$, when π, r *temporarily satisfies* Φ_p, i.e., π is currently compliant with Φ_p, but a possible system future prosecution may lead to falsify Φ_p;
- $\pi, r \models [\Phi_p]_{\text{RV}} = \textit{temp\_false}$, when that the current trace *temporarily falsify* Φ_p, i.e., π, r is not current compliant with Φ_p, but a possible system future prosecution may lead to satisfy Φ_p;
- $\pi, r \models [\Phi_p]_{\text{RV}} = \textit{true}$, when π, r *satisfies* Φ_p and it will always do, no matter how it proceeds;
- $\pi, r \models [\Phi_p]_{\text{RV}} = \textit{false}$, when π, r *falsifies* Φ_p and it will always do, no matter how it proceeds.

A new request r is *denied* if $\pi, r \models [\Phi_p]_{\text{RV}} = \textit{false}$, and *granted* otherwise.

Intuitively, every time a new request is presented, we check that: (*i*) from the current automaton state there exists an outgoing edge whose formula is satisfied by the current request (which corresponds to check (**C1**) in the introduction) and (*ii*) from the arrival state there exists a path to a final state, which is a WSP that exactly corresponds to check (**C2**). Such analyses are performed on an automaton which is different from the one presented in the previous Section, as it not only has to check prefixes of Φ_{p}, but also that of $\neg\Phi_{\mathsf{p}}$, in order to distinguish among the four cases above (see [15] for details). However, the automaton technique shown in [19] can equally be used.

Once the automaton has been computed, the current sequence of requests is analyzed. Notice that, differently from the offline verification, assignment of users to tasks are partially given by the current and previous requests, and hence we have to check if such partial assignments can be extended, according to policy $\mathcal{P}$, in order to reach a final state. Consider again $r_0 := (\mathsf{wid}, \mathsf{bob}, \mathsf{interview}, \mathsf{sam}, \mathsf{jobHunting})$ to be the already granted request and $r_1 := (\mathsf{wid}, \mathsf{sam}, \mathsf{optOut}, \mathsf{sam}, \mathsf{jobHunting})$ to be the current one. Request r_0 provides the assignment bob/sub_1 which forces us to solve the WSP with the additional constraint of $sub_1 = \mathsf{bob}$. Actually, $r_0, r_1 \models [\Phi_p]_{\text{RV}} = \textit{temp\_false}$, as $r_0, r_1 \not\models \Phi_{\mathsf{p}}$ but there exists a sequence of assignments of users to tasks that eventually satisfies it, which is sequence r_2, r_3, r_4, r_5 shown in the previous Section. We remark that a WSP instance must be performed each time a new request is presented. Indeed, the actual next request $\hat{r}_2$ (still unknown at the current time) is in general different from r_2, and hence sequence r_2, r_3, r_4, r_5 found at this step as a witness of a possible future execution is of no use as the system progresses. Notably, the fact that we discover inconsistencies at the earliest possible time (early detection) allows us to block workflow executions exactly when a possible successful path can still be followed. When this is not guaranteed, the online enforcement is inefficient as when the precise point of deviation from the right path is unknown: (*i*) possible several tasks are executed before realizing the inconsistency (hence several data are accessed thus breaking the security) and (*ii*) possibly it is too late to recover the execution.

Theorem 3. *Given a purpose-aware policy $\mathcal{P}$ and a purpose* p *with* $def(\mathsf{p}) = \varphi$, *the* runtime policy verification *requires, at each step, exponential time in the size of φ, sod and bod.*

