IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 23, 2017, accepted November 16, 2017, date of publication November 27, 2017,
date of current version February 14, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2778029

Linked Data Authorization Platform

RISTE STOJANOV, SASHO GRAMATIKOV, IGOR MISHKOVSKI, AND DIMITAR TRAJANOV
Faculty of Computer Science and Engineering, Saints Cyril and Methodius University, 1000 Skopje, Macedonia
Corresponding author: Riste Stojanov (riste.stojanov@finki.ukim.mk)

ABSTRACT The expansion of the smart devices, the growing popularity of the social networks, and the wide spread of the corporate services impose huge amounts of heterogeneous data to be generated and stored in separate silos on a daily basis. Parts of this data are private and highly sensitive as they reflect owner's behavior, obligations, habits, and preferences. On the other hand, the emerging crowd services challenge the owners to expose these data in return to the convenience they offer. Therefore, it is imperative not only to protect the interaction with sensitive data, but also to selectively open it in an unharmful manner for the owner's personal integrity. One of the main enablers of the crowd services is the emerging linked data, which is all about opening heterogeneous knowledge from separate data silos. Its growing popularity encourages the data owners to publish their personal data in linked data format. The fusion of sensor, social, and corporate data opens new security challenges, which extend the standard security considerations toward more flexible and context aware authorization platforms. In this paper, we propose a linked data authorization (LDA) platform atop a policy language flexible enough to cover all newly emerged requirements, including context awareness. The proposed policy language extends the widely accepted W3C's SPARQL query language and leverages its expressiveness to protect every part of the data. The novelty of our LDA platform is its unique capability of design time policy validation through stand-alone testing, conflict detection, and overall protection coverage extraction.

INDEX TERMS Authorization platform, security policy language, data security, linked data, semantic web.

I. INTRODUCTION

In the smart world around us, billions of devices, users and applications continuously generate vast amount of heterogeneous data. These devices range from regular computers to mobile phones, wearables, and sensor nodes. Thanks to the Internet, they are connected to each other, enabling them to exchange and fuse data with the social media resources, which may be further integrated with the corporate databases. Such a data mixture provides situation awareness that empowers the applications and the human users to better understand their surrounding environment. Furthermore, combining multiple data sources can make the context aware applications capable of making intelligent decisions, regarding their environment changes. This fused data is continuously gathered and stored at various enterprise servers, cloud infrastructures or even on private storage servers.

The heterogeneity of the devices makes interoperability a challenging task because of the different nature of the generated data (temperature, light, video, text, location), the inconsistent quality, and the trustworthiness of the sources. In the past decades, the W3C recognized this problem and introduced new standards known as Web 3.0, or Semantic Web [1]. The main goal of these W3C standards is to move the World Wide Web towards the Web of Data, transforming the pages into linked data resources consumable by software agents. The Linked Data approach [2], [3] provides new ways of integrating and consolidating data from various and distributed sources, solving the issue of isolated data silos such as the traditional relational database systems. This approach enables publishing and contextual linking of data on the Web of Data. An illustration of some of the entities that publish data in the Linked Open Data (LOD) Cloud [4] is shown in Fig. 1. The LOD cloud[1] visualizes a subset of the datasets[2] published using the Linked Data principles [3].

The data we generate, directly or through the devices we posses, combined with the general public knowledge (such as DBpedia [5]) and the corporate databases can describe our habits, environment, and, in case of wearables, even our health. Since this data is of sensitive nature, it should remain private and protected. If not properly secured, it can cause

[1]Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/.

[2]http://stats.lod2.eu/

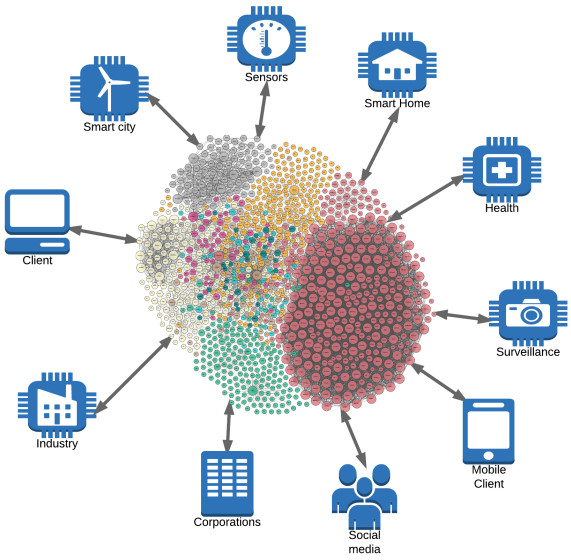

FIGURE 1. Linked open data cloud and data publishers.

serious privacy violation. Analysis of misuse of sensitive location data is shown in [6], where the authors discuss that the advertising industry largely benefits from personal information exposed by improper data handling. Nevertheless, when the user data is handled properly, it can be used to provide crowd sourced services that can significantly improve our lives. Such an example is the Google Traffic service[3] that uses user's locations to detect congested routes and propose alternative faster routes. This service saves precious time of the users and favors the general community.

The need for personalized, user defined protection of vast amounts of heterogeneous data has not been considered before in such scale. The security protocols, procedures and tools are always a step behind in handling new security challenges. When it comes to sensitive data, no matter whether it is personal, social or corporate, strict rules must be applied to ensure that it is properly accessed and handled. The data owners' ability to control who and under what conditions gets accesses to their data can encourage them to expose beneficiary data for the greater goods, and at the same time, protect their privacy. Tools that enable security policy testing and preview of the protected data are important step towards gaining trust in the authorization platforms.

The data storage and management systems expose their data through various processing actions, submitted by a requester, who may be a person, software agent or any other subject that can send its intention to the system. A requester is identified by the evidences it presents to the system, such as its authentication token or other environmental parameters, usually provided by the software agent. These evidences, together with the system's state, define the context in which the action executes. When an action is invoked, its operations, parameters and context describe the processing intention, or Intent.

The security rules are typically expressed as natural language requirements. Although the free text requirements have great expressiveness, their diversity makes them almost impossible to implement. Therefore, the requirements need to be formalized and translated into machine readable security policies that are easier to implement. The policy formalism settles the boundaries of free text requirement transformation into security policies, i.e., rules that define which parts of the data can be protected in a given contextual environment. Furthermore, the policy language provides a syntax that supports this formalism and simplifies the management process. In the traditional enterprises, the translation process is usually carried out by properly trained security officers and administrators. However, the scale of the data to be protected, its creation velocity and its heterogeneity makes the centralized policy management unfeasible. Therefore, the policy formalism should support tools that will enable regular users to protect their own heterogeneous and distributed data.

The design time validation of the policies' correspondence to the security requirements is crucial for proper data protection. The correspondence can be validated only if the policy language enables extraction of the data protected by the policy. Additionally, every well designed policy language should enable detecting and resolving possible conflicts in the policies.

Since the W3C's Semantic Web initiatives have defined standards that enable linking of the resources stored in different datasets [1], [7], and the Linked Open Data initiative have defined mechanisms for matching the same resources with different representations [3], one of the issues that remains open is the design of a policy language able to protect data stored or exposed as Linked Data [8], [9]. This policy language should provide various levels of protection granularity over distributed datasets and flexibility to match the natural language requirements. Apart from the flexibility, the policy language should be easy to understand and learn, and it should enable creating tools that will simplify the policy management and validation.

The main motivation of this paper is to design a platform that protects every interaction with arbitrary parts of the Linked Data and supports design time policy correctness validation. The proposed Linked Data Authorization (LDA) platform fulfills this goal and is able to protect read, insert, delete and manage operations using its flexible policy language that links the protected data to the authorization environment. The policy language is based on the well established W3C standardized SPARQL syntax[4] which enables reuse of the existing knowledge and tools for policy management simplification. Additionally, we implemented a policy management module that enables design time validation of the

[3]http://www.dailymail.co.uk/sciencetech/article-4706666/Google-introduces-traffic-time-maps.html, accessed 20 July 2017.

[4]SPARQL is the query language of the Semantic Web: https://www.w3.org/TR/rdf-sparql-query.

data protected by the policies, conflict detection and overall protected and unprotected data extraction.

The rest of the paper is organized as follows: in §II, the design principles of the LDA platforms used for comparison with the literature and assertion of our work are emphasized. Based on these platform design principles, §III provides literature overview and emphasizes the need for our LDA platform. In §IV, we describe the architecture of our LDA platform, together with the policy language description and enforcement implementation. In §V, the platform implementation is described, and use case is presented for better understanding. Afterwards, in §VI, we point out the strengths and weaknesses in respect to the defined design principles, together with performance evaluation of the implementation. Eventually, in §VII, we conclude our work.

II. LINKED DATA AUTHORIZATION DESIGN PRINCIPLES

Designing a complete authorization platform requires identifying security aspects for quality assessment. The expansion of Linked Data has motivated many researchers to work on security issues, either by restricting the set of available actions [10]–[12] or by filtering the data available for the operations [13]–[20]. In this section, we discuss the authorization aspects required for suitable protection not only of the data published using the Linked Data design principles, but also the data produced by various mobile devices and sensors, possibly stored in traditional database systems. Based on extensive literature review we have defined the following four design principles: (1) *flexibility* defines the need for covering wide range of requirements in the authorization system; (2) *maintainability* focuses on the policy management process optimization; (3) *correctness validation* emphasizes the importance of the policy testability, and (4) *understandability* settles the importance of using standardized syntax as basis of the policy language. Furthermore, we use these principles to analyze the related work in §III and lay the foundations of our policy language and authorization platform.

