Journal on Data Semantics (2019) 8:203–218
https://doi.org/10.1007/s13740-019-00106-2

ORIGINAL ARTICLE

Dynamic Access Control to Semantics-Aware Streamed Process Logs

Marcello Leida[1] · Paolo Ceravolo[2] · Ernesto Damiani[3] · Rasool Asal[4] · Maurizio Colombo[4]

Received: 13 August 2018 / Revised: 19 May 2019 / Accepted: 13 July 2019 / Published online: 24 July 2019

Abstract
Business process logs are composed of event records generated, collected and analyzed at different locations, asynchronously and under the responsibility of different authorities. Their analysis is often delegated to auditors who have a mandate for monitoring processes and computing metrics but do not always have the rights to access the individual events used to compute them. A major challenge of this scenario is reconciling the requirements of privacy and access control with the need to continuously monitor and assess the business process. In this paper, we present a model, a language and a software toolkit for controlling access to process data where logs are made available as streams of RDF triples referring to some company-specific business ontology. Our approach is based on the novel idea of *dynamic enforcement*: we incrementally build dynamic filters for each process instance, based on the applicable access control policy and on the current prefix of the event stream. The implementation and performance validation of our solution is also presented.

1 Introduction

Business process analysis is the activity of reviewing existing business practices and changing them so that they fit a new, improved process model. By conducting process analysis, companies expect to streamline their internal processes and become more effective in their business operation [1]. Traditionally, research in this field spans two major areas: *Process Monitoring* (PMon), also called Business Activity Monitoring, which collects and processes events while they occur, and *Process Mining* (PMin), which typically focuses on the offline analysis of process logs. Both areas describe business processes using abstract concepts such as tasks, subprocesses, start and end times, properties, relationships and work-flows. Such basic entities are usually organized into a schema, which can be extended to define additional domain-specific concepts (Fig. 1). The difference between monitoring and mining mainly lies in the collection of observable process events. In Pmon, remote *process probes* broadcast events over the network to be analyzed by monitors; in Pmin, event data are often imported to some static *data lake*. Both PMon and PMin have been extensively studied in the literature [2–4], but the problem of performing PMon while satisfying privacy and access control requirements has received less attention. In order to clarify the problem, we need first to introduce some basic terminology of computer security. We follow [5] distinguishing between Access Control (AC) *policies*, *mechanisms*, and *models*. AC policies are high-level requirements that express who or what should be granted or denied access to what resources. AC mechanisms implement such policies in an IT system. For example, AC policies are often expressed as *rules* of the form *Requestor:Resource:Condition:Action* that the AC mechanism (usually implemented as a *Policy Evaluation Point* (PEP)) can be used to determine whether an access request to a resource should be granted or denied. Finally, AC models are formal representations of the internal operation of AC mechanisms, used to prove their correctness. In our case, the resources to be accessed are *business process streams* including (representations of) activities, tasks, and their attributes. Intuition suggests that a mechanism for enforcing AC policies on such resources could be a *filter*, sanitizing the process event and leaving only the information the requester is entitled to see. Indeed, filters have been proposed since long as a practical way to exclude events from, or otherwise, manipulate business process logs [6]; but the study of automatic generation of log filters as an AC mechanism is

✉ Paolo Ceravolo
paolo.ceravolo@unimi.it

Marcello Leida
mleida@stratio.com

[1] Stratio, Madrid, Spain

[2] Università Degli Studi di Milano, Milan, Italy

[3] Center on Cyber-Physical Systems, Khalifa University, Abu Dhabi, UAE

[4] EBTIC, Khalifa University, Abu Dhabi, UAE

still in its infancy. The problem got even more difficult with the advent of multi-enterprise business coalitions: besides having a huge size, today's process streams are generated, collected and analyzed asynchronously, at different locations and under the control of different organizations. In such scenarios, writing the filter corresponding to a given AC policy manually is awkward and error-prone, and solutions targeting Big Data source are needed [7,8].

In this paper, we present a model and the corresponding extension to the standard XACML policy language [9] for controlling access to process streams. The stream is represented as a flow of RDF [10] triples complying with some company-specific activity ontology. We discuss representational issues in Sect. 2.2. In Sect. 2.4, we briefly discuss the related but distinct problem of mapping finer-grained events (e.g., sensor readings) to activities, whose solution is outside the scope of this paper. Encoding process events as RDF triples is a solid way to achieve a semantic extension of traditional process logs, specifying the semantics of individual model elements [11] and ensuring that process log items can be universally understood by all applications that have the relevant domain knowledge. This explicit encoding of events can be exploited by our AC mechanism in enforcing XACML access policies on the stream by computing and dynamically updating a process log filter. Applying the filter generates a fully sanitized flow of events, complying with the AC policy. Besides enforcing the AC policy, our technique "weaves the log" in order to preserve the semantic integrity of the sanitized process flow (e.g., w.r.t. timing). We claim that our approach is: (i) fully compatible with current access control models like classic Role-Based Access Control (RBAC) [12] and (ii) suitable for deployment in a Big Data environment where processes are continuously monitored and users can register continuous queries in order to be notified by new events matching their query. Last but not least, our approach supports *policy auditing*: being written in a standard language which is suitable for auditing [13], our extended XACML policies can be forwarded to an independent auditor to verify their compliance with auditing rules and other regulations without disclosing the process data. Indeed, our technique is particularly suitable for use cases where auditing is of paramount importance, such as telecommunication companies and cloud providers[1]. The remainder of the paper is organized as follows. Section 2 presents the process data model used in our system. In Sect. 3, we discuss and compare the related work to our approach. Section 4 summarizes our main contributions. In Sect. 5, we present our extension to XACML, while Sect. 6 describes our enforcement mechanism in detail. Finally, Sect. 7 validates our approach by providing a proof of correctness (Sect. 7.1) and an experimental performance analysis (Sect. 7.2). Section 8 concludes the paper and discusses future research directions.

[1] Such organizations are subject to data protection regulations from a variety of organizations ranging from the European Union (Data Retention Directive) to national and international law enforcement agencies. Regulations explicitly prescribe the deployment of AC mechanisms that prevent unauthorized access to clients' critical information [14,15]

2 Representing Process Log Data

The collection of event logs is a pre-condition to business process analysis. Event logs originate from the messages generated by an executing procedure or by a probe installed to capture events. The data structure adopted for event logs can be organized in a variety of ways [16].

In this section, we discuss semantics-aware log representation; then, we introduce our extensible RDFS schema for representing process logs, suitable for PMin and PMon applications.

2.1 Event-Logs Standards

The Process Mining community has encouraged the adoption of the eXtensible Event Stream (XES) [17], that is the format processed by most BPMin algorithms and libraries. This standard correlates events generated during a same business process execution, by the notion of case, and sequences them based on their associated time-stamps. It also includes the notions of the *originator* of an event and the *resources* exploited during its execution.

A known limit of this standard is to impose a flat multi-dimensional data structure[2]. A general limitation of flat log formats is their redundancy, as entities (e.g., involved resources) recurring in multiple events are recorded in all corresponding event tuples. Relational [19] or graph-based [20] databases for storing event log data have been proposed as a solution to address this limitation. Besides handling process data, using a database supports computing *views* on them. Query patterns allow to extract log files collecting variants of a business process that comply with specific perspectives [21–23]. For example, they can isolate processes executed by a specific department, within a specific time-frame, or failing to achieve specific performance levels. As a result, these different variants can be comparatively assessed getting relevant insight into the enablers of specific behaviors [24,25]. Partially motivated by the same reasons, the authors of [26] proposed an ontology-based data access procedure to support the extraction of event logs from a legacy system, using a domain-specific conceptual model. All these techniques perform manipulation after collection and are

[2] Object-oriented representation that can capture one-to-many and many-to-many relationships among events have been proposed [18] but not yet deployed in practice.

therefore more suitable for Pmin than for Pmon applications. Also, they imply *data normalization*, reducing redundancy at the expense of requiring expensive joins at extraction time [27].

2.2 Semantics-Aware Process Log Representations

Besides multi-dimensional analysis, the conceptual model encoding process logs can support the definition of additional transformation functions. Representation languages equipped with a formal semantics support (i) the organization of data according to different levels of abstraction, (ii) the composition of knowledge from multiple sources, or (iii) the identification of conflicting facts that identify the violation of expected behavior.

Identifying the appropriate abstraction level is a key prerequisite for event-logs analysis [28]. Moreover, when integrating logs from multiple systems, even in presence of a common vocabulary, multiple abstraction levels may apply at the source level, imposing specific reconciliation strategies [29]. Formal semantics can extend the capability of executing conformance checking procedures by assessing an event log in integration with external knowledge [30].

Early proposals of semantics-aware PMin focused on extending event logs with annotations linked to a conceptual model [31,32], if necessary supporting transformation stages [33,34].

In [20], an RDF serialization of graphs is adopted to model and analyze process logs. In this paper, we rely on a model we already used for semantic lifting [35], model reconciliation [29] and knowledge discovery [25].