5 Discussion and Related Work

We have presented a declarative framework to specify and enforce purpose-aware policies. In the literature, several proposals have attempted to characterize the notion of purpose in the context of security policies. Some of them (e.g., [10]) propose to manage and enforce purpose by self-declaration, i.e. subjects explicitly announce the purpose for accessing data. While this provides a first effort to embody purpose in access control policies, these approaches are not able to precent malicious subjects from claiming false purposes. Other works (e.g., [30]) propose to extend the Role Based Access Control model with mechanisms to automatically determine the purpose for which certain data are accessed based on the roles of subjects. The main drawback of these approaches is the fact that roles and purposes are not always aligned and members of the same role may serve different purposes in their actions. Other approaches (e.g., [11]) are based on extensions of (Attribute Based) Access Control models for handling personal data in web services or (e.g., [31]) on extensions of Usage Control models for the distributed enforcement of the purpose for data usage (see, e.g., [29]). The main problem of these approaches derives from the limited capability of the application initiating the handling of personal data to control it when this has been transferred to another (remote) application in the distributed system. Our framework avoids this problem by assuming a central (workflow-based) system acting as a Policy Enforcement Point (PEP) intercepting all requests of executing a particular action on certain data. Such an architecture for the PEP is reasonable in cloud-based environments (e.g., the Smart Campus platform briefly described in Section 2) in which services and applications are implemented via of Application Programming Interfaces (APIs) so that remote applications provide only the user interface while the control logic is executed on the cloud platform and can thus be under the control of the central PEP.

The common and more serious drawback of the approaches considered above is that they fail to recognize that the purpose of an action (or task) is determined by its position in a workflow, i.e. by its relationships with other, interrelated, actions. This observation has been done in more recent works—e.g., [33,28,21]—and is also the starting point of our framework in this paper. We share the effort of formalizing the notion of purpose as a pre-requisite of future actions. In future work, we would like to study how hierarchical workflows (i.e. workflows containing complex activities, specified in terms of lower level tasks) can be expressed in our framework so as to capture the specification of purposes as high-level activities as done in [21]. The main difference with previous works is our focus on run-time enforcement of purpose-aware policies while [33] and [28] on auditing. Furthermore, the formal framework adopted to develop these proposals are different from ours: [33] uses Markov Decision Processes, [28] a process calculus,

and [21] an *ad hoc* modal logic. These choices force the authors of [28,21] to design algorithms for policy enforcement or auditing from scratch. For instance, [21] gives a model checking algorithm on top of which a run-time monitor for purpose-based policies can be implemented, without studying its complexity. In contrast, we use a first-order temporal logic which comes with a wide range of techniques to solve logical problems (see, e.g., [15]) that can be reused (or adapted) to support the run-time enforcement of purpose-aware policies. For instance, we were able to derive the first (to the best of our knowledge) complexity result of answering authorization requests at run-time under a given purpose-aware policy: exponential time in the number of authorization constraints (Theorem 3). This complexity is somehow intrisic to the problem when assuming that the purpose of an action is determined by its position in a workflow, as we do in this paper. As a consequence, achieving a purpose amounts to the succesful execution of the associated workflow, which is called the Workflow Satisfiability Problem (WSP) in the literature [12] and is known to be NP-hard already in the presence of one SoD constraint [34]. Several works have proposed techniques to solve both the off-line [34,27] and the on-line [6,8] version of the WSP but none considers purpose-aware policies as we do here.

Indeed, we are not the first to use first-order LTL for the specification of security policies. For example, [26] shows how to express various types of SoD constraints in the Role Based Access Control model by using first-order LTL. However, the paper provides no method for the run-time monitoring of such constraints and do not discuss if and how the approach can be used in the context of workflow specifications. Instead, the work in [13] uses (a fragment of propositional) LTL to develop algorithms for checking the successfull termination of a workflow. Both works do not discuss purpose-aware policies. To the best of our knowledge, only the approach in [4] shares with ours the use of first-order LTL to specify and enforce utility and privacy in business workflows. The main difference is in the long-term goal: we want to give rigorous foundation to specification and verification techniques for purpose-based policies, while [4] is seen as a first step towards to the development of a general, clear, and comprehensive framework for reducing high-level utility and privacy requirements to specific operating guidelines that can be applied at individual steps in business workflows. It would be an interesting future work to see if and how the two approaches can be combined together in order to derive purpose-aware policies from high-level privacy requirements typically found in laws, directives, and regulations.