A. FLEXIBILITY

Flexibility is the ability to transform arbitrary free text authorization requirements into security policies. It defines the power of the policy language to express a range of different security requirements. According to [15], a policy can be generally formalized using the tuple $\langle R, D, AR \rangle$, which describes the access rights AR assigned to the requesters R (or Subject) for the data D (or resources). Regarding this policy formalization, there are different flexibility aspects that every modern LDA platform should meet:

1) OPERATION COVERAGE

This aspect defines the operations supported by the authorization platform, referred to as Permission Model in [21]. In this paper, we use the term *atomic operations* (or operations for brevity) for the inseparable actions that can be executed, while the term *action* is used to denote a more complex processing (usually from the business domain) that requires one or more operations to be invoked. Throughout the literature, the operations *read*, *insert* (often referred to as *create* [21] or *write* [10], [22]) and *delete* are usually covered. Inspired by [15], where the authors use the SPARQL query types as atomic operations for protection, in this paper, we additionally include the *manage* atomic operation, which unifies the named graph[5] creation, dropping, copying and moving interactions.

2) GRANULARITY

The Linked Data follows a directed graph structure where the nodes are represented by resources and literals, and the edges are represented by properties of the resources. The resources are used to describe real world or abstract concepts, and the literals represent primitive values that can be assigned as a property value for a resource. Every pair of connected nodes, including the property that connects them (edge), is referred to as triple, which can logically belong to a named graph and be physically stored in a dataset [1], [2].

The granularity aspect defines the ability of the policy language to capture the data structure and semantics at a various levels. The granularity level is determined through the ability to protect resources, triples, named graphs and datasets and is one of the most commonly discussed aspect in other authorization platforms [23]. In our work, we use the following features that enable various granularity levels for protection:

- *Triple Patterns (TP)* are essentially triples that can have variable at any position, used for selection of resources that have certain property or connection with other resource. The TP selection results with a set of triples.
- *Basic Graph Patterns (BGP)* are set of triple patterns that can capture more complex relationships among the resources.
- *Graph membership (Gm)* enables selection of triples that belong to a given named graph.
- *Dataset membership (DSm)* enables selection of triples that belong to a given dataset.

3) CONTEXT AWARENESS

The growing popularity and presence of sensing devices for ubiquitous computing make the context inevitable part of the modern systems. In order to provide suitable protection of the private and sensitive data they produce, a flexible authorization framework must take the context into account [11], [13], [21]. Nevertheless, the data context may infer an emergency situation that requires exposure of sensitive data to corresponding authorities so they can react appropriately. For example, a person wearing a heart-rate sensor may want to keep his/her measurements private until an emergency situation occurs. In such situations, the data needs to be opened to the doctor in order to react properly. Therefore,

[5]The *named graphs* provide logical organization of the Linked Data, whereas the datasets provide physical storage.

the authorization platform must provide both protection and exposure of the data, based on the context.

4) CONTEXT-DATA-REQUESTER ASSOCIATION

The authorization requirements may define protected data D in respect to the requester [20], [24], [25] and its environment [11], [13]. Therefore, the policy language must link the requester's properties, the guarded data and the contextual evidences.

5) AGGREGATED DATA SHARING

Crowd source applications provide many convenient services to the users in return to sharing of anonymized and aggregated sensitive data. Consequently, the authorization platforms should enable exposure of portions of the protected data based on user's choice.

6) CONFLICT RESOLUTION

Since most of the authorization platforms support multiple allow or deny policies, a conflict may arise when one policy allows and other denies interaction with the same portion of data for a given requester. Conflict resolution mechanisms [11], [15], [16], [24] are crucial for consistent data protection, giving the data owner possibility to choose which part of the conflicting data should be allowed.

B. MAINTAINABILITY

The maintainability is determined by the time it takes to transform the security requirements into policies. Mechanisms that support policy management can simplify this process and can provide more accurate protection [23], [26].

1) CONFLICT DETECTION

The conflicts are not always obvious and they may remain undetected for a long period of time [26], [27]. Therefore, a conflict detection mechanism should be provided in order to detect the conflicts at the very policy creation phase, preventing inappropriate data protection.

2) PROTECTED DATA COVERAGE

The data protected by each policy, along with the overall protected an unprotected data are very important for accurate data protection [30]. In an authorization platform that offers data coverage preview, the data owner is aware of the data portions that are unprotected and can react by specifying additional policies as needed. Lack of this kind of information may lead to ambiguities and exposure of protected data.

C. CORRECTNESS VALIDATION

The policy testing is a challenging task since the requesters and the context introduce high-dimensional space that is hard to be covered and understood. In order to ensure correctness during design time, it is necessary to validate that the policy corresponds to the requirement it represents [30]. Therefore, the authorization platform must enable mechanisms that will provide a preview of the data protected by the policy, so that the data owner can check if this data corresponds to the requirement.

D. UNDERSTANDABILITY

Understandability can be observed as the time it takes to master the policy format and language of the authorization platform [23], [31]. Hence, using a standardized policy language supported by a larger community can significantly reduce the learning time and make the language more understandable.

III. RELATED WORK

In order to review the related work in respect to the authorization principles in §II, we created Table 1, which summarizes and compares the most relevant Linked Data authorization approaches that support data protection.

The column *Operation* represents the *operation coverage* and shows that the *read* operation protection is the most popular. However, the policy languages that support Insert and Delete operations [10], [15], [21], [22], [25] lack flexibility for data selection with complex graph patterns with exception of the policy language in [25], where the authors just mention these operations without any explanation how they are enforced. The manage operation is only supported in [15]. In [11], the policies protect the actions as a whole using ontology defined activation context.

The column group *Granularity* covers the corresponding flexibility aspect in respect to the requesters environment (column group *Requester*) and the data being protected (column group *Protected data*). The ability to use triple patterns for the *Requester* and *Protected Data* is shown in the columns *TP*. The columns *BGP* show whether the data or the requester can be selected through basic graph patterns. In the columns *Gm* and *DSm*, the capability for representing graph and dataset membership is shown, correspondingly. In [10], [15], [17], and [18], the requester is specified by its Internationalized Resource Identifier (IRI),[6] by its class in [10], [15], and [29], and by its role attribute in [19] and [29]. The SPARQL query language used in [21] and [22] enables policy activation based on the requester's graph membership. When it comes to defining the protected data, the policy language from [25] offers best flexibility through the SPARQL expressiveness, followed by the policies that use SWRL [24] or equivalent syntax [13], [20]. However, these policies are limited regarding the graph membership.

The column *Context* describes whether the approach covers the *context awareness* considerations. Although the context is part of the policies in [13], the authors do not provide detailed explanation of the available evidences used to build it. The *Schi3ld framework* [21] uses static context represented with the prissma ontology,[7] while [11] uses dynamic context definition for evidence description.

The columns grouped as *Association* show whether it is possible to combine the context with the requester (column *CR*) or with the data (column *CD*), and whether the

[6] http://www.ietf.org/rfc/rfc3987.txt
[7] http://ns.inria.fr/prissma/v2/prissma_v2.html

TABLE 1. Flexibility authorization design principles coverage.

	Operation	Granularity: Requester TP	Requester BGP	Protected data TP	Protected data BGP	Protected data Gm	Protected data DSm	Context	Association CR	Association CD	Association RD
Hollenbach et al. [10]	R, I, D	p					✓				
Sacco et al. [28]	R, I, D	✓	✓	✓		✓					
Toninelli et al. [11]	A	p	p	p	p			✓	✓	✓	✓
Muhleisen et al. [24]	R	✓	✓	✓							✓
Flouris et al. [16]	R			✓	✓						
Kirrane [29]	R	p		✓		✓					
Dietzold & Auer [25]	R, I[?], D[?]	✓	✓	✓	✓	✓					✓
Costabello et al. [21]	R, I, D	✓	✓	p		✓		✓	✓		
Franzoni et al. [17]	R	p		✓	p						
Abel et al. [13]	R	✓	✓	✓	✓			✓	✓	✓	✓
Chen&Stuckenschmidt [19]	R	p		p							
Oulmakhzoune et al. [18]	R	p		✓	p						
Kirrane [15]	R, I, D, M	p		✓		✓					
Padia et al. [20]	R	✓	✓	✓							✓

TP - triple pattern; BGP - basic graph pattern; Gm - graph membership; DSm - dataset membership;
✓-yes; p-partially; i-resource IRI; c-Class; r-Role; R - Read; I - Insert; D - Delete; M - Manage; A - Action; []$^{?}$ - Not explained
C - context awareness; CR - context-requester association; CD - context-data association; RD - requester-data association;

approach supports combination of the requester's properties with the data (column *RD*). This group of columns represents the *context-data-requester association* consideration. Only [11] and [13] are able to link all properties and evidences together. The Proteus framework [11] activates the context based on description logic and logical programming rules, which links the requester to the data and the context. The work in [13] uses the Protune policy language [32] to define policies that implicate an allowed or denied triple based on a basic graph pattern matching. In [24] and [25], the policies are defined in the Semantic Web Rule Language (SWRL) [33], which enables combining the requester's properties with the data. A policy format with similar expressiveness to SWRL is employed in [20]. The work in [21] links the static context information with the requester's properties through a SPARQL query [34], [35] assigned to the activation policy's property. The policy languages that do not link the requesters' properties to the data usually have policy activation phase [10], [15], [17], [21], [22], which depends on the requester and the action it intends to execute. This activation phase selects the applicable policies that are later enforced. The enforcement phase can filter only the permitted data [13], [15]–[18], [20], [21], [24], [25], or it can allow or deny the requested action as a whole [10], [11].

The rest of the authorization principles are only marginally covered in the reviewed literature, and therefore, they are omitted in Table 1. The *conflict resolution* consideration is considered only in [13], [16], and [21]. All these approaches are using strategy for conflict resolution where the deny policies have higher precedence than the allowing one, or vice versa. However, the default strategy does not provide conflict resolution flexibility since the policy administrator cannot specify precedence of the policies. Custom algorithms for conflict resolution [15] partially solve this problem, but they are complex and require extensive testing and validation. The policy priorities [11], [36] provide the most flexible way of conflict resolution and are closest to the way people resolve conflicts.