2.3 The EBTIC-BPM Process Vocabulary

Today's PMon tools have to be flexible enough to handle process information that was unknown at the time of process model definition [20]. Our approach relies on two standards: the *Resource Description Framework* (RDF) [10] and *RDF Schema* (RDFS) [36]. We shall not try to provide any RDF/RDFS tutorial in this section: the interested reader can refer to [10,36]. Historically, RDF has been used as metadata, i.e., to annotate logs (and, in general, data streams) by adding information about data provenance, collection, etc. [37]. Here, we use RDFS to provide a vocabulary defining the basic entities of the business process domain. This simple representation allows a high degree of flexibility and at the same time includes all entities (concepts, relations, and attributes) that are needed by process management applications.

The RDF schema shown in Fig. 1 is actually just an envelope which includes only two concepts: `ebtic-bpm:Process`, which represents the business process, and `ebtic-bpm:Task`, which represents the activities that

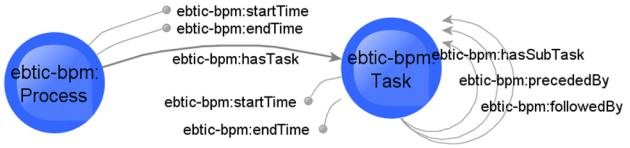

Fig. 1 The conceptual model representing the basic business process representation

compose the process. Each concept has a basic set of attributes: `ebtic-bpm:startTime` and `ebticbpm:endTime`, which represent the beginning and termination times. The relation between these two basic concepts defines the simplest way to represent a business process. The `ebtic-bpm:hasTask` relation links `ebtic-bpm:Process` to `ebtic-bpm:Task` and represents the set of tasks belonging to a process. Three more relations apply to the `ebtic-bpm:Task` concept: `ebtic-bpm:followedBy`, `ebticbpm:precededBy` and `ebtic-bpm:hasSubTask`, which respectively indicate which tasks precede and follow a given one and which tasks are subtasks of a given one. These relations allow layering our model with specific ontologies describing the business activities addressed by an organization or additional domain knowledge originated from normative or contractual regulations [20].

2.4 From Sensor Events to Semantics-Aware Log Entries

Most tools recording events adopt a fine-grained representation model suitable to the events generated during process enactment rather than to business process activities. Activities' and events' representations may have very different abstraction levels, leading to misinterpretation of process analysis results [38]. Consider, for example, the event logs of a sensor network that collects sensor records. The abstract *start* and *end* event delimiting execution of a process activity may correspond to variable sequences of finer-grained events [39]. Our semantic-aware log format is indeed the target of a semantic lifting procedure [35,40] which is outside the scope of this paper. Recently, machine learning procedures have been adopted to automatically learn a mapping linking aggregated events with activities at the business process level [41,42].

2.5 Software Architecture

Zeus, our process analysis platform [43], is a PMon system based on the data model introduced in the previous section. As illustrated in Fig. 2, Zeus includes *monitors* that use domain-specific extensions of the EBTIC-BPM basic vocabulary to tag process events and submits the corresponding

RDF triples to the Zeus triple store. In other words, when a monitor detects an activity, it generates the RDF triples representing it and sends them to the message queue. A *message listener* reads the message queue and inserts the triples into the Zeus triple store [43]. The Zeus triple store exposes a SPARQL query service allowing external applications to query it. SPARQL is a standard query language for RDF graphs based on conjunctive queries on triple patterns, which identify paths in the RDF graph [44]. SPARQL (or its ad hoc extensions like FPSPARQL [45]) is used for analyzing event logs of process-related systems, enabling analysts to group-related events in the logs or find paths among events. A major feature of the Zeus triple store that makes it suitable for handling streams is that it allows to *register* continuous SPARQL queries; such queries are automatically invoked whenever new triples get inserted in the Zeus triple store, and their results are automatically returned to the client that registered the query.

3 PMon Security and Access Control

In this section, we discuss related work in two research areas relevant to our approach: business process security and RDF access control.

3.1 Access Control to Business Process Mining and Monitoring

Access Control (AC) constrains "what a user can do directly, as well as what programs executing on behalf of the users are allowed to do" [5]. In other words, AC decides which subject is authorized to perform certain operations on a given object and which is not. Along the years, many AC models like access *control lists* (ACL), *capability lists* and *role-based access control* (RBAC) have been proposed. In particular, RBAC models proposed in the 1990s [12,46] are still widely used today. AC models for business processes have been mostly aimed at process execution, i.e., at controlling who can perform process activities. Some early works in this area discussed authorizations to perform Web interactions [47] or service invocations [48]. Later, Russell and van der Aalst [49] have surveyed various possibilities for assigning process activities to users. Wainer et al. [50] proposed the W-RBAC access control model, featuring an expressive logic-based language for selecting users that are authorized to perform certain process tasks. Over time, additional approaches for dealing with other security issues related to process management were introduced. In the context of the ADEPT project, Barbara Weber et al. [51] dealt with controlling access to process schemata, extending RBAC to support design-time actions like schema changes. In turn, the CEOSIS project proposed a way to adapt this type of AC rules when the underlying organizational model is changed [52]. A method for verification of security policies for business processes is presented in [53], where the authors present a system for automatic verification of secure models in order to prevent implicit information leaks about confidential parts of the process flow.

These lines of research are related to ours, inasmuch as the resources to be accessed are the representations of the activities in the process log rather than the services performing such activities or the entities in the process schema.

Event flows generated by process monitors have been traditionally filtered using static ad hoc views on event stores [46]. This technique works as follows: for each process schema, a number of static views is manually defined, each of them including the information accessible for users with a particular role. This way, rights over the process are defined implicitly by each view. The view-based approach relies on existing view definition languages (a survey on RDFS view definition can be found in [54]) to express access rights over process data. However, static views are costly to maintain. If the underlying process schema is modified, views affected by this change have to be identified, possibly among a large number of existing views. Also, when a user has multiple roles, more than one view applies, and view combination may lead to conflicts with unexpected results.

An AC model aimed at process logs has been described in the framework of the Proviado project for process data visualization [55,56]. The Proviado access control model expresses access rights on specific process aspects called objects. However, while the Proviado AC language has been used to express access rights in some process analysis case studies, no enforcement on streams has been described. Another related line of research is the one that led to Staab and Ringelstein's PAPEL (Provenance Aware Policy definition and Execution Language) [57].

PAPEL focuses on policies expressing temporal conditions based on processing histories and uses a sticky logging mechanism to provide the needed provenance. A point of contact with our work is that conditions are checked on graphs, but in the case of PAPEL, the graph structure represents the process log's temporal structure. While PAPEL syntax is very different from standard XACML, the language scope is fully complementary to ours (Sect. 7.2).

Finally, it is important to remark that AC is not the only motivation for filtering process models and log data. Selective visualization of process models has been proposed since long to enable process analysis at different levels of abstraction (for example point-of-view analysis [23]). For instance, Polyvanyy et al. [58] describe structural aggregations of the process logic in order to realize different levels of abstraction. Greco et al. [59] proposed an approach to produce hierarchical process views of the process that capture process behavior at a different level of detail. However, none of these tech-

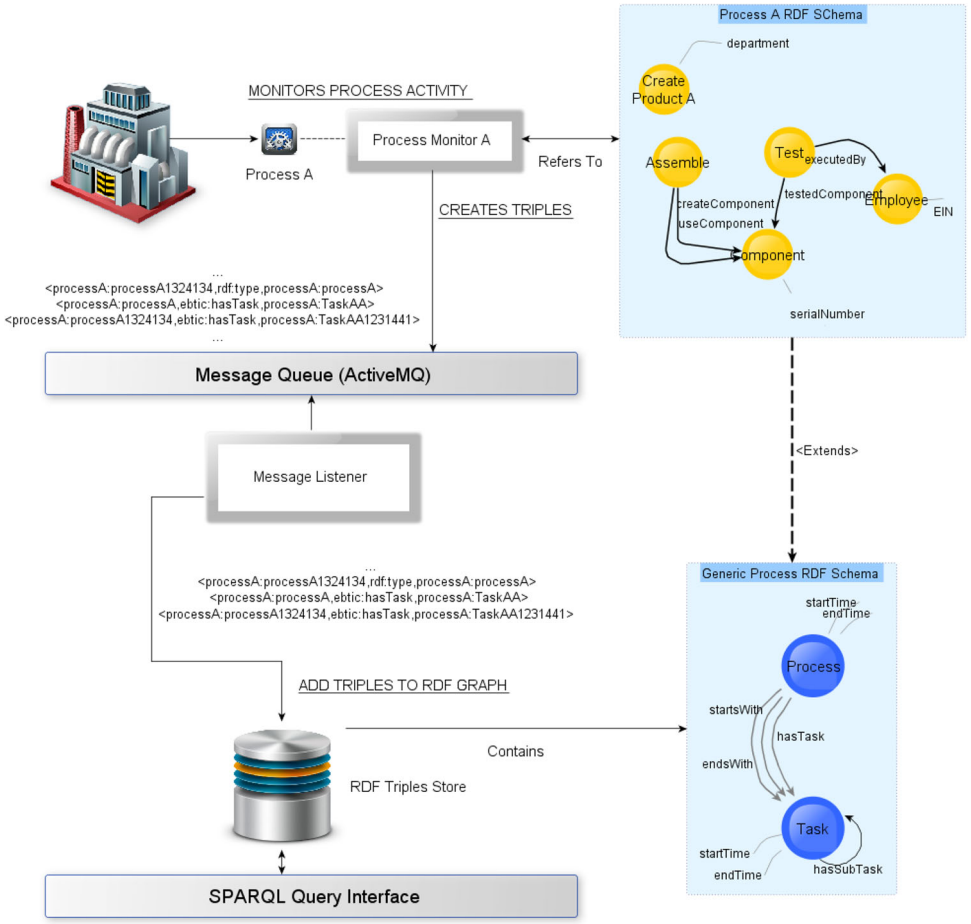

Fig. 2 The conceptual architecture of Zeus, our RDF-based PMon system

niques handles selective visualization of data generated by process monitoring.