Our choice of using a first-order LTL formula (Section 3.1) as the semantics of purpose has two main advantages. First, it allows us to reuse well-known techniques to specify access control policies, what we call data- and rule-centric policies, by using (fragments of) first-order logic (see, e.g., [23,3]). Second, it allows us to reuse the techniques for the specification and enforcement of workflows put forward by the declarative approach to business process specification in [2,35,24]. However, these works focus on tasks and their execution constraints, disregarding the security and privacy aspects related to accessing the data manipulated by the tasks. A first proposal of adding the data dimension to this

approach is in [18], which has been from which we have borrowed the construction of the automata for off-line and on-line verification in this paper (Section 4). The choice of considering first-order LTL over finite instead of infinite traces goes back to [16] in which it is argued that this is the right choice for business process which are supposed to terminate as the workflows associated purposes, which can be achieved in this way. In general, monitoring first-order LTL formulae is undecidable [7] but, under the finite domain assumption, [19] shows decidability. Such an assumption is reasonable in our framework where subjects are usually employees of a company (e.g., the job hunting organization in Section 2) whose number is bounded. Our verification techniques are also related to mechanisms for enforcing security policies, such as [29,5,32]. However, these works mainly focus on access or usage control policies and are of limited or no use for purpose-based policies considered in this paper.

References

1. Directive 95/46/ec of the european parliament and of the council of 24 october, 1995, `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML`
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. CS - R&D 23(2), 99–113 (2009)
3. Arkoudas, K., Chadha, R., Chiang, C.J.: Sophisticated access control via SMT and logical frameworks. Proc. of ACM TISSEC 16(4), 17 (2014)
4. Barth, A., Datta, A., Mitchell, J.C., Sundaram, S.: Privacy and utility in business processes. In: Proc. of 20th IEEE Computer Security Foundations Symposium (July 2007)
5. Basin, D., Klaedtke, F., Müller, S.: Monitoring security policies with metric first-order temporal logic. In: Proc. of ACM SACMAT. pp. 23–34. SACMAT, ACM, New York, NY, USA (2010)
6. Basin, D., Burri, S.J., Karjoth, G.: Dynamic enforcement of abstract separation of duty constraints. ACM TISSeC 15(3), 13:1–13:30 (Nov 2012)
7. Bauer, A., Küster, J., Vegliach, G.: From propositional to first-order monitoring. In: Runtime Verification. pp. 59–75 (2013)
8. Bertolissi, C., dos Santos, D.R., Ranise, S.: Automated Synthesis of Run-time Monitors to Enforce Authorization Policies in Business Processes. In: Asia CCS. ACM (2015)
9. Byun, J.W., Bertino, E., Li, N.: Purpose based access control of complex data for privacy protection. In: Proc. of the ACM SACMAT. pp. 102–110. ACM (2005)
10. Byun, J., Li, N.: Purpose based access control for privacy protection in relational database systems. VLDB J. 17(4), 603–619 (2008)
11. C.A.Ardagna, M.Cremonini, S.De Capitani di Vimercati, P.Samarati: A privacy-aware access control system. Journal of Computer Security (JCS) 16(4), 369–392 (September 2008)
12. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: Proc. of ACM SACMAT. pp. 38–47. ACM (2005)
13. Crampton, J., Huth, M., Kuo, J.P.: Authorized workflow schemas: deciding realizability through LTL(F) model checking. Int. J. on Software Tools for Technology Transfer (STTT) 16(1), 31–48 (2014)

14. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Samarati, P.: Access control policies and languages. IJCSE 3(2), 94–102 (2007)
15. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on LTL and LDL for finite traces. In: Proc. of BPM. pp. 1–17 (2014)
16. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: Insensitivity to infiniteness. In: Proc. of AAAI Conf./ on AI. pp. 1027–1033 (2014)
17. De Masellis, R., Ghidini, C., Ranise, S.: A declarative framework for specifying and enforcing purpose-aware policies (2015), http://arxiv.org/abs/1507.08153
18. De Masellis, R., Maggi, F.M., Montali, M.: Monitoring data-aware business constraints with finite state automata. In: Proc. of ICSSP. pp. 134–143 (2014)
19. De Masellis, R., Su, J.: Runtime enforcement of first-order LTL properties on data-aware business processes. In: Proc. of ICSOC. pp. 54–68 (2013)
20. Jafari, M., Safavi-Naini, R., Sheppard, N.P.: Enforcing purpose of use via workflows. In: Proc. of WPES. pp. 113–116 (2009)
21. Jafari, M., Safavi-Naini, R., Fong, P.W.L., Barker, K.: A framework for expressing and enforcing purpose-based privacy policies. ACM Trans. Inf. Syst. Secur. 17(1), 3:1–3:31 (Aug 2014)
22. Kröger, F., Merz, S.: Temporal Logic and State Systems. Texts in Theoretical Computer Science. An EATCS Series, Springer (2008)
23. Li, N., Mitchell, J.C.: Datalog with constraints: a foundation for trust management languages. In: PADL'03. pp. 58–73 (2003)
24. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: BPM. pp. 132–147 (2011)
25. Masoumzadeh, A., Joshi, J.B.: PuRBAC: Purpose-Aware Role-Based Access Control. In: On the Move to Meaningful Internet Systems: OTM 2008, LNCS, vol. 5332, pp. 1104–1121. Springer Berlin Heidelberg (2008)
26. Mossakowski, T., Drouineaud, M., Sohr, K.: A temporal-logic extension of role-based access control covering dynamic separation of duties. In: Proc. of TIME-ICTL. pp. 83–90 (2003)
27. P. Yang, X. Xie, I.R., Lu, S.: Satisfiability analysis of workflows with control-flow patterns and authorization constraints. IEEE TSC 99 (2013)
28. Petkovic, M., Prandi, D., Zannone, N.: Purpose control: Did you process the data for the intended purpose? In: Secure Data Management. pp. 145–168 (2011)
29. Pretschner, A., Hilty, M., Basin, D.: Distributed usage control. Comm. ACM 49, 39–44 (2006)
30. Qun, N., Elisa, B., Jorge, L., Carolyn, B., Karat, C.M., Alberto, T.: Privacy-aware Role-Based Access Control. TISSeC 13, 1–31 (July 2010)
31. Rath, A.T., Colin, J.N.: Modeling and Expressing Purpose Validation Policy for Privacy-aware Usage Control in Distributed Environment. In: Proc. of ICUIMC. pp. 14:1–14:8. ACM (2014)
32. Schneider, F.B.: Enforceable security policies. TISSeC 3, 30–50 (2000)
33. Tschantz, M.C., Datta, A., Wing, J.M.: Formalizing and enforcing purpose restrictions in privacy policies. In: IEEE Symposium on Security and Privacy. pp. 176–190 (2012)
34. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. TISSeC 13, 40:1–40:35 (December 2010)
35. Westergaard, M., Maggi, F.M.: Declare: A tool suite for declarative workflow modeling and enactment. In: Proc. of BPM (2011)
36. Westin, A.: Privacy and Freedom. Atheneum, New York, USA (1968)

De Masellis R., Ghidini C., Ranise S., A Declarative Framework for Specifying and Enforcing Purpose-Aware Policies. *In proc. of* Security and Trust Management, Volume 9331 of the series Lecture Notes in Computer Science, pp. 55-71 10.1007/978-3-319-24858-5_4.


```
@inproceedings{DeMasellisGR15,
  author    = {Riccardo {De Masellis} and
               Chiara Ghidini and
               Silvio Ranise},
  title     = {A Declarative Framework for Specifying and Enforcing Purpose-Aware
               Policies},
  booktitle = {Security and Trust Management - 11th International Workshop, {STM}
               2015, Vienna, Austria, September 21-22, 2015, Proceedings},
  pages     = {55--71},
  year      = {2015},
  crossref  = {DBLP:conf/stm/2015},
  url       = {http://dx.doi.org/10.1007/978-3-319-24858-5_4},
  doi       = {10.1007/978-3-319-24858-5_4}
}
```