The *conflict detection* is also required to detect requirement anomalies and to resolve them before a damage is made. Rules for policy anomaly detection able to detect contradiction or conflict are defined in [19], but their policy language lacks flexibility and the policy correspondence to the requirement is not considered. Similarly, in [31], a safety and consistency of the policies is considered through their interactions. Regarding *protected data coverage*, Flouris *et al.* [16] and Oulmakhzoune *et al.* [18] consider conflicts and policy data coverage, but they only discuss the default protection mechanisms for the guarded data and do not present this information during the policy maintenance.

Besides the flexibility, the *correctness* of the policies is rarely considered in the reviewed platforms. A methodology for implementation correctness validation of query rewriting authorization frameworks is proposed in [15], but it is not suitable for design time policy validation against the requirements. The policy language should enable designing tools that provide policy testing so that one can validate whether they meet the requirements they are designed for.

The main challenge that remains open is providing a design time validation of the policy correctness, while enabling flexibility to represent the natural language requirements. In this direction, the system presented in [13] uses the most complete policy language – Protune [32]. However, this system protects only the *read* operations and is unable to protect the data graph and dataset memberships. The authors do not discuss conflict detection and resolution. Only Kirrane [15] validates the implementation correctness of their authorization system, but they do not provide design time testability and correctness validation. In emergency situations, context awareness is of crucial importance for the data owner. In such scenarios, emergency policies should override the default system behavior and expose the sensitive private information to the

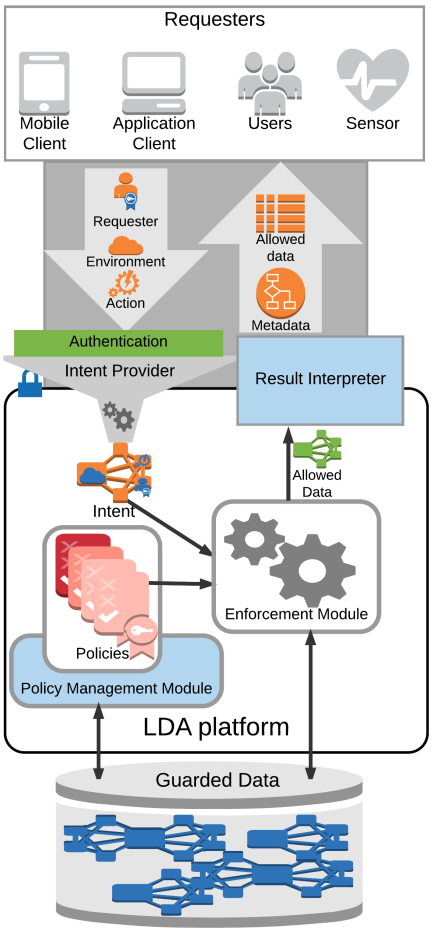

FIGURE 2. LDA platform architecture.

parties that can help the data owner. Moreover, exposing aggregated information that hides the sensitive personal information can enable emerging of new and more advanced crowd sourced services that will improve the quality of life in general. To the best of our knowledge, there is no policy language that provides this kind of protection. Our work aims to fill in the gaps of the existing approaches by proposing a flexible and complete LDA platform that covers all settled design principles in §II.

IV. LDA PLATFORM ARCHITECTURE

The goal of the LDA platform is to enforce the clients' security requirements represented in the platform as policies. The architecture of the LDA platform is shown in Fig. 2. The LDA platform protects the guarded data $\mathbb{D}$ (Definition 2) by enforcing the protection requirements represented by the security policies (Definition 5). The Intent (Definition 3) describes the action that the requester is attempting to execute, which, at some point, should interact with the guarded data $\mathbb{D}$. Figure 3 (a) shows the *allowed data* (Definition 4) synthesized by the *enforcement module* for multiple activated polices, which is a subset of the guarded data, while Figure 3 (b) additionally shows that the *enforcement module* filters and returns only the allowed portion of the requested results (denoted with IR).

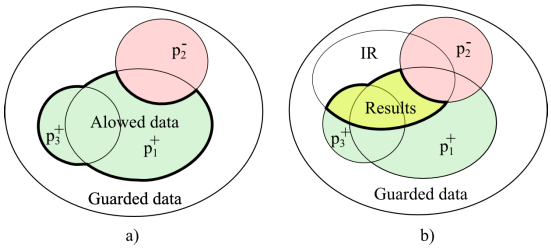

FIGURE 3. Data protection per intent.

One of the central components of the platform is the *IntentProvider* component that we designed to extract the Intent $\mathbb{I}$ from the available evidences in the platform (usually describing the requester, environment and action). This component provides context awareness by embedding dynamic context evidences in the Intent $\mathbb{I}$. Additionally, the *IntentProvider* is tightly coupled with the authentication process and it extracts the evidences presented after the requester is authenticated. The idea of the Intent component is based on our analysis on various distributed multiprocessing platforms that store and generate much of the todays data, especially the Android operating system *Intent*[8] component. The Android's Intent concept provides a flexible way of expressing both user intention and its environment, so we adopted this data structure in order to represent the requester's intention in the Linked Data management platforms. In the LDA platform, the Intent $\mathbb{I}$ is implemented as a set of RDF triples that describes the context in which the intended action executes, including the operation, the action parameters, the requester with its environmental evidences and the event that invoked the action. Additionally, the term intent is preferred because it represents the implicit user intention, but it is uncertain whether it will be permitted, denied or partially executed.

The *enforcement module* executes the algorithms that ensure the authorized interaction with the underlaying data. We defined separate enforcement algorithm for each operation, where the operation is executed against previously constructed *allowed data* (Definition 4) for the intent. The policies used to construct the allowed data are selected based on the requested operation and Intent $\mathbb{I}$. In the LDA platform, the *allowed data* is stored in temporal dataset. Unlike the approaches in [16], [24], and [25], which create a *"protected graph"*, we create temporal dataset in order to preserve the structural organization of the data without loosing any information. Once the *enforcement module* finishes processing and filtering of the data, the allowed data (if any) is transfered to the *ResultInterpreter* component, which transforms the results in the requested form based on the Intent $\mathbb{I}$. The *ResultInterpreter* additionally inserts the activated policies as an explanation of the result. When the operation is denied as a whole, the *enforcement module* throws an exception, which is serialized by the *ResultInterpreter*.

The *enforcement module* behavior and the legitimate protection depend on the policy correctness. The *policy man-*

[8]https://developer.android.com/reference/android/content/Intent.html

agement module supports the design time policy correctness validation and provides conflict detection, standalone policy testing, policy protection per intent, and overall protected and unprotected data extraction. This module is valuable tool for the data owners and simplifies the policy maintenance process.

In order to support maintainability and correctness validation, we make an effort to formally define the LDA platform and policy language. The formal definitions enable the design of the transformation algorithms for standalone policy testing, conflict detection and protected/unprotected data extraction, all used in the *policy management module*.

Definition 1: ***Linked data authorization (LDA) platform*** *is represented by the tuple* $\langle \mathbb{D}, \mathbb{I}, \mathbb{P} \rangle$*, where* $\mathbb{D}$ *is the guarded data,* $\mathbb{I}$ *is the requester's Intent and* $\mathbb{P}$ *is the set of the policies defined in the platform.*

Definition 2: ***Guarded data*** $(\mathbb{D})$ *is a set of tuples* $\langle S, P, O, G, DS \rangle$ *that defines the RDF triples* $\langle S, P, O \rangle \in (I \cup B) \times (I \cup B) \times (I \cup B \cup L)$ *logically organized in graphs G, physically stored in datasets DS. I, B and L represent the sets of all Internationalized Resource Identifiers (IRI), blank nodes and literals, respectively.*

Definition 3: ***Intent*** $(\mathbb{I})$ *is a semantic representation of the requester's intention, composed of multiple RDF triples.*

Definition 4: ***Allowed data*** *is the largest subset of the guarded data* $\mathbb{D}$ *that is permitted for the intent* $\mathbb{I}$ *by the policies* $\mathbb{P}$*.*

A. POLICY LANGUAGE

A policy (Definition 5) specifies the affected data after the intended action is executed. The intended action may involve multiple operations that should be constrained by the policy. *READ* operation policies define the data that can be obtained from the guarded data $\mathbb{D}$, while the modification operation policies (*INSERT* and *DELETE*) define the allowed state of the guarded data $\mathbb{D}$ after the operation is executed. *MANAGE* operation policies describe whether the operation should be allowed or not. In this way, the policies are easier to comprehend, define and validate against the requirements. Depending on the operation, the activated policies define whether the action will be rejected, allowed or it will have restricted access to the allowed data.

Definition 5: ***Policy*** *is a tuple of the form* $\langle \epsilon, o, q, \varphi_i, \varphi_d, \rho \rangle$*, where:*

- $\epsilon \in \{allow(+), deny(-)\}$ *is the permission for interaction with the policy's protected data* $\mathbb{D}^{(\mathbb{I},p)}$
- $o \in \{READ, INSERT, DELETE, MANAGE\}$ *is the intended operation*
- *q is a tuple of four variables used to project the extracted variable bindings into protected data quads*[9] $\mathbb{D}^{(\mathbb{I},p)}$*.*
- φ_i *and* φ_d *are predicate functions used for variable extraction from the Intent* $\mathbb{I}$ *and the guarded data* $\mathbb{D}$*.*
- ρ *is a priority used for conflict resolution.*

[9]Quad is essentially a triple with additional element that denotes the graph in which it belongs. It is a four element tuple.