3.2 RDF Access Control

The protection of information encoded in RDF graphs represents a relatively novel direction of research. In [60], the authors present their Semantic Based Access Control (SBAC), a model based on OWL ontologies[3]. SBAC is only partly relevant to our research, as it is aimed at controlling access to elements of RDF Schemata (Concepts, Individuals, Attributes and Relations). Other researchers used RDF annotations to complement AC policies. A seminal paper on this topic is [62], where the authors show that standard AC policy languages like XACML can be extended via RDF annotations and describe a reference architecture for enforcing annotated AC policies. This line of research was developed in more recent works. For instance, [63] presents ROWLBAC, a role-based access control architecture defined using the OWL ontology language. This way, existing OWL reasoners can be used for policy evaluation and policies defined with this language are exportable and verifiable.

Access control mechanisms targeting RDF triple stores are more relevant to this paper. In the abundant literature on this subject, we identify two main trends: the former relies on access control lists pointing to specific elements of RDF graphs (concepts, relations, attributes, and triples), while the latter uses the AC policy to define a set of "safe views" on the triple store's RDFS schema. An example of the former approach is the one proposed by Chen et al. [64], where the standard XACML policy language is used to control access to elements of an ontology at A-Box level. In their implementation, a security proxy rewrites incoming SPARQL queries in order to make them policy-compliant. However, their rewriting uses a static technique where pre-set filters are appended to individual user queries.

Both approaches are static in nature, but the former is complicated by the fact that simple facts (i.e., individual RDF triples) cannot be omitted or deleted by the AC enforcement mechanism without risking to compromise the integrity of the information passed on to the requester. The latter approach makes it easier to grant access to sub-graphs (or transformed

[3] OWL stands for Ontology Web Language and is an extension of RDF based on Description Logics [61].

graphs) of the original RDF graph that preserve information integrity.

A straightforward development of the latter line of research is run-time composition of safe views with user queries, thus enforcing access control policies via query rewriting. Our approach falls into this category: providing a query rewriting mechanism which allows to maintain the integrity of the graph. However, with respect to available rewriting approaches, our solution is applied on a stream of RDF triples instead of a static RDF graph. Most query rewriting approaches assume that AC policies can be expressed as sets of static views; so when a requester wants to query the data, her safe view can be retrieved and composed with her query. Generally speaking, given a set of views $V = \{V_1, V_2, \ldots, V_n\}$ over a RDF graph G, and a SPARQL query Q over the vocabulary of the views, query rewriting computes a SPARQL query Q_0 over G such that $Q_0(G) = Q(V(G))$. Rewriting must satisfy two properties: *soundness* and *completeness* [65]. Namely:

- a rewriting is *sound* $\iff Q_0(G) \subseteq Q(V(G))$;
- a rewriting is *complete* $\iff Q(V(G)) \subseteq Q_0(G)$.

Checking these properties may be easy in a static RDF triple store but is unfeasible for stream scenarios where triples are generated dynamically and may not be all present at the time of enforcement. Also, SPARQL rewriting algorithms used to enforce access control policies [65] tend to show high complexity either in time or in the number of views involved. The reason is somewhat intuitive: whenever a requester submits a query, it is necessary to identify which safe views need to be taken into account to rewrite it. The work coming closer to our approach in terms of rewriting policies is probably the SPARQL Query Rewriting (SQR) proposal [65]. This proposal relies on the following idea: if a variable mapping exists between a pattern in the result description and one of the triple patterns in the graph pattern of a query Q, then we can conclude that view V_j is needed to rewrite Q. SQR performs the query rewriting in two steps. In the first step, the algorithm determines, for each triple pattern in the user query, the set of views whose variables appear in the pattern. In the second step, the algorithm rewrites the original query as a union of conjunctive queries to the triple store. Rewritten queries use the original schema graph but select only "authorized" nodes. A similar approach is mentioned in a patent [66] that describes a method to rewrite SPARQL queries based on static views.

However, the approaches available so far are static approaches which are applied on a static graph, while our approach is applied on a stream of incoming RDF triples, which is dynamic by nature. For this reason, it is hard to experimentally compare the described related works with our approach.

4 Research Contributions

Our approach provides several new contributions with respect to the state of the art[4].

A first contribution is *dynamic filter generation*: upon receiving an access request, our technique automatically generates a filter and keeps updating it on the basis of the events generated by process execution. Our filters are continuously applied to the process event flow to keep it compliant with the AC policy. This will ensure that queries performed by the users (e.g., to compute performance metrics over the process) are executed on a "sanitized" flow of events. Our approach is suitable for deployment in a real-time environment where processes are continuously monitored and users can register *continuous queries* in order to be notified when new events matching their query arrive. As we shall see in Sect. 6, our technique is less expensive in terms of computation than approaches based on static views, which use the entire AC policy to build the views associated with a user.

A second contribution is preserving the log's *semantic integrity*. We use XACML *obligation* mechanism to keep the sanitized flow sound with respect to the correctness of metrics computed on it, like total elapsed time. In standard XACML, an obligation is a directive on an action that must be carried out after access is approved. We use obligations as a way to require filter updates on demand, i.e., when new triples arrive.

A third contribution is our approach's *auditability* due to standard syntax of AC policies. Most existing solutions provide an abstract AC model to define access rights in an abstract format, but such access rights must then be manually mapped to static views over process data; this mapping process is error-prone and difficult to debug independently. Our approach allows policy authors to write their policies in standard XACML. Such policies can be forwarded to an independent evaluator to verify that they are compliant with auditing rules and data protection regulations [13]. As we shall see, our extensions to XACML are fully modular and comply with the prescribed language extension points[5], so they do not affect policy audit.

5 The Policy Language

In order to support the Administrator in writing AC policies in a machine-readable format, we defined an extension of the standard XACML policy language [9]. Our extension is not focused on process logs; rather, it provides a generic obligation language for RDF streams. The format of our obligations is composed of five main elements, described in detail in Table 1.

[4] Our AC engine is covered by US Patent 20150172320.

[5] http://mvpos.sourceforge.net/xacml.htm.

Following the sample process log introduced in Sect. 2 we will now define a set of restrictions on the process flow using our language.

For the sake of simplicity and as further motivated in the evaluation section, we define a policy that is somewhat simplified with respect to real-life ones but will ease understanding of the results of our evaluation. Let us assume that a process analyst is called in for performing analysis on our sample process on loan approval. Due to data protection regulations[6] all details of loan applications should only be accessible to employees involved in their assessment, and not to the process analyst. In our sample loan application process, all the activities related to the submission, acceptance, and approval of a loan can be performed by different types of agents (human or automatic systems). To ensure compliance with regulations, the process analyst is not allowed to see the type of agent that performed these actions in case the amount of the loan is equal to (or greater than) a certain value. We define a policy associated with the role of process analyst. The policy replaces the type of agent with a generic one where the conditions described previously are verified. Once the analyst authenticates and logs into the system, a standard XACML policy is used to verify that her credentials enable her to access the log after sanitization. Definition 2 provides a version of this policy in simplified English. Once access is granted, the set of obligations shown in Definition 2 is enforced by our system against the stream of triples of the process log, as illustrated in Excerpt 1.

```
Task{deny}
Match{
    ?task org-resource ?res.
    ?task lifecycle-transition ?lt
}
Condition{
    ?process ebtic-bpm:hasTask ?task.
    ?task rdf:type ?tasktype.
    ?tasktype rdfs:subClassOf ebtic-bpm:task
}
Alternative{
    Find{
       ?task rdf:type ?tasktype.
       ?tasktype rdfs:subClassOf ebtic-bpm:task }
    Replace{
       ?task rdf:type ebtic-bpm:task }
}
DecisionPoint{
    ?task ebtic-bpm:endTime ?endTime}          (1)
```

[6] In the USA, the Gramm-Leach-Bliley Financial Services Modernization Act (GLB Act), Pub. L. No. 106-102, 113 Stat. 1338 (Nov. 12, 1999).

The incoming triples are verified against the *Match* block; in our sample policy we look for processes having a task performed by an agent of type "human." In case the triple matches the match block then the *conditions* need to be tested in order to verify if it can be added to the user's sanitized RDF graph. As seen in Eq. 2, our policy expresses three disjunctive conditions: one checking that the loan process is for an amount of 100000 and that the task performed by a human agent is of type submission. A second condition is checking that the loan process is still for an amount of 100000 and that the task performed by a human agent is of type accepted and a final alternative condition is checking in the process is for an amount of 200000 and the task performed by the human agent is of type approved. In case one of these conditions is verified, the triples in the *find* block are removed from the graph and replaced with the ones in the *replace* block. The conditions are verified on the same process until either they are satisfied or the *decision point* is reached. In our sample policy the decision point if the triple indicating the ending time of the process. This way, even if the filter updater seems to process a triple at the time, the decision point provides a mechanism to handle the triples of a process as related.

Policy#1:

In Process Loan **it is denied** Analyst **with** *has_access_to Task* Submitted **with** *amount* **more than** *100000*,
or *Task* Approved **with** *amount* **more than** *200000*,
or *Task* Accepted **with** *amount* **more than** *100000*. (2)

6 Overview of the Approach

In this section, we provide an overview of our AC mechanism (Fig. 3). The goal of the approach is to generate a logical process log for each user (or client application), showing only the information the user is entitled to see according to the AC policy and preserving the semantic integrity of the resulting RDF stream.

For the sake of simplicity, here we model only two stakeholders but the same mechanism can be executed with more actors. In the example we use, the *Requester* (i.e., the analyst/auditor accessing the process) assigns access rights to the *Administrator* (i.e., the process owner or a delegate) by encoding them in an XACML policy. The policy is enforced by a *Policy Enforcement Point* (PEP) operating on the incoming stream of process events. Let us now describe the entities mentioned in Fig. 3 in more detail.

6.1 The Filter Updater

The Filter Updater takes care of creating and incrementally updating the filter to be used for computing the Requestor's view of the process data. At any given time, each Requestor u

Table 1 Policy elements

Policy element	Description
Task	This element can assume two values: `allow` or `deny`. It provides information on the effect of the policy: if it allows the resources to be accessed by the Requestor or vice versa
Match	This element is the resource on which the policy will be applied; it represents a set of RDF triples that the Requestor is allowed to access or not (according to the value of *Task*)
Condition	This element contains a set of graph patterns which are translated for their evaluation into SPARQL ASK queries[a] on the process flow that needs to be satisfied in order to apply the policy. Conditions can be connected with logical operators (*And*, *Or* and *Not*)
Alternative	This element is used to sanitize the RDF graph when the Requestor is not authorized to access the triples in the Match Block. The Alternative element has two children: *Find* and *Replace* containing a graph pattern each. The Find element tells the Filter Updater which part of the original RDF data needs to be replaced with the Requestor-specific Replace one.
DecisionPoint	This element contains a graph pattern that acts as a terminator: when it is found in the flow, the PEP stops the evaluation of Conditions[b]

[a] ASK queries return results of type `Boolean` and are best suited to identify the existence of a graph pattern in the RDF graph

[b] The DecisionPoint element is important because the policies are applied to streams of triples. When a triple matching the Match block is detected, Condition may not yet be satisfied, but it can become so due to other triples that come next in the flow. So, the PEP goes on checking until DecisionPoint is reached

corresponds to a filter $V_u =< V_{allow}, V_{deny}, V_{alternative} >$, where V_{allow}, V_{deny} and $V_{alternative}$ are sets of *selectors* representing the policies (or, better, the obligations) $\{P_1, .., P_i, .., P_n\}$ that apply to the requestor. We also introduce a constraint of mutually exclusion between V_{allow} and V_{deny}, so that if $V_{allow} \equiv \emptyset \iff V_{deny} \neq \emptyset$, because all obligations associated to a given Requestor have to be consistent with respect to the Task value (*deny* or *allow*). This is enforced both at the time of loading the policies in the system, by checking that only one type of obligations is present (allow or deny), and at the time of policies editing, by preventing the policy editor to put both deny and allow types of obligations. This implies the Filter Updater create and update the filter V_u by maintaining two selectors Q_i and QA_i for each policy P_i so that, each element V_{allow} or V_{deny}, and $V_{alternative}$ are composed by a set of selectors $V_{allow} = \{Q_1, .., Q_i, .., Q_n\}$ or $V_{deny} = \{Q_1, .., Q_i, .., Q_n\}$ depending on the Task value (allow or deny) and mutually exclusive, and $V_{alternative} = \{QA_1, .., QA_i, .., QA_n\}$. We represent a selector Q_i or QA_i as a SPARQL CONSTRUCT query composed by the following elements:

```
CONSTRUCT {
  [RD]}
WHERE {
  [GP]}.
```

RD is the Result Descriptor (RD) which is triples pattern representing the triples the query will return to the user, while GP is the Graph Pattern (GP) in the RDF graph that will extract the triples from the RDF graph to filter and that will be used to create RD. Let us assume that all users subscribe to the log at the same time `t=0`. At `t=0`, all access policies get simultaneously enforced and all filters are empty. Each time a monitor generates a new triple, the Filter Updater checks if the Requestor u is allowed to access the triple by invoking the PEP. The results of these calls update the selectors Q_i and QA_i. Each call to the PEP is performed by passing the triple to filter, the PEP and returns an element containing the policy's unique identifier, the type of Task (allow or deny), the Match block and the satisfied conditions or the Alternative block in case the triple needs to be filtered out. The results returned by the PEP to the Filter Updater are used to update the two selectors Q_i and QA_i of the policy P_i represented by the V_{allow}, V_{deny} and $V_{alternative}$ sets in the filter V_u. The selectors are updated according to the templates shown in Table 2. Each element contained in the response from the PEP is replaced in the template: in case the Match block is returned, the element $[RD]$ is replaced by the graph pattern contained in the Match block and the $[GP]$ element is replaced by the graph pattern in the Match combined with the graph patterns of the elements in the Condition block. In case an Alternative block is returned, the element $[RD]$ is replaced by the triples in the graph pattern defined in the

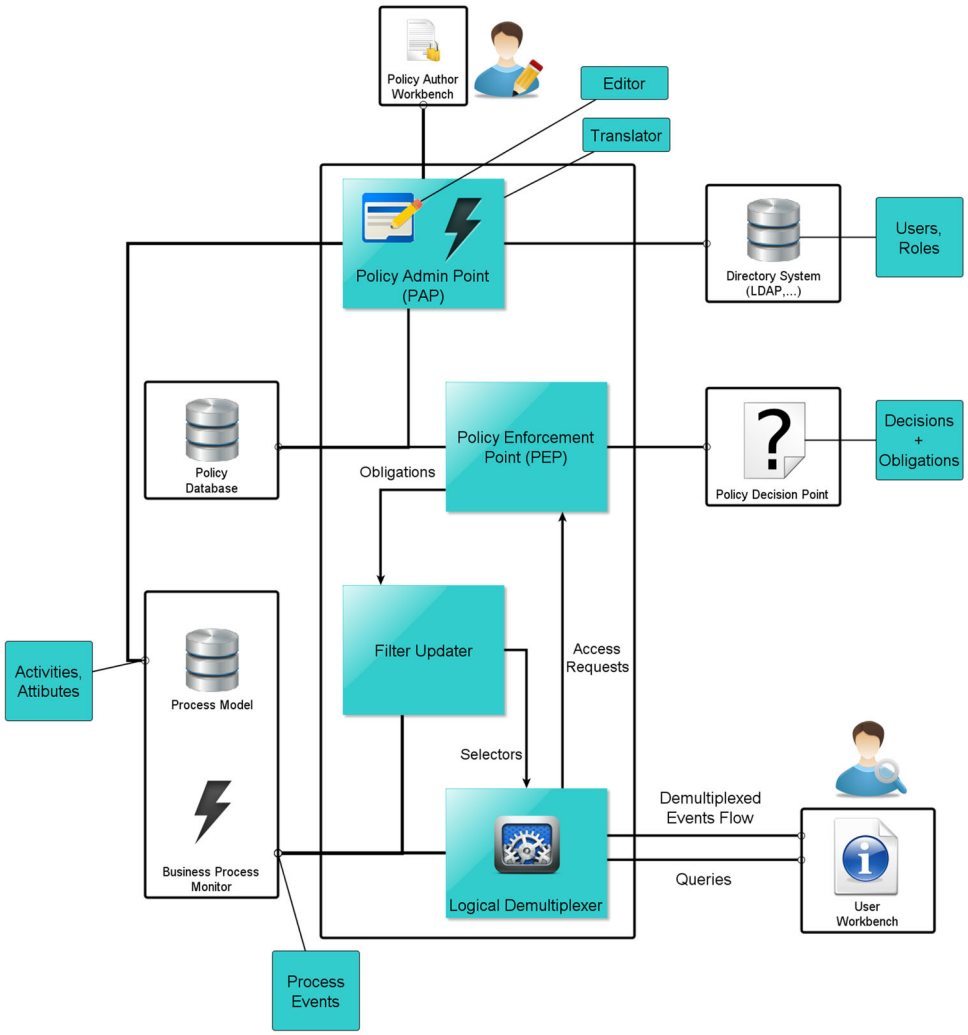

Fig. 3 Conceptual architecture of the AC system

Table 2 Selector templates for each policy P_i in case of allow types of policies

Q_i	QA_i

`CONSTRUCT {`
`  [match] }`
`WHERE {`
`  {[match].`
`  [conditions]}`
`  (UNION {[match].`
`  [conditions]})*}`
	`CONSTRUCT {`
	`  [alternative.replace]}`
	`WHERE {`
	`  [alternative.find].`
	`  (FILTER NOT EXISTS {`
	`  GP(Qi)})?`
	`  }`

Replace part of the alternative block and the $[GP]$ element is defined by the graph pattern defined in the Find part of the Alternative block minus the Graph Pattern $GP(Q_i)$ this in order to ensure that the selectors contained in $V_{alternative}$ only replace triples where the policy conditions apply. In case the policy does not apply to the triple, a `null` result is returned and the selectors Q_i and QA_i are not updated.