Definition 6: ***Predicate function*** $\varphi(t_1, \ldots, t_n)$ *is an n-ary function that returns true if the terms* $t_i \in T = I \cup B \cup L \cup \{nil\}$ *satisfies its predicates, where* $n = |vars(\varphi)|$*.* $vars(\varphi)$ *extracts the set of variables defined in the predicate function* φ*, while* $vars(\varphi, i)$ *returns the variable expected as i-th argument in the* φ *function.*

The protected data by a single policy depends on the Intent $\mathbb{I}$ and is a subset of the guarded data $\mathbb{D}$. Definition 7 describes how this data is obtained for a given policy p and Intent $\mathbb{I}$.

Definition 7: ***Policy protected data*** $\mathbb{D}^{(\mathbb{I},p)}$ *for Intent* $\mathbb{I}$ *and policy* $p = \langle \epsilon, o, q, \varphi_i, \varphi_d, \rho \rangle$ *is a set of quads obtained as a result of the expression:*

$\mathbb{D}^{(\mathbb{I},p)} = \pi(q, \sigma(\varphi_i, \mathbb{I}, \varphi_d, \mathbb{D}))$

Definition 8: ***Variable evaluation function*** $\sigma(\varphi_1, D_1, \varphi_2, D_2)$ *extracts a set of term tuples from* $D_1 \cup D_2$ *that satisfy the predicate functions* φ_1 *and* φ_2 *such that:*

$$
\begin{aligned}
&\sigma(\varphi_1, D_1, \varphi_2, D_2) \\
&\quad = \{(t_1, \ldots, t_{m+n}) \mid \\
&\qquad m = |vars(\varphi_1)| \ \wedge \ n = |vars(\varphi_2)| \\
&\qquad \wedge \varphi_1(t_1, \ldots, t_m) = true \ \wedge \ \varphi_2(t_{m+1}, \ldots, t_{m+n}) = true \\
&\qquad \wedge \forall i \in [1..m], \quad \forall j \in [m+1..m+n] \Rightarrow \\
&\qquad t_i \in D_1 \ \wedge \ t_j \in D_2 \\
&\qquad \wedge vars(\varphi_1, i) = vars(\varphi_2, j) \Rightarrow t_i = t_j \\
&\qquad \}.
\end{aligned}
$$

Definition 9: ***Variable projection function*** $\pi(v_d, v_s, t)$ *restricts the set of term tuples t that correspond to the variable names* v_s *to the tuples that correspond to the variable names* v_d *such that* $m = |v_d|$*,* $n = |v_s|$ *and:*

$$
\forall (t'_1, \ldots, t'_m) \in \pi(v_d, v_S, (t_1, \ldots, t_n)) \Rightarrow
$$

$$
\forall j \in [1..m] \Rightarrow t'_j = \begin{cases} t_i, & \exists v_d[j] = v_s[i] \\ nil, & otherwise \end{cases}
$$

According to Definition 7 and Definition 8, the protected data is obtained by first extracting the set of tuples $(t_1, \ldots, t_m)$ from the Intent $\mathbb{I}$ that satisfy the conditions in φ_i, the set of tuples $(t_{m+1}, \ldots, t_{m+n})$ from the guarded data $\mathbb{D}$ that satisfy the conditions in φ_d, and then joins them based on the expression $vars(\varphi_1, i) = vars(\varphi_2, j) \Rightarrow t_i = t_j$. Next, the projection π from Definition 9 reduces these results to four-element tuples that correspond to the variables of interest in q. An example that illustrates this formalism on a real case sample data is shown in §V.

1) POLICY COMBINATION

In every non-trivial system, there are multiple requirements represented by policies. In such case, it is possible that multiple policies are activated for the same Intent, and hence, their protected data needs to be combined.

The operator $\odot$ determines how the data from two policies is combined. When two policies have the same permission, their protected data is combined together with the union

operator $\cup$, preserving the permission. In the opposite case, the protected data by the second policy is removed from the protected data by the first policy using the set minus operator $\setminus$, keeping the permission of the first policy.

Definition 10: ***Protected data combination*** $\odot$ *is a binary, non-commutative operator which combines the protected data of two policies with the same operation* $p_1 = \langle \epsilon_1, o, q_1, \varphi_{i1}, \varphi_{d1}, \rho_1 \rangle$ *and* $p_2 = \langle \epsilon_2, o, q_2, \varphi_{i2}, \varphi_{d2}, \rho_2 \rangle$ *as follows:*

$$\langle \epsilon_1, \mathbb{D}^{(\mathbb{I},p_1)} \rangle \odot \langle \epsilon_2, \mathbb{D}^{(\mathbb{I},p_2)} \rangle = \begin{cases} \langle \epsilon_1, \mathbb{D}^{(\mathbb{I},p_1)} \cup \mathbb{D}^{(\mathbb{I},p_2)} \rangle, & \epsilon_1 = \epsilon_2 \\ \langle \epsilon_1, \mathbb{D}^{(\mathbb{I},p_1)} \setminus \mathbb{D}^{(\mathbb{I},p_2)} \rangle, & \epsilon_1 \neq \epsilon_2 \end{cases}$$

Definition 11: ***Allowed data extraction function*** $\alpha(\mathbb{I}, \mathbb{D}, \mathbb{P})$ *is a function that outputs the subset of the guarded data* $\mathbb{D}$ *that is allowed for a particular Intent* $\mathbb{I}$*, using the LDA platform configured policies* $\mathbb{P}$*, such that:*

$$\alpha = \alpha^{(k)}$$
$$\alpha^{(i)} = \alpha^{(i-1)} \odot \langle \epsilon_i, \mathbb{D}^{(\mathbb{I},p_i)} \rangle$$
$$\alpha^{(0)} = \begin{cases} \langle \epsilon_+, \emptyset \rangle, & \epsilon_1 = + \\ \langle \epsilon_+, \mathbb{D} \rangle, & \epsilon_1 = - \end{cases}$$

where

$$\mathbb{P} = \{p_i = \langle \epsilon_i, o_i, q_i, \varphi_{ii}, \varphi_{di}, \rho_i \rangle \mid i \in [1,k] \wedge \forall j \in [2,k] \Rightarrow \rho_{j-1} \leq \rho_j\}$$

Definition 11 states that the allowed data function α extracts all the data from the guarded data $\mathbb{D}$ that is available for a given Intent $\mathbb{I}$ and allowed by the entire set of LDA platform configured policies $\mathbb{P}$. The policy's protected data $\mathbb{D}^{(\mathbb{I},p)}$, on the contrary, contains only the quads that are allowed or denied by a single policy for a given Intent $\mathbb{I}$. The function α actually combines the protected data of every policy using the $\odot$ operator, considering the policy priorities ρ. The $\alpha = \alpha^{(k)}$ equation denotes that the allowed data for the Intent $\mathbb{I}$ will be the result obtained after the highest priority policy p_k is applied. The $\alpha^{(i)} = \alpha^{(i-1)} \odot \langle \epsilon_i, \mathbb{D}^{(\mathbb{I},p_i)} \rangle$ defines an ordered processing of the policies, because $\forall j \in [2,k] \Rightarrow \rho_{j-1} \leq \rho_j$. Since the operator $\odot$ always takes the permission from its first argument, $\alpha^{(0)}$ is used to force the allowed data as result of $\alpha^{(1)}$, while $\alpha^{(i)} = \alpha^{(i-1)} \odot \langle \epsilon_i, \mathbb{D}^{(\mathbb{I},p_i)} \rangle$ is used to propagate this permission to the final result. For example, if the lowest priority policy p_1 allows access, $\alpha^{(1)} = \langle \epsilon_+, \emptyset \cup \mathbb{D}^{(\mathbb{I},p_1)} \rangle = \langle \epsilon_+, \mathbb{D}^{(\mathbb{I},p_1)} \rangle$, i.e., nothing is allowed at the beginning, and the allowed data is extended with the next policy. In the opposite case, when p_1 denies access, $\alpha^{(1)} = \langle \epsilon_-, \mathbb{D} \setminus \mathbb{D}^{(\mathbb{I},p_1)} \rangle$, i.e., everything is allowed at the beginning, and the allowed data is reduced by the next policy.

2) POLICY LANGUAGE SYNTAX

The role of a policy language is to provide a syntax that will support the policy formalism. The policy language should be designed to correspond to the policy formalism, while simplifying the requirement transformation into policies. Our policy language extends the SPARQL syntax for policy definition. The SPARQL extension is used in order to make the policies more readable and understandable, without mixing more syntaxes, which is the case in [21], [22], and [25]. The large amount of SPARQL resources makes the core of our policy language easier to understand and learn. The additional parts introduced in our policy language, on the top of SPARQL syntax, are the *permission, operation, priority* and the *dataset* elements, which are simple and do not introduce much complexity.

```
policy:=permission operation selection priority datasets?;
permission:=('ALLOW' | 'DENY');
operation:=dataOp | manageOp;
selection:=where solutionModifier?
priority:='PRIORITY' decimal;
datasets:='DATASETS' iri+;
dataOp:=opKeyword '{' quad '}';
manageOp:='MANAGE';
where:=<#rWhereClause>;
solutionModifier:=<#rSolutionModifier>;
opKeyword:='READ'| 'INSERT'| 'DELETE'| 'MODIFY';
quad:=varOrIri varOrIri varOrIriOrLiteral varOrIri;
varOrIri:=var| iri;
varOrIriOrLiteral:=( var | iri | literal);
var:=<#rVar>;
iri:=<#riri>;
literal:=<#rRDFLiteral>;
decimal:=<#rDECIMAL>;
```

Listing 1. Policy language definition with first order logic using prologue syntax.

The first order logic definition of the SPARQL extension in our language[10] is shown in Listing 1. The *permission* element defines whether the policy allows or denies the *operation* and corresponds to the ϵ element of the policy formalism in Definition 5. The *operation* element is composed of one of the operation keywords *opKeyword* that defines the operation o, optionally followed by a *quad* pattern that specifies the projection variables q. Although our formalism does not explicitly support *MODIFY* operations, in order to provide better maintainability, we added this operation in our language for convenience. Nevertheless, the *MODIFY* operation policies are formally represented by separate *INSERT* and *DELETE* operation policies.