As an example let us consider the policy previously defined in 1 and the stream of triples reported in Table 3: when the first triple arrives, the Filter Updater asks the PEP if the user is allowed to see the triple, the PEP analyses the triple against the Match block. Since it does not match the graph patter in the Match block and the Task is of type deny, the user is allowed to see the triple.

At this time, the filter V_u is composed by the selector Q_1 in V_{deny}:

```
CONSTRUCT { ?s ?p ?o }
WHERE { ?s ?p ?o}.
```

This selector just returns all the triples in the original graph G since the user at the moment is allowed to see all the triples in the graph. The set of selectors in $V_{alternative}$ is empty because there is no need to sanitize the flow. When the fifth triple which matches the Match pattern arrives, the conditions need to be tested in order for the Filter Updater

Table 3 A sample stream of triples based on the RDF translation of the BPI Challenge 2012 [67]

s	p	o
bpi:process1	rdf:type	bpi:LoanProcess
bpi:process1	bpi:amount	120000
bpi:task1	rdf:type	bpi:A_SUBMITTED
bpi:agent1	rdf:type	bpi:human
bpi:task1	bpi:performed_by_agent	bpi:agent1
bpi:process1	ebtic-bpm:hasTask	bpi:task1
bpi:process1	ebtic-bpm:endTime	2010-03-02
bpi:process2	rdf:type	bpi:LoanProcess
bpi:process2	bpi:amount	100000
bpi:task2	rdf:type	bpi:A_SUBMITTED
bpi:agent2	rdf:type	bpi:human
bpi:task2	bpi:performed_by_agent	bpi:agent2
bpi:process2	ebtic-bpm:hasTask	bpi:task2
bpi:process2	ebtic-bpm:endTime	2010-03-02
bpi:process3	rdf:type	bpi:LoanProcess
bpi:process3	bpi:amount	200000
bpi:task3	rdf:type	bpi:A_APPROVED
bpi:agent3	rdf:type	bpi:human
bpi:task3	bpi:performed_by_agent	bpi:agent3
bpi:process3	ebtic-bpm:hasTask	bpi:task3
bpi:process3	ebtic-bpm:endTime	2010-03-02
bpi:process4	rdf:type	bpi:LoanProcess
bpi:process4	bpi:amount	100000
bpi:task4	rdf:type	bpi:A_ACCEPTED
bpi:agent4	rdf:type	bpi:human
bpi:task4	bpi:performed_by_agent	bpi:agent4
bpi:process4	ebtic-bpm:hasTask	bpi:task4
bpi:process4	ebtic-bpm:endTime	2010-03-02

to decide if the user is allowed to see the triple. The system generates a SPARQL ASK query for each condition by replacing the variables in the graph pattern with the elements of the triple (in this case the variables `?task` and `?agent` replaced with the subject and object of the triple, respectively) the conditions are tested until one is satisfied. In this case, the conditions are all tested but none of them is satisfied, so the filter cannot decide if the user is allowed to see the triple, so the triple is kept in a buffer until the decision point is satisfied (seventh triple); testing the decision point is done in the same way than testing the conditions, and it is done for each triple kept in the buffer. The selectors are not updated until the arrival of the triple, which satisfies the first condition. At this point, the PER return to the filter updater a response saying that the user is not allowed to see the triple and the elements required for updating the selectors which at this point are the ones reported in Table 4.

The filter updater processes all the triples in Table 3, and at the end of the flow, the selectors Q_1 and QA_1 will look like the ones in Appendix A.

6.2 The RDF Stream Demultiplexer

The RDF Stream Demultiplexer (Fig. 3) makes our dynamic selectors available as SPARQL endpoints to which the Requestor can freely submit her queries. The Demultiplexer component provides Requestors with the abstraction of separate logical process event flows. We remark that our Demultiplexer is fully agnostic with respect to the choice between materializing the user's graph G_u and using a query rewriting technique. In the former case, the Demultiplexer applies the selectors computed by the Updater to the original graph G as a dynamic continuous query [43] materializing the graph G_u, whenever a new triple passes the filter. The materialized graph G_u can be consulted by the user using SPARQL query language transparently as a traditional triple store. This approach maximizes performance but requires additional storage space in order to physically store the users' graphs. It is a solution that is not advisable in case of high number of different users with short-lived sessions but can be safely adopted for testing purposes and experi-

Table 4 Selectors after the first condition is satisfied

Q_1	QA_1

CONSTRUCT {
?s ?p ?o }
WHERE {
{?s ?p ?o
FILTER(NOT EXISTS{
FILTER (?s = ?x0 OR ?s = ?x1 OR ?s = ?x2)
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x1 rdf:type bpi:A_SUBMITTED .
?x2 bpi:amount 100000.
}}
	CONSTRUCT {
	?x1 bpi:agent ?x4.
	?x2 ebtic-bpm:hasTask ?x1
	?x4 rdf:type bpi:unknown.}
	WHERE{?x1 bpi:agent ?x0.
	?x0 rdf:type bpi:human.
	?x2 ebtic-bpm:hasTask ?x1.
	FILTER(EXISTS{?x2 ebtic-bpm:hasTask ?x1.
	?x1 bpi:agente ?x0.
	?x0 rdf:type bpi:human.
	?x2 ebtic-bpm:endTime ?x3.
	?x1 rdf:type bpi:A_SUBMITTED .
	?x2 bpi:amount 100000.}

mentation. In the latter case, the Demultiplexer simply waits for the requestor to submit a query. When the query arrives, our Demultiplexer uses the current set of selectors in V_u to rewrite the original user query Q_u with the selectors in the filter V_u, applying a standard query rewriting algorithm ([65]) and executed the query on the original graph G. The query rewriting technique is sound and complete, as shown in Sect. 3.2, and it ensures that the triples returned by the query are equivalent to executing the SPARQL query to the sanitized graph G_u, the difference, in this case, is that G_u is not materialized. Depending on deployment constrains (space, processing power, number of users and policies), one deployment solution may be preferable to the other.

7 Validation

To validate our approach, we considered three aspects: correctness, completeness, and performance. *Correctness* can be formally proved, as illustrated in Sect. 7.1. *Completeness* requires the availability of ground truth and will then be assessed only on a restrict set of triples, Sect. 7.1. *Performance* is analyzed against the SQR approach, as discussed in Sect. 7.2.

7.1 Correctness and Completeness

To discuss the correctness of our approach, we have to focus on the construction of the filter V_u.

Each of the selectors Q_i and QA_i created by the filter updater for each policy P_i returns a view over the original RDF graph G. As discussed previously in Sect. 5, our policy language allows to define two types of obligations: *allow* and *deny* which are mutually exclusive ($V_{allow} \equiv \emptyset \iff V_{deny} \neq \emptyset$). We can define an obligation P_i as the pair $\{Q_i \in (V_{allow} \cup V_{deny}), QA_i \in (V_{alternative})\}$. With the notation $[[Q]]_G$, we define the result of the execution of a query Q over an RDF graph G, which in case of a CONSTRUCT query is an RDF graph.

Let us start from the case of *allow* type obligations, and the sanitized graph G_{P_i} related to a specific policy is the union of the results from the queries Q_i and QA_i which can be defined as:

$$G_{P_i} \equiv [[Q_i]]_G \cup [[QA_i]]_G,$$

since the final graph for a user u is the graph G_u returned by the execution of all the policies we can define:

$$G_u \equiv \bigcup_{i=0}^{n=|P_i|} G_{P_i} \equiv \bigcup_{i=0}^{n=|P_i|} ([[Q_i]]_G \cup [[QA_i]]_G),$$

which, for the commutative property of the union operator, can be rewritten as:

$$G_u \equiv \left(\bigcup_{i=0}^{n=|P_i|} [[Q_i]]_G \right) \cup \left(\bigcup_{i=0}^{n=|P_i|} [[QA_i]]_G \right),$$

if we define

$$G_{u\_allow} \equiv \bigcup_{i=0}^{n=|P_i|} [[Q_i]]_G,$$

$$G_{u\_alternative} \equiv \bigcup_{i=0}^{n=|P_i|} [[QA_i]]_G,$$

then, the sanitized graph G_u can be defined as:

$$G_u \equiv G_{u\_allow} \cup G_{u\_alternative},$$

which is the union of the results of the single executions of the selectors contained in V_{allow} and $V_{alternative}$ (as defined using the templates in Table 2).

In the case, the Task value of the policy is of *deny* type, and G_{allow} will be the set-theoretical difference between the original RDF graph G and the union of all the triples the user is not authorized to access (V_{deny}) [7]. This can be defined as:

$$G_{u\_allow} \equiv G \setminus G_{u\_deny} \equiv G \setminus \left(\bigcup_{i=0}^{n=|P_i|} [[Q_i]]_G \right),$$

which can be rewritten as the intersection of the graphs generated by the single execution of the queries:

$$G_{u\_allow} \equiv \left(\bigcap_{i=0}^{n=|P_i|} G \setminus [[Q_i]]_G \right),$$

this aspect needs to be considered in case the *deny* queries are divided into separate executions. By definition [44], the results of such queries are themselves RDF graphs; therefore, the result of applying V_u over the original graph G is an RDF graph G_u, which contains only triples that the user is authorized to access.