Our policy language takes advantage of the SPARQL expressiveness for data selection which is used to specify the policy protected data $\mathbb{D}^{(\mathbb{I},p)}$ in Definition 5. The policy language *selection* part is a valid SPARQL element that outputs variable bindings and corresponds to the $\sigma(\varphi_i, \mathbb{I}, \varphi_d, \mathbb{D})$ function. The predicate function φ_i is specified as SPARQL *GRAPH* element,[11] while φ_d is specified as SPARQL group

[10]The reused parts from the original SPARQL specification are referenced by their URL. The <#something> elements from the definition should be replaced with <https://www.w3.org/TR/sparql11-query/#something>.

[11]https://www.w3.org/TR/rdf-sparql-query/#rGraphGraphPattern

graph pattern.[12] They are both used to extract variable bindings: the former from the implicitly managed named graph that represents the Intent $\mathbb{I}$, while the later, from the dataset against which the query is executed $\mathbb{D}$. The policy language *quad* pattern defines the four variables of interest from the solutions obtained by the σ function. The LDA platform uses these solutions to construct a set of quads that will represent the protected data quads $\mathbb{D}^{(\mathbb{I},p)}$. In this way, the administrators can specify an arbitrary resource, triple, quad or graph that should be protected and test the selection against their guarded data $\mathbb{D}$.

The *priority* element is used for conflict resolution. We use this approach since it is closest to the way people resolve the conflicts in the requirements. They either add another meta-policy that has higher priority compared to the others, or just tell that one of the requirement is more important than the others. Both cases can be modeled with policy priorities where the policies with higher priority are overriding the results of those with lower priority.

One of the key features of the semantic data is its interoperability and re-usability. In order to follow this trend, our policy language is capable of defining policies that can protect multiple datasets through the *datasets* element. This feature encourages distributed data storage and enables definition of policies that will protect this data. The datasets are represented by their dereferencable IRI, which is a downside from a maintainability point of view because all IRIs should be manually provided. The absence of this element means that the policy applies to all datasets.

B. ENFORCEMENT MODULE

The goal of the *Enforcement module* is to ensure interaction only with the allowed data for a given intent. It uses *partial data filtering strategy* [23] and generates temporal dataset against which executes the operation. The operation determines the behavior of the *Enforcement module*. When read operation is executed, the results are either filtered or conditionally returned. Insertion of new data and removal of existing data requires validation if the data is allowed for the given Intent. The allowed data extracted with the function α is always constructed from the policies that protect the given operation.

Depending on the Intent and the data sensitivity, the operations may be executed against the allowed data extracted with the function α from Definition 11 or they can be allowed or denied conditionally. Based on the action's configuration, in the process of conversion of available evidences into an Intent $\mathbb{I}$, the *IntentProvider* embeds the information weather partial results are allowed or not. When partial results are not allowed, the LDA platform throws an exception when some part of the requested data is denied, i.e., it conditionally rejects the requested data without filtering it.

[12]https://www.w3.org/TR/rdf-sparql-query/#rGroupGraphPattern

1) POLICY ALIGNMENT

The policies defined in the LDA platform can use arbitrary variable names. The LDA platform aligns the policies by unifying the variable names that refer to the same resources, which is useful in the processes of policy combination, conflict detection and overall data coverage estimation. The policy alignment starts by renaming all variables in the policies with globally unique variable names. Then, the policies' *quad* patterns q are aligned, providing that the same variables are used for the protected quads. We use the quad pattern $q_0 = (?s, ?p, ?o, ?g)$ to unify all policies' projections q. In this process, each policy variable that occurs in the *quad* element is replaced in the policy's predicate functions φ_i and φ_d with the corresponding variable from q_0.

Next, the triple patters from the predicate functions φ_i are aligned against each other. The variables that occur at the same position in the quad patterns that are ''equivalent'' are aligned by replacing the corresponding variables. The equivalent quad patterns are either the same or differ only in the variable names at the corresponding triple's positions.

This step is executed for each pair of policies, starting from the policies with the lowest priority.

2) ALLOWED DATA EXTRACTION

In the LDA platform, the *allowed data* is stored in temporal dataset since it is composed of multiple quads returned by the policies' protected data $\mathbb{D}^{(\mathbb{I},p)}$. Thus, we achieve execution of the requested queries in their native form, without modification. In the cases when a *''temporal graph''* is used [16], [24], [25], the queries containing GRAPH element should be transformed in order to obtain the permitted results, usually by removing the GRAPH elements in the query. This query modification looses information and often may produce incorrect results.

Algorithm 1 describes the implementation of the allowed data extraction function α from Definition 11 and the creation of the temporal dataset DS_{tmp}. The policies in $\mathbb{P}$ are aligned as described in §IV-B1 and sorted by their priority in ascending order. Since the SPARQL query language supports combining the elements using *UNION* and *MINUS* operations, the allowed data α is constructed with a single query.

According to Algorithm 1, the Intent is first registered (line 1) by inserting the Intent's data $\mathbb{I}$ in a newly created named graph *<http://intent/{id}>*, where the *{id}* is sequentially generated in order to enable concurrent Intent processing. The *registerIntent*($\mathbb{I}$, $\mathbb{D}$, $\mathbb{P}$) additionally replaces the <http://intent> string with the newly created graph name in every policy from $\mathbb{P}$ and returns a set of modified policies $\mathbb{P}'$. Next, the SPARQL's *WHERE* expression W is initialized depending on the lowest priority policy permission $p_1.permission$. The initialization of α_0 from Definition 11 (lines 2-5) uses the quad pattern φ_0 = *{ ?s ?p ?o ?g }*, which matches every quad from the guarded data $\mathbb{D}$, and $\varphi_\emptyset$, which is an empty predicate function that has no variables and does not select any data. Afterwards, all policies from $\mathbb{P}'$ are

Algorithm 1 Generating Temporal Dataset

Data: $\mathbb{I}$, $\mathbb{D}$, $\mathbb{P}$
Result: DS_{tmp}
1 $\mathbb{P}'$:=registerIntent($\mathbb{I}$, $\mathbb{D}$, $\mathbb{P}$);
2 **if** $p_1.permission = ALLOW$ **then**
3 | W:= $\{\varphi_{\emptyset}\}$;
4 **else**
5 | W:=$\{\varphi_{q_0}\}$;
6 **foreach** p *in* $\mathbb{P}'$ **do**
7 | **if** $p.permission = ALLOW$ **then**
8 | | W:=W UNION $\{p.WHERE\}$;
9 | **else**
10 | | W:=W MINUS $\{p.WHERE\}$;
11 **end**
12 varMappings:=execute(W, $\mathbb{D}$);
13 DS_{tmp}:=asQuads(varMappings, q_0);
14 **return** DS_{tmp}.

processed (lines 6-11). When a policy has allow permission, the content of its *WHERE* element[13] is appended using the *UNION* operation (line 8). In the opposite case, the *MINUS* operation is used (line 10). After all policies from $\mathbb{P}'$ are processed, the resulting *WHERE* element is executed against the guarded data $\mathbb{D}$ to extract the variable mappings (line 12). The extracted variable mappings for q_0 are transformed into quads in order to form the resulting temporal dataset DS_{tmp} (line 13).

After the temporal dataset is created, all operations are executed against it. Since this dataset contains only the allowed data, it provides implicit security, meaning that the queries do not interact with the data which is not allowed.

3) READ OPERATION PROTECTION

The READ operation is used to extract data of interest that satisfies certain criteria, specified by a query $Q(D)$ that can be executed against multiple datasets D. However, when requester authorization is required, the read operation additionally depends on the Intent $\mathbb{I}$ and on the configured policies $\mathbb{P}$ in order to determine the allowed data.

Definition 12 formalizes the outcome of the read operations, showing that the allowed data for the query should be obtained by its execution against the allowed data. The *conditional* equation in Definition 12 refers to the scenario where no data filtering is involved. This protection may be required when result consistency is needed. In this case, the read operation is allowed only if the protected results $R(\mathbb{I}, Q)$ are the same as the query results executed against the original data.

Definition 12 (Read Protection):

a) partial: $R(\mathbb{I}, Q) := Q(\alpha_{read}(\mathbb{I}, \mathbb{D}, \mathbb{P}))$
b) conditional: $Q(D) = R(\mathbb{I}, \mathbb{Q})$

[13] $p.WHERE$ denotes the content of the policys's *WHERE* element that represents $\sigma(\varphi_i, \mathbb{I}, \varphi_d, \mathbb{D})$.

This approach allows implicit protection even when the requester is aware of the data structure and is able to execute multiple probing queries with malicious intention. The correctness of Definition 12 is confirmed in [15], where the allowed data is used as baseline for proving that a query rewriting algorithm is *secure*.

Our LDA platform executes each query against the temporal dataset that represents the allowed data for reading α_{read}, which is constructed using only the policies with read operation.

4) DELETE OPERATION PROTECTION

The delete operation usually requests a set of quads to be deleted from the existing data, defined directly or with a query that selects the quads for removal. As Definition 13 shows, only the permitted data should be deleted in the partial deletion scenario. Since the data is already present in the dataset, the allowed data function α_{delete} will return the existing quads that are allowed for removing, and the intersection with the requested one gives what can be deleted after the enforcement process. When the data consistency should be preserved, which is the most often case, all requested data must be permitted.

Definition 13 (Delete Protection):

a) partial: $D(\mathbb{I}, q_d) := q_d \cap \alpha_{delete}(\mathbb{I}, \mathbb{D}, \mathbb{P})$
b) conditional: $q_d = D(\mathbb{I}, q_d)$

For each delete operation, our LDA platform checks if the quads are present in the temporal dataset constructed from the delete operation policies. If partial removal is allowed, only the quads that are found in this dataset are removed. In cases when SPARQL query is used to define the data that should be removed, this query is first executed against the permitted dataset for the read operations, and the resulting quads are filtered in the permitted dataset for deletion.

5) INSERT OPERATION PROTECTION

The *INSERT* operations usually modify the state of the dataset and receive a set of quads to be created (d_i) in addition to the Intent. Since most of the platforms have some consistency assumptions, the conditional protection will be the most common way of inserting data, where all the quads requested for creation should be approved. Such an example are the relational databases that constrain the data with their model. However, there also exist platforms with looser constrains that may allow partial data entry. In this case, the partial data allowed for insertion is $I(\mathbb{I}, q_i)$ from Definition 14.

Definition 14 (Insert Protection):

a) partial: $I(\mathbb{I}, q_i) := q_i \cap \alpha_{insert}(\mathbb{I}, \mathbb{D} \cup q_i, \mathbb{P})$
b) conditional: $q_i = I(\mathbb{I}, q_i)$

Since the policies describe the final state of the dataset after the interaction is executed, the permitted data function α_{insert} selects the allowed consequences of the insert operation. This data is used to select which of the requested quads for insertion are allowed with $I(\mathbb{I}, q_c)$. When data consistency is required, the conditional equation in Definition 14 shows that

the operation will be allowed only when all requested quads are permitted for insertion.

Our LDA platform first inserts the requested quads for creation q_i in the original dataset, and then, constructs the allowed data for insertion. Next, every quad from q_i that is not present in the allowed dataset for insertion is removed from the original dataset. The entire process is executed in a single transaction.

6) MANAGE OPERATION PROTECTION

The atomic managing operations are interacting with graph arguments embedded in the Intent as operation parameters. Partially allowing this kind of operations is impossible since they often require graph creation, deletion, coping or moving, which are either allowed or denied. The policies are processed starting from the highest priority towards the lowest. The permission of the first policy (which is with highest priority) that will return result is considered as final. The query executed for each policy is constructed with the transformation T_{ask} (shown in Listing 3) and it returns whether the policy should be activated or not. Since the business actions are also described through the Intent, the same procedure is used for their protection.

C. AUTHORIZATION PRINCIPLES SUPPORT IN THE LDA PLATFORM

The LDA platform is designed to support all authorization design principles defined in §II. Table 2 summarizes the components and features that support the *Flexibility, Maintainability, Correctness validation* and *Understandability* principles. The policy language is designed to cover all *flexibility* aspects discussed in §II. In order to increase the *understandability*, our policy language is based on the W3C standardized SPARQL language, and hence, the policies can be easily transformed into SPARQL queries that can be executed against the guarded data $\mathbb{D}$, which simplifies their *maintainability*. The *IntentProvider* component provides dynamic context representation, which is important part of the *flexibility* principle. The *Enforcement module* uses temporal datasets which ensures correct protection [15] and provides implicit security. The policy transformation algorithms denoted as *Policy to SPARQL*, *Policy coverage*, *Minimal Intent Binding*, *Conflict detection* and *Protected/Unprotected data* are employed in the *Policy management module* and can be executed in design phase. They simplify the maintenance process and can be used to validate that the guarded data is properly protected.

1) POLICY TO SPARQL

With this transformation algorithm, each policy becomes a regular SPARQL query that can be executed for a given Intent $\mathbb{I}$. The resulting queries are used in the *Enforcement Module* during the guarded data protection. Additionally, these queries enable performing automated policy tests through standalone policy testing.

TABLE 2. Authorization design principles support in the LDA platform.

LDA platform component/feature	Flex.	Maint.	Corr.	Und.
Policy language	✓	✓		✓
Intent provider	✓			
Enforcement module			✓	
Policy management module		✓	✓	
Policy to SPARQL		✓	✓	
Policy coverage		✓	✓	
Minimal Intent Binding		✓	✓	
Conflict detection		✓	✓	
Protected/Unprotected data		✓	✓	

```
SELECT q
WHERE {
 φ_i .
 φ_d
}
```

Listing 2. T_{select}: Policy transformation into SPARQL SELECT query.

```
ASK
WHERE {
 φ_i.
 φ_d
}
```

Listing 3. T_{ask}: Manage policy transformation into SPARQL ASK query.

The transformation in Listing 2 applies to the *READ, INSERT, DELETE* and *MODIFY* operation policies, while the transformation in Listing 3 applies to the *MANAGE* operation policies. With the first transformation, T_{select}, the policy becomes a regular SPARQL SELECT query that first extracts the variable bindings from the Intent graph that meet the φ_i conditions, and then, joins them with the solutions for φ_d. The variable of interest are returned with the SELECT q part. We rely on the SPARQL's algebra [37], which is similar to the relational algebra [38], as an implementation for the variable extraction σ and the projection function π.

Similarly, the transformation T_{ask} shows how the *MANAGE* operation policies can be transformed into SPARQL ASK queries that answers whether there is matching data for the selection conditions.

2) POLICY COVERAGE

When the data owner is interested in the overall data protected by a policy, no mater what the Intent is, it can use the query in Listing 4 to determine its coverage. The policy coverage is independent on the Intent and returns a subset from the guarded data $\mathbb{D}$ that satisfies the variable predicate function φ_d. Formally, the policy coverage can be represented as $\pi(q, vars(\varphi_d), \sigma(\varphi_\emptyset, \emptyset, \varphi_d, \mathbb{D}))$, where $\varphi_\emptyset$ is an empty predicate function that has no variables ($vars(\varphi_\emptyset) = \emptyset$) and always evaluates to *true*.

This transformation is very important for the policy design

```
SELECT q
WHERE {
  φ_d
}
```

Listing 4. $T_{policy\_coverage}$: **All the data protected by a policy.**

```
SELECT DISTINCT vars(φ_i) ∩ vars(φ_d)
WHERE {
  φ_d
}
```

Listing 5. **Minimal intent binding extraction.**

process since it provides a preview of the data covered by the policy and enables the policy designer to validate if all the data of interest is included.

3) MINIMAL INTENT BINDING EXTRACTION

Constructing an Intent that will activate the policy is a straight forward task because, if each variable from the policy's φ_i element is replaced by an arbitrary resource, the resulting triples will form an Intent that will activate the policy. However, this Intent does not guarantee that the policy will protect any data $\mathbb{D}^{(\mathbb{I},p)}$, so we have to carefully choose the resources. Finding an Intent that will result in non-empty protected data $\mathbb{D}^{(\mathbb{I},p)}$ may require writing of many additional queries. Therefore, the goal of Minimal Intent Binding extraction is to find all the Intents that result in non-empty protected data $\mathbb{D}^{(\mathbb{I},p)} \neq \emptyset$. This transformation algorithm is the base of conflict detection and protected data per minimal intent extraction algorithms.

Definition 15: ***Minimal intent binding*** *extracts the variable mappings that satisfy only the guarded data predicate function φ_d and selects only the variables that occur in both predicate functions (φ_i and φ_d):*

$$\pi(vars(\varphi_i) \cap vars(\varphi_d), vars(\varphi_d), \sigma(\varphi_\emptyset, \emptyset, \varphi_d, \mathbb{D}))$$

The Minimal Intent Binding extracts only the variables from φ_i that influence the protected data extraction, i.e., the variables that occur both in φ_i and φ_d. The variables that occur only in the predicate function φ_i are not considered because they only contribute to policy activation, but do not constrain the protected data $\mathbb{D}^{(\mathbb{I},p)}$. Therefore, we use the empty predicate function $\varphi_\emptyset$ so that σ extracts only the solutions that satisfy φ_d. The Minimal Intent Binding restricts the variables of interest to $vars(\varphi_i) \cap vars(\varphi_d)$. Listing 5 presents the structure of the SPARQL query that extracts all Minimal Intent Bindings.

This transformation can help the data owners to choose appropriate Intents for policy testing. All the possible Intents can be obtained by replacing the variables $vars(\varphi_i) \cap vars(\varphi_d)$ in every triple from φ_i with the corresponding Minimal Intent Bindings solutions, whereas the variables $vars(\varphi_i) \setminus vars(\varphi_d)$ are replaced with blank nodes.

```
SELECT q ∪ vars(φ_i) ∩ vars(φ_d)
WHERE {
  φ_d
}
ORDER BY vars(φ_i) ∩ vars(φ_d)
```

Listing 6. **Policy's protected data per minimal intent binding.**

Moreover, the Minimal Intent Binding extraction can be used to detect anomalies in the policy definition. An anomaly can be detected when $vars(\varphi_i) \cap vars(\varphi_d) \neq \emptyset$ and the Minimal Intent Bindings extraction query does not return any results. In this case, the policy will never be activated, meaning that it is obsolete and should be reexamined.

Listing 6 shows a query that selects the protected data $\mathbb{D}^{(\mathbb{I},p)}$ for each Minimal Intent Binding represented by the variables from $vars(\varphi_i) \cap vars(\varphi_d)$. The policy's predicate function φ_d selects the data for all possible Intents, and the *ORDER BY* element brings together the protected data per Intent.

4) CONFLICT DETECTION

When allow and deny permissions are allowed in the policies, conflicts may occur. Conflict detection in design phase is not a trivial task since the policy's protected data $\mathbb{D}^{(\mathbb{I}^{(c)},p)}$ depends on the conflicting Intent $\mathbb{I}^{(c)}$, the content of which is not known in advance. Therefore, the conflict detection process should find the conflicting Intents $\mathbb{I}^{(c)}$ for which there are two policies with opposite permission that protect the same data. The output of the conflict detection process must be comprehensive in order to provide simpler conflict resolution. Definition 16 shows that two policy are in conflict when they have opposite permissions and protect the same data.