To verify the completeness of our approach, it is required to manually validate the triples resulted in the RD generated by the Demultiplexer. During our evaluation, we tested around 400 triples without observing any misalignment with a ground truth result set used for comparison.

7.2 Performances

Generally speaking, the complexity of the evaluation problem for SPARQL queries [68] is known to be NP-complete in case no `OPTIONAL` operators are present in the query, while is PSPACE-complete in the presence of `OPTIONAL` operators. Still, efficient execution of SPARQL has been achieved in many practical systems by using data partitioning heuristics [69].

In our policies, we do not impose any restriction to the use of the `OPTIONAL` operator, which can be present in the *conditions* block of the policy. However, boolean operators used to combine different conditions inside the *conditions* block can be used to limit the need for `OPTIONAL` operator only to exceptional cases. In order to validate our approach, we designed a set of experiments to measure performance improvement provided by our dynamic filtering enforcement approach with respect to an approach that statically apply filters such as SQR [65]. In fact, SQR can dynamically generate queries, but the enforcement of the filtering procedure is statically applied to any triple it processes. This means that SQR can be applied to a stream of RDF triples, but the triples are filtered by applying the entire policy, making it inefficient. Our approach builds the filter dynamically by applying only the necessary part of the policy. The experiment presented in this section is intended to show that, even given the high complexity of SPARQL query answering, our approach provides a viable solution for a practical PMon system, suitable for further improvement, e.g., by data partitioning heuristics. Since stream filter execution heavily impacts the overall performance of PMon, an improvement in this aspect positively affects the entire behavior of the system. We implemented our AC mechanism as a set of components for the Zeus process analyzer [43]. In our implementation, Requestors request a SPARQL endpoint address or submit a SPARQL query through a Web Service interface, after standard authentication. The Web Service takes care of passing the Requestor's credentials to the other components where policies are extracted and applied. In our experiment, the Demultiplexer physically decouples the RDF graph G_u from the original graph G of triples. The data used for this experiment are the BPI Challenge 2012 [67] introduced in Sect. 2. The log is composed of 13.087 process instances and 262.200 activity instances divided into 24 different activity types. The log is available in the OpenXES[8] format, and therefore it has been converted into RDF in order to be used with our system. The resulting RDF graph is composed of 2.379.557 triples. Experiments were carried out on a desktop pc with processor Intel core i5 2,53 ghz with 8 GBytes (1067 MHz) of RAM memory and Hard Disk of 500 GBytes (5400 rpm). The test-suite has been developed using Java[9] version 7. The test-suite is composed by a Java implementation of the AC module, represented by the conceptual architecture in Fig. 2 and described in detail in Sect. 6, the flow of events, a log re-player and a flow listener. The flow of events is represented by a message queue[10] ensuring that the overhead introduced by the message queue does not influence the performance of our approach. The events (represented by the triples in the RDF log) are inserted in the message queue by a Log re-player, which reads the RDF representation of the BPI Challenge Log and submits the triples to the message queue with a configurable delay between triples. From our experience in a real deployment of the analyzer, we observed that the arrival rate of triples is not constant: normally the process monitors generate bursts of triples to represent new activities in the monitored process. In the BPI Challenge log, an activity is defined by a block of 10 triples providing information on the identifier of the activity, its type, the process the activity belongs to, preceding activities, the start time and end time of the activity and the values of its attributes. This behavior has been simulated in the log re-player which sends to the

[7] Traditional view-based approaches do not consider this types of policies [65].

[8] http://www.xes-standard.org/.

[9] http://www.oracle.com/technetwork/java/.

[10] for this experiment we used ActiveMQ http://activemq.apache.org/.

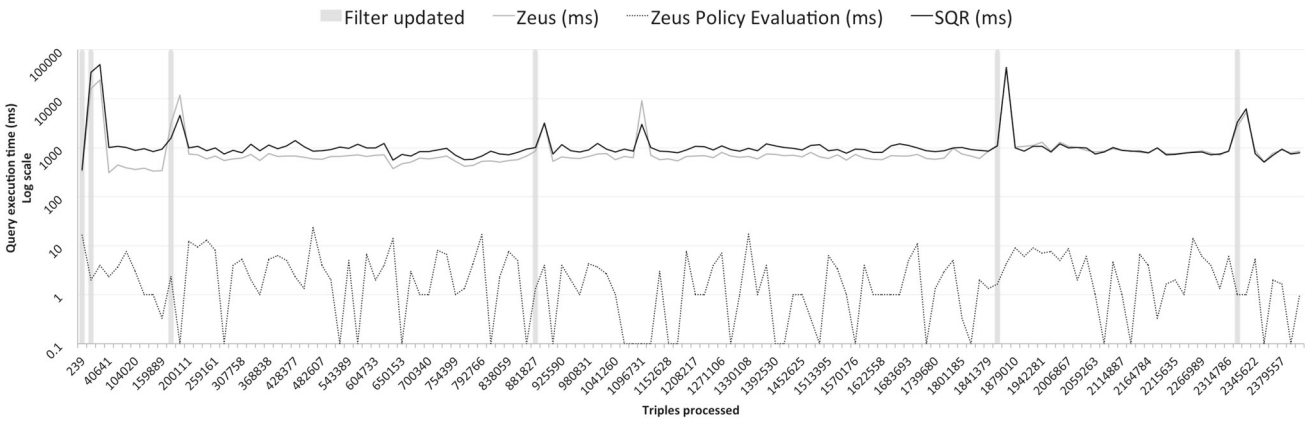

Fig. 4 Performance results of dynamic filter

message queue blocks of 10 triples with a variable delay with an average value of 1000 ms between the blocks. The benchmark represents the worst case scenario for our approach: the number of different activity types is relatively small. The Listener is invoked every time a triple is detected in the message queue and takes care of testing the triples against the PEP (Fig. 3) informing the Filter Updater with the obligations to apply to the triples flow. In order to simulate a stream analysis environment, the cache of the events log where the conditions are tested is cleaned and the RDF triples made persistent in the final triple store. The policy used for the experiments is the one defined in 1. The queries we used in the static SQR test are reported in Appendix A. (They are equal to the selectors when all the policy is applied.) We executed three runs for each test and compared the average of the resulting times.

Results of our evaluation are shown in Fig. 4, which shows query execution time in milliseconds (ms). It compares our dynamic filtering approach versus the SQR static approach, the X axis reports the number of triples processed. In the chart is also present the time of execution of the queries used to check the conditions performed by our PEP. In our experimentation, RDF triples are regularly removed from the stream and stored in the final repository. (Every 20000 complete processes, the stream is cleared.) The vertical bars in the chart represent the moments in the stream where one of the conditions of the policy is verified. It is possible to see how our approach remains well under the static one until the fifth vertical bar when the selectors of the filter updater became equivalent to the queries executed by the static approach. It is also worth noticing that the time of execution of the testing of the conditions is minimal [11]. Also, our test has been carried out on one policy at a time; in a production environment where multiple AC policies are applied, the improvement in query execution time due to building the filter dynamically will improve.

[11] The spikes in the chart are due to the Java garbage collector when the triples buffer is cleaned.

8 Conclusions and Perspectives

In this paper, we presented a model and a toolkit for controlling access to RDF streams generated by business process monitoring. Our approach introduces a new notion of dynamic enforcement: dynamic filters are incrementally built for each process monitoring instance, based on the applicable AC policy, on the requestor and on the current content of the process event log. We consider dynamic filtering a major innovation in enforcing multiple AC policies on huge graph streams. When static filters are used, all user queries use identical filters for determining which events of the audit log they can access. Our dynamic filters respond to real-time events in the process environment, setting only the traps that suit the specific process trace when selecting events to audit. Policy-wise, our approach is based on obligations in the standard XACML language and enables auditing of AC policies on work-flow streams like any other policy expressable in XACML. Today, results of audits on many compliance-driven (e.g., "need-to-know") mandates depend on the audited organization having certified access control regimes in place. We consider this to be a key factor for the practical adoption of our approach. A related line of research regards extending the scope of our XACML obligations. It is important to remark that this would not require extending their syntax: in the spirit of kindred languages like PAPEL [57], it would suffice to add additional layers of metadata expressing the timing (and possibly the physical location of probes) of process event generation. RDF formats for such layers have been available since long [70], and conditions on such additional metadata layers can be straightforwardly expressed using our current obligations. We see a number of other directions for further developing our approach. First of all, we designed our approach to be suitable for Big Data-style, low-level parallelization by executing our filters in parallel over a high number of data partitions corresponding to probe locations. We plan to explore this aspect in a future paper. Another aspect worth exploring is using reasoning to

keep our dynamic filter compact. In our current implementation, when a Zeus process monitor submits a triple to which a policy applies, the dynamic filter gets updated unless it already contains a specific selector for the incoming triple. However, it may conceivably be the case that the filter already contains a selector that will filter out that triple anyway (e.g., because it filters a concept that subsumes concepts related to the incoming triples).

If the Filter Updater could notice this *selector absorption*, it would avoid adding a condition to the filter that will never be checked. However, the amount of reasoning to be done to notice absorption may in some cases be substantial and the eventual gain (or loss) in performance would largely depend on the log content. Trying to take into account absorption in filter construction may also interfere with automatic cost reordering of conjunctions in SPARQL query planning, a crucial issue for SPARQL performance [68]. We plan to explore this subject in a future paper.