Definition 16: Two policies $p_1 = \langle \epsilon_1, o_1, q_1, \varphi_{i1}, \varphi_{d1}, \rho_1 \rangle$ and $p_2 = \langle \epsilon_2, o_2, q_2, \varphi_{i2}, \varphi_{d2}, \rho_2 \rangle$ are in conflict if $\epsilon_1 \neq \epsilon_2$ and there is a conflicting Intent $\mathbb{I}^{(c)}$ for which $\mathbb{D}^{(\mathbb{I}^{(c)},p_1)} \cap \mathbb{D}^{(\mathbb{I}^{(c)},p_2)} \neq \emptyset$.

Definition 17 shows that the conflict detection process should be able to extract the Minimal Intent Bindings for which the conflict occurs and to output them together with the conflicting data. In this way, the data owners can react in the design phase to prevent unwanted consequences through the conflict resolution process.

Definition 17: For a given guarded data $\mathbb{D}$, the conflict detection process accepts two aligned policies $p'_1 = \langle \epsilon_1, o_1, q_0, \varphi'_{i1}, \varphi'_{d1}, \rho_1 \rangle$ and $p'_2 = \langle \epsilon_2, o_2, q_0, \varphi'_{i2}, \varphi'_{d2}, \rho_2 \rangle$ and searches for Minimal Intent Bindings that form conflicting Intent $\mathbb{I}^{(c)}$ and conflicting data portion $\mathbb{D}^{(c)} := \pi(q_0, \sigma(\varphi'_{i1}, \mathbb{I}^{(c)}, \varphi'_{d1}, \mathbb{D})) \cap \pi(q_0, \sigma(\varphi'_{i2}, \mathbb{I}^{(c)}, \varphi'_{d2}, \mathbb{D}))$ that satisfies $\mathbb{D}^{(c)} \neq \emptyset$.

When two policies are in conflict, Definition 16 implies that there should exist a conflicting Intent $\mathbb{I}^{(c)}$ that will activate them both i.e. $\mathbb{I}^{(c)}$ satisfies both φ'_{i1} and φ'_{i2}, which leads to $\sigma(\varphi'_{i1}, \mathbb{I}^{(c)}, \varphi'_{i2}, \mathbb{I}^{(c)}) \neq \emptyset$. The variables extracted by the previous expression are $vars(\varphi'_{i2}) \cup vars(\varphi'_{i2})$. Since the

SELECT
$q_0 \cup vars(\varphi'_{i1}) \cap vars(\varphi'_{d1}) \cup vars(\varphi'_{i2}) \cap vars(\varphi'_{d2})$
WHERE {
φ'_{d1} .
φ'_{d2}
}
GROUP BY
$vars(\varphi'_{i1}) \cap vars(\varphi'_{d1}) \cup vars(\varphi'_{i2}) \cap vars(\varphi'_{d2})$

Listing 7. Conflict detection transformation.

SELECT q_0
WHERE {
{ φ'_{d1} } UNION
...
UNION { φ'_{dk} }
}

Listing 8. Overall protected data.

Minimal Intent Bindings for a single policy are represented by the variables $vars(\varphi_i) \cap vars(\varphi_d)$, the conflicting Minimal Intent Bindings are represented by $vars(\varphi'_{i1}) \cap vars(\varphi'_{d1}) \cup vars(\varphi'_{i2}) \cap vars(\varphi'_{d2})$. Additionally, as the policies p'_1 and p'_2 are aligned, their protected data is represented by the variables in q_0. Listing 7 shows the query that selects $\pi(q_0 \cup vars(\varphi'_{i1}) \cap vars(\varphi'_{d1}) \cup vars(\varphi'_{i2}) \cap vars(\varphi'_{d2}), vars(\varphi'_{d1}) \cup vars(\varphi'_{d2}), \sigma(\varphi'_{d1}, \mathbb{D}, \varphi'_{d2}, \mathbb{D}))$. The grouping gathers together the conflicting data $\mathbb{D}^{(c)}$ for the same conflicting Minimal Intent Binding $\mathbb{I}^{(c)}$.

5) PROTECTED/UNPROTECTED DATA

In §IV-C2, we showed a transformation that obtains the data covered by the policy. This transformation can be generalized to obtain the overall protected and unprotected data in the platform. Listing 8 shows a query that returns the overall protected data in the platform. All aligned policies $p'_i \in \mathbb{P}$ ($i \in [1..k]$, $k = |\mathbb{P}|$) are combined together using the SPARQL *UNION* element.

A query that obtains overall unprotected data is shown in Listing 9, where φ_{q_0} is a predicate function that uses the variables from q_0 in order to denote the selection of every quad from the guarded data $\mathbb{D}$. This query removes the protected data selected in Listing 8 from the guarded data $\mathbb{D}$ using the SPARQL's MINUS element. The result of this query represents an important tool for validation of the authorization completeness - when there is unprotected data, the data owners will be alerted in the very policy design phase.

V. PLATFORM IMPLEMENTATION AND VALIDATION

In order to validate the implementation of the LDA platform, we developed a fully functional prototype implementation. Additionally, we developed secured query execution

SELECT q_0
WHERE {
{ φ_{q_0} }
MINUS
{
{ φ'_{d1} } UNION
...
UNION { φ'_{dk} }
}
}

Listing 9. Overall unprotected data.

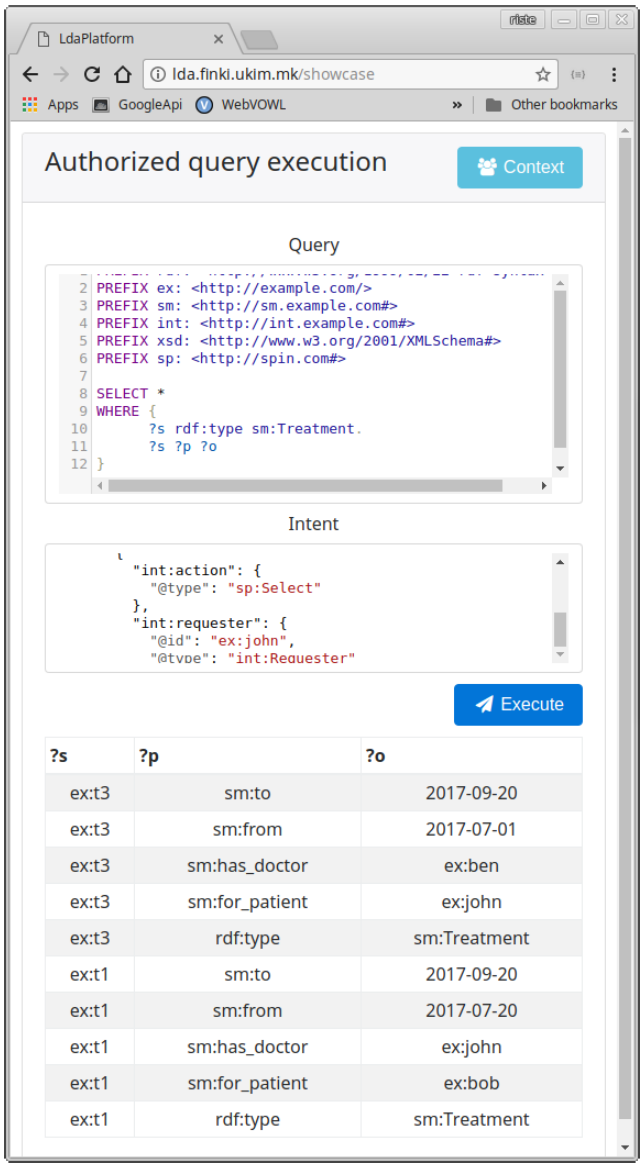

Figure 4. LDA platform secure query execution.

console (Figure 4)[14] for interactive testing of the platform. This interactive console provides submission of an arbitrary

[14]http://lda.finki.ukim.mk

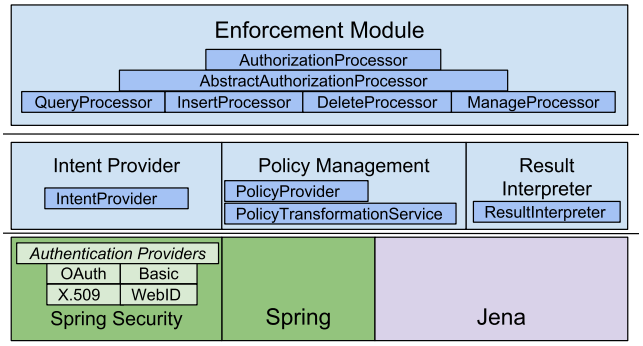

Figure 5. LDA technology implementation stack.

query against the LDA platform guarded data in a user configurable context. The *Context* button in Figure 4 opens a modal window, where the user specifies the context evidences in Resource Description Format (RDF) format.[15] The data from the console is sent to the *IntentProvider* component that parses this data, creates an Intent and sends the Intent to the *enforcement* module, which ensures interaction with the allowed data only. In case of a *read* query, the *Results Interpreter* serializes the allowed results and sends them back to the user. Additionally, in order to support and simplify the policy administration, we developed a user interface that exposes the functionalities of the *policy management module* (Figure 6).