Acknowledgements This work has been partly supported by Abu Dhabi-IT Fund under Project 3093 "Grid-Enabled Business Process Management and e-Government."

Appendix A

Selector Q_1 in V_{deny}

```
CONSTRUCT {?s ?p ?o .}
WHERE {?s ?p ?o .
FILTER(NOT EXISTS {
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x1 rdf:type bpi:A_SUBMITTED .
?x2 bpi:amount 100000 .
FILTER (?s = ?x0 OR?s = ?x1 OR ?s = ?x2)}&& NOT EXISTS {
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x1 rdf:type bpi:A_APPROVED .
?x2 bpi:amount 200000 .
FILTER (?s = ?x0 OR?s = ?x1 OR ?s = ?x2)}&& NOT EXISTS {
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x2 bpi:amount 100000 .
?x1 rdf:type bpi:A_ACCEPTED .
FILTER (?s = ?x0 OR?s = ?x1 OR ?s = ?x2)})}
```

Selector QA_1 in $V_{alternative}$

```
CONSTRUCT{?x1 bpi:agente ?x4.
?x2 ebtic-bpm:hasTask ?x1.
?x4 rdf:type bpi:unknown.}
WHERE{
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:hasTask ?x1.
FILTER (EXISTS{
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x1 rdf:type bpi:A_SUBMITTED .
?x2 bpi:amount 100000 .}OR  EXISTS{
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x1 rdf:type bpi:A_APPROVED .
?x2 bpi:amount 200000 .}OR  EXISTS{
?x2 ebtic-bpm:hasTask ?x1.
?x1 bpi:agente ?x0.
?x0 rdf:type bpi:human.
?x2 ebtic-bpm:endTime ?x3.
?x2 bpi:amount 100000 .
?x1 rdf:type bpi:A_ACCEPTED .})}
```

References

1. Shtub A, Karni R (2010) Business process improvement. In: ERP. Springer US, pp 217–254. https://doi.org/10.1007/978-0-387-74526-8_13
2. van der Aalst W, Adriansyah A, de Medeiros AKA, Arcieri F, Baier T, Blickle T, Bose JC, van den Brand P, Brandtjen R, Buijs J, Burattin A, Carmona J, Castellanos M, Claes J, Cook J, Costantini N, Curbera F, Damiani E, de Leoni M, Delias P, van Dongen BF, Dumas M, Dustdar S, Fahland D, Ferreira DR, Gaaloul W, van Geffen F, Goel S, Günther C, Guzzo A, Harmon P, ter Hofstede A, Hoogland J, Ingvaldsen JE, Kato K, Kuhn R, Kumar A, La Rosa M, Maggi F, Malerba D, Mans RS, Manuel A, McCreesh M, Mello P, Mendling J, Montali M, Motahari-Nezhad HR, zur Muehlen M, Munoz-Gama J, Pontieri L, Ribeiro J, Rozinat A, Seguel Pérez H, Seguel Pérez R, Sepúlveda M, Sinur J, Soffer P, Song M, Sperduti A, Stilo G, Stoel C, Swenson K, Talamo M, Tan W, Turner C, Vanthienen J, Varvaressos G, Verbeek E, Verdonk M, Vigo R, Wang J, Weber B, Weidlich M, Weijters T, Wen L, Westergaard M, Wynn M (2012) Process mining manifesto. In: Daniel F, Barkaoui K, Dustdar S (eds) Business process management workshops. Springer, Berlin, pp 169–194
3. Alles M, Brennan G, Kogan A, Vasarhelyi MA. Continuous monitoring of business process controls: a pilot implementation of a continuous auditing system at siemens, ch. 10, pp

219–246. https://www.emeraldinsight.com/doi/abs/10.1108/978-1-78743-413-420181010
4. Barbon S, Junior, Tavares GM, da Costa VGT, Ceravolo P, Damiani E (2018) A framework for human-in-the-loop monitoring of concept-drift detection in event log stream. In: Companion proceedings of the web conference 2018, ser. WWW '18. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, pp 319–326. https://doi.org/10.1145/3184558.3186343
5. Sandhu R, Samarati P (1994) Access control: principle and practice. IEEE Commun. Mag. 32(9):40–48
6. Gunther CW, van der Aalst WMP (2006) A generic import framework for process event logs. In: Business process management workshops, workshop on business process intelligence (BPI 2006), volume 4103 of lecture notes in computer science, Tech. Rep
7. Aalst Wvd, Damiani E (2015) Processes meet big data: connecting data science with process science. IEEE Trans Serv Comput 8(6):810–819
8. Ceravolo P, Azzini A, Angelini M, Catarci T, Cudré-Mauroux P, Damiani E, Mazak A, Van Keulen M, Jarrar M, Santucci G, Sattler K-U, Scannapieco M, Wimmer M, Wrembel R, Zaraket F (2018) Big data semantics. J Data Semant 7(2):65–85. https://doi.org/10.1007/s13740-018-0086-2
9. eXtensible Access Control Markup Language (XACML) version 2.0, OASIS access control TC, Tech. Rep., Feb 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
10. Hayes P,McBride B (2004) Resource description framework (RDF), W3C, recommendation. http://www.w3.org/TR/rdf-mt/
11. Thomas O, Fellmann M (2007) Semantic EPC: enhancing process modeling using ontology languages. In: Hepp M, Hinkelmann K, Karagiannis D, Klein R, Stojanovic N (eds) SBPM, ser. CEUR workshop proceedings, vol 251. CEUR-WS.org
12. Ferraiolo D, Kuhn R (1995) Role-based access control (RBAC): features and motivations. http://csrc.nist.gov/rbac/
13. Chowdhury O, Chen H, Niu J, Li N, Bertino E (2012) On XACML's adequacy to specify and to enforce HIPAA, In: Proceedings of the 3rd USENIX conference on health security and privacy, ser. HealthSec'12. USENIX Association, Berkeley, CA, USA, pp 11–11. http://dl.acm.org/citation.cfm?id=2372366.2372381
14. Banisar D, Davies SG (1999) Global trends in privacy protection: an international survey of privacy, data protection, and surveillance laws and developments. John Marshall J Comput Inf Law XVIII. http://ssrn.com/abstract=2138799
15. Al-Ali H, Damiani E, Al-Qutayri M, Abu-Matar M, Mizouni R (2016) Translating BPMN to business rules. In: International symposium on data-driven process discovery and analysis. Springer, pp 22–36
16. Ceravolo P, Damiani E, Torabi M, Barbon S (2017) Toward a new generation of log pre-processing methods for process mining. In: Carmona J, Engels G, Kumar A (eds) Business process management forum. Springer, Cham, pp 55–70
17. Verbeek H, Buijs JC, Van Dongen BF, Van Der Aalst WM (2010) Xes, xesame, and prom 6. In: Forum at the conference on advanced information systems engineering (CAiSE). Springer, pp 60–75
18. Li G, de Murillas EGL, de Carvalho RM, van der Aalst WM (2018) Extracting object-centric event logs to support process mining on databases. In: International conference on advanced information systems engineering. Springer, pp 182–199
19. Schönig S, Rogge-Solti A, Cabanillas C, Jablonski S, Mendling J (2016) Efficient and customisable declarative process mining with SQL. In: International conference on advanced information systems engineering. Springer, pp 290–305
20. Leida M, Majeed B, Colombo M, Chu A (2013) A lightweight RDF data model for business process analysis, vol. 162. In: Cudre-Mauroux P, Ceravolo P, Gašević D (eds) Data-driven process discovery and analysis, vol 162. Lecture notes in business information processing. Springer, Berlin, pp 1–23. https://doi.org/10.1007/978-3-642-40919-6_1
21. Vogelgesang T, Appelrath H-J (2015) A relational data warehouse for multidimensional process mining. In: International symposium on data-driven process discovery and analysis. Springer, pp 155–184
22. Berberi L, Eder J, Koncilia C (2018) A process warehouse model capturing process variants. Enterp Model Inf Syst Archit 13:77–85
23. van der Aalst WMP (2013) Process cubes: slicing, dicing, rolling up and drilling down event data for process mining. In: Song M, Wynn M, Liu J (eds) Asia Pacific business process management, vol 159. Lecture notes in business information processing. Springer, Berlin, pp 1–22. https://doi.org/10.1007/978-3-319-02922-1_1
24. Ceravolo P, Azzini A, Damiani E, Lazoi M, Marra M, Corallo A (2016) Translating process mining results into intelligible business information. In: Proceedings of the 11th international knowledge management in organizations conference on the changing face of knowledge management impacting society. ACM, p 14
25. Ceravolo P, Zavatarelli F (2015) Knowledge acquisition in process intelligence. In: 2015 international conference on information and communication technology research (ICTRC). IEEE, pp 218–221
26. Calvanese D, Montali M, Syamsiyah A, van der Aalst WMP (2016) Ontology-driven extraction of event logs from relational databases. In: Reichert M, Reijers HA (eds) Business process management workshops. Springer, Cham, pp 140–153
27. Ardagna CA, Ceravolo P, Damiani E (2016) Big data analytics as-a-service: issues and challenges. In: 2016 IEEE international conference on big data (big data), pp 3638–3644
28. Smirnov S, Reijers HA, Weske M (2011) A semantic approach for business process model abstraction. In: Mouratidis H, Rolland C (eds) Advanced information systems engineering. Springer, Berlin, pp 497–511
29. Azzini A, Ceravolo P (2013) Consistent process mining over big data triple stores. In: 2013 IEEE international congress on big data (BigData Congress). IEEE, pp 54–61
30. Nykänen O, Rivero-Rodriguez A, Pileggi P, Ranta PA, Kailanto M, Koro J (2015) Associating event logs with ontologies for semantic process mining and analysis. In: Proceedings of the 19th international academic mindtrek conference, ser. AcademicMindTrek '15. ACM, New York, NY, USA, pp 138–143. https://doi.org/10.1145/2818187.2818273
31. de Medeiros AKA, van der Aalst W, Pedrinaci C (2008) Semantic process mining tools: core building blocks. In: 16th European conference on information systems. http://oro.open.ac.uk/23397/
32. Okoye K, Tawil ARH, Naeem U, Lamine E (2015) Semantic process mining towards discovery and enhancement of learning model analysis, In: 2015 IEEE 17th international conference on high performance computing and communications, 2015 IEEE 7th international symposium on cyberspace safety and security, and 2015 IEEE 12th international conference on embedded software and systems, pp 363–370
33. Cairns AH, Ondo JA, Gueni B, Fhima M, Schwarcfeld M, Joubert C, Khelifa N (2014) Using semantic lifting for improving educational Please provide accessed date for reference Marr (2017).and analysis. In: Proceedings of the 4th international symposium on data-driven process discovery and analysis (SIMPDA 2014). http://ceur-ws.org/Vol-1293/paper11.pdf. Accessed May 2019
34. Kingsley O, Tawil ARH, Naeem U, Islam S, Lamine E (2016) Using semantic-based approach to manage perspectives of process mining: application on improving learning process domain data. In: 2016 IEEE international conference on big data (big data), pp 3529–3538
35. Azzini A, Braghin C, Damiani E, Zavatarelli F (2013) Using semantic lifting for improving process mining: a data loss prevention system case study, pp 62–73

36. Brickley D, Guha R, McBride B (2004) RDF vocabulary description language 1.0: RDF schema, W3C, recommendation. http://www.w3.org/TR/rdf-schema/
37. Simmhan YL, Plale B, Gannon D (2005) A survey of data provenance in e-science. SIGMOD Rec 34(3):31–36. https://doi.org/10.1145/1084805.1084812
38. Baier T, Mendling J (2013) Bridging abstraction layers in process mining by automated matching of events and activities. In: Daniel F, Wang J, Weber B (eds) Business process management. Springer, Berlin, pp 17–32
39. Al-Ali H, Damiani E, Al-Qutayri M, Abu-Matar M, Mizouni R (2018) Translating BPMN to business rules. In: Ceravolo P, Guetl C, Rinderle-Ma S (eds) Data-driven process discovery and analysis. Springer, Cham, pp 22–36
40. De Nicola A, Di Mascio T, Lezoche M, Tagliano F (2008) Semantic lifting of business process models. In: 2008 12th enterprise distributed object computing conference workshops. IEEE, pp 120–126
41. Mannhardt F, De Leoni M, Reijers HA, Van Der Aalst WM, Toussaint PJ (2016) From low-level events to activities—a pattern-based approach. In: International conference on business process management. Springer, pp 125–141
42. Baier T, Di Ciccio C, Mendling J, Weske M (2015) Matching of events and activities-an approach using declarative modeling constraints. In: International conference on enterprise, business-process and information systems modeling. Springer, pp 119–134
43. Leida M, Chu A (2013) Distributed SPARQL query answering over RDF data streams. In: IEEE international congress on big data (bigdata congress), pp 369–378
44. Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF, W3C, recommendation. http://www.w3.org/TR/rdf-sparql-query/
45. Beheshti S-M-R, Benatallah B, Motahari-Nezhad H, Sakr S (2011) A query language for analyzing business processes execution. In: Rinderle-Ma S, Toumani F, Wolf K (eds) Business process management, vol 6896. Lecture notes in computer science. Springer, Berlin, pp 281–297. https://doi.org/10.1007/978-3-642-23059-2_22
46. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. Computer 29(2):38–47. https://doi.org/10.1109/2.485845
47. Herrmann G, Pernul G (1999) Viewing business-process security from different perspectives. Int J Electron Commerce 3(3):89–103
48. Koshutanski H, Massacci F (2003) An access control framework for business processes for web services. In: Proceedings of the 2003 ACM workshop on XML security, ser. XMLSEC '03. ACM, New York, NY, pp 15–24. https://doi.org/10.1145/968559.968562
49. Russell N, Aalst W, Hofstede A, Edmond D (2005) Workflow resource patterns: identification, representation and tool support. In: Pastor O, Falcão e Cunha J (eds) Advanced information systems engineering, vol 3520. Lecture notes in computer science. Springer, Berlin, pp 216–232. https://doi.org/10.1007/11431855_16
50. Wainer J, Kumar A, Barthelmess P (2007) DW-RBAC: a formal security model of delegation and revocation in workflow systems. Inf Syst 32(3):365–384. https://doi.org/10.1016/j.is.2005.11.008
51. Weber B, Reichert M, Wild W, Rinderle-Ma S (2005) Balancing flexibility and security in adaptive process management systems. In: Proceedings of the 2005 confederated international conference on the move to meaningful internet systems—volume part I, ser. OTM'05. Springer, Berlin, pp 59–76. https://doi.org/10.1007/11575771_7
52. Rinderle-Ma S, Reichert M (2008) Managing the life cycle of access rules in CEOSIS. In: Proceedings of the 2008 12th international IEEE enterprise distributed object computing conference, ser. EDOC '08. IEEE Computer Society, Washington, DC, USA, pp 257–266. https://doi.org/10.1109/EDOC.2008.16
53. Lehmann A, Fahland D (2012) Information flow security for business process models—just one click away. In: Lohmann N, Moser S (eds) BPM (Demos), ser. CEUR workshop proceedings, vol 940. CEUR-WS.org, pp 34–39. http://dblp.uni-trier.de/db/conf/bpm/bpmd2012.html#LehmannF12
54. Etcheverry L, Vaisman AA (2012) Views over RDF Datasets: a state-of-the-art and open challenges. CoRR arXiv:1211.0224
55. Bassil S, Reichert M, Bobrik R (2009) Access control for monitoring system-spanning business processes in Proviado. In: EMISA, pp 125–139
56. Reichert M, Bassil S, Bobrik R, Bauer T (2010) The Proviado access control model for business process monitoring components. Enterp Model Inf Syst Archit Int J 5(3):64–88
57. Ringelstein C, Staab S (2011) Papel: Provenance-aware policy definition and execution. IEEE Internet Comput 15(1):49–58
58. Polyvyanyy A, Smirnov S, Weske M (2009) The triconnected abstraction of process models. In: Proceedings of the 7th international conference on business process management, ser. BPM '09. Springer, Berlin, pp 229–244. https://doi.org/10.1007/978-3-642-03848-8_16
59. Greco G, Guzzo A, Pontieri L (2005) Mining hierarchies of models: from abstract views to concrete specifications. In: Proceedings of the 3rd international conference on business process management, ser. BPM'05. Springer, Berlin, pp 32–47. https://doi.org/10.1007/11538394_3
60. Javanmardi S, Amini M, Jalili R. An access control model for protecting semantic web resources
61. Dean M, Schreiber G, Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2004) Owl web ontology language reference, W3C, recommendation. http://www.w3.org/Submission/SWRL/
62. Damiani E, De S, Vimercati C, Fugazza C, Samarati P (2004) Extending policy languages to the semantic web. In: Proceedings of the international conference on web engineering, pp 330–343
63. Finin T, Joshi A, Kagal L, Niu J, Sandhu R, Winsborough W, Thuraisingham B (2008) ROWLBAC: representing role based access control in OWL. In: Proceedings of the 13th ACM symposium on Access control models and technologies, ser. SACMAT '08. ACM, New York, NY, USA, pp 73–82. https://doi.org/10.1145/1377836.1377849
64. Chen W, Stuckenschmidt H (2009) A model-driven approach to enable access control for ontologies. In: Wirtschaftsinformatik (1), pp 663–672
65. Le W, Duan S, Kementsietsidis A, Li F, Wang M (2011) Rewriting queries on SPARQL views, In: Proceedings of the 20th international conference on World wide web, ser. WWW '11. ACM, New York, NY, USA, pp 655–664. https://doi.org/10.1145/1963405.1963497
66. Aravind Yalamanchi SD, Banerjee Jayanta (2010) Access control for graph data, US Patent US20 100 268 722 A1, 10 21. http://www.patentlens.net/patentlens/patent/US_7062320/
67. van Dongen B (2012) BPI challenge 2012. 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
68. Pérez J, Arenas M, Gutierrez C (2009) Semantics and complexity of SPARQL. ACM Trans Database Syst 34(3):16:1–16:45. https://doi.org/10.1145/1567274.1567278
69. Wang X, Yang T, Chen J, He L, Du X (2015) Rdf partitioning for scalable SPARQL query processing. Front Compu Sci 9(6):919–933. https://doi.org/10.1007/s11704-015-4104-3
70. Sheth A, Henson C, Sahoo S (2008) Semantic sensor web. IEEE Internet Comput 12(4):78–83