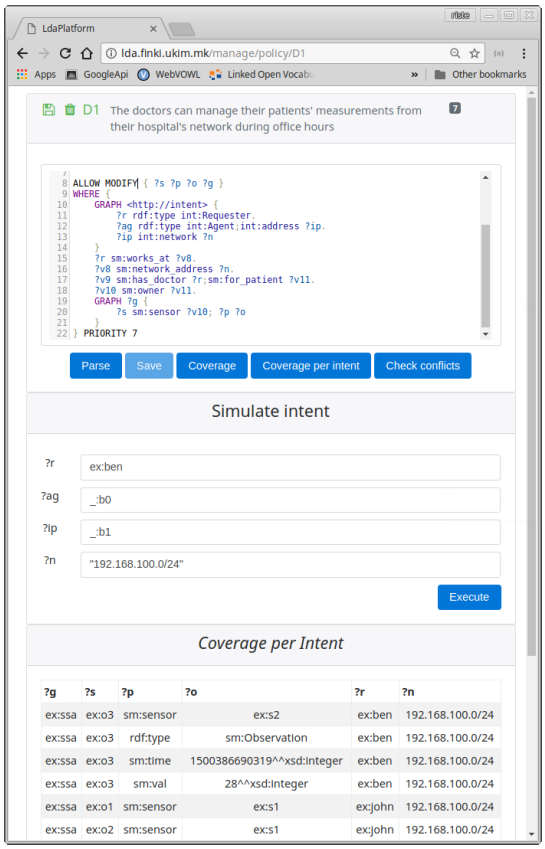

Figure 6. Policy management module.

A. TECHNICAL IMPLEMENTATION DETAILS

The main challenge of the LDA platform is the design of an extendable and modular architecture independent on the authentication mechanism, capable of flexible authorization and security policy management. In order to meet the modularity requirement, the LDA platform utilizes the Spring Framework, as shown in Figure 5. All semantic operations are implemented using the Apache Jena Library [39], which is mainly used in the authorization enforcement and policy management processes.

The second layer in Figure 5 displays the modules that support the *enforcement* module which runs the authorization algorithms. The *Intent provider* module has a key role in the context aware authorization. The current implementation of the *Intent provider* module relies on the state of the art Spring Security [40] library which provides multiple authentication mechanisms. The current *IntentProvider* implementation uses the WebID authentication protocol [41], [42]. However, due to the configurability that the Spring Security library provides, the *IntentProvider* component can be easily modified to use another *AuthenticationProvider* and to extract the Intent based on other authentication protocol, such as OAuth protocol,[16] X509 certificates,[17] Basic Authentication[18] and many others. The *policy management* module enables the maintenance process and correctness validation. The *PolicyProvider* returns the policies based on the operation being executed and the dataset it is intended for. The *PolicyTransformationService* is used to extract the parts of the policies using the Jena API and it implements the transformation algorithms described in §IV-C. The *result interpreter* module serializes the results in the requested format and appends the query used to create the temporal graph as an explanation.

Figure 5 also shows that the *enforcement* module exposes the *AuthorizationProcessor* interface, which accepts the submitted query that should be protected, the *Intent*, the dataset against which the query should be executed and an implementation of the *ResultInterpreter* interface. The *AbstractAuthorizationProcessor* first inserts the *Intent* in the dataset as a separate named graph, than delegates the execution to the specific operation implementation, and eventually, removes the named graph for *Intent*. Since the *Dataset* implementations in Jena provide transactions, this process is executed in a single transaction. All operation implementations constructs in-memory temporal dataset, but this behavior can be changed with different *DatasetProvider* implementation.

[15] https://www.w3.org/TR/rdf11-concepts/
[16] https://tools.ietf.org/html/rfc6749
[17] https://tools.ietf.org/html/rfc5280
[18] https://tools.ietf.org/html/rfc2617

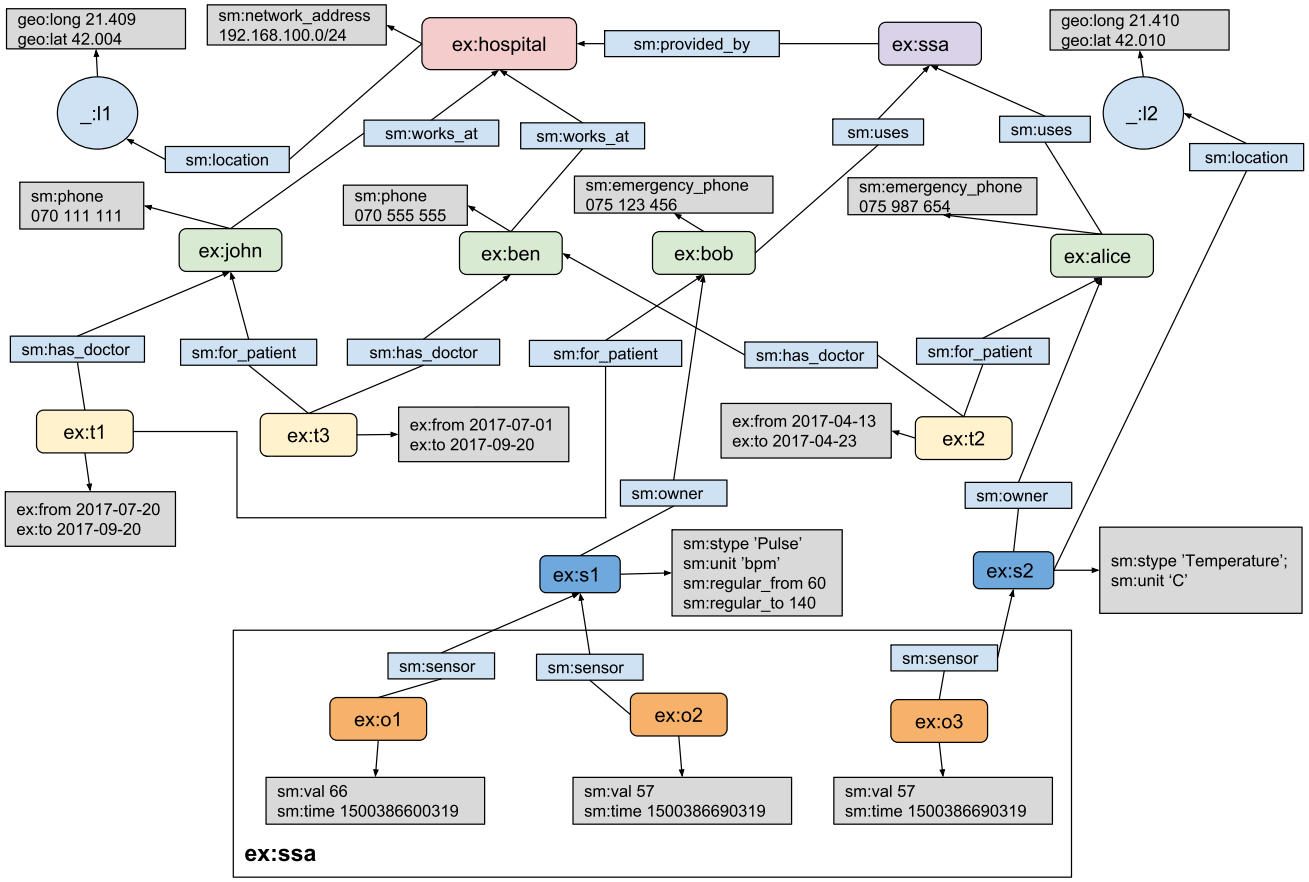

Figure 7. Example dataset.

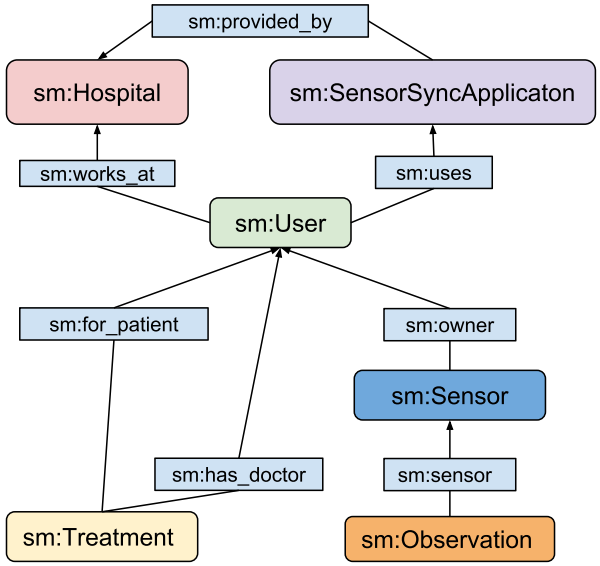

Figure 8. Ontology.

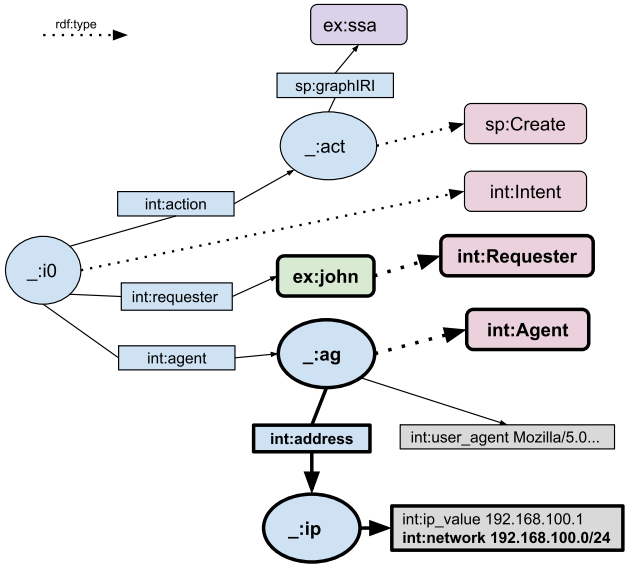

Figure 9. Intent example.

B. PLATFORM USE-CASE VALIDATION

In order to explain all details regarding LDA platform implementation and its capabilities, we designed a use case scenario that covers all important aspects of the LDA platform operation. Figure 7 shows a sample guarded data $\mathbb{D}$ that represents the popular mixing of data from smart devices with social and corporate data. It visualizes sensor observations that are fused with the users' personal data and with

a patient management system's data. Figure 8 shows the ontology that models the classes and the properties of the data in Figure 7. Our example covers one hospital, *ex:hospital*, with a *sm:network_address "194.168.100.0/24"*, located at the location *_:l1*, described with the Geo-names ontology.[19] Although, the ontology in Figure 8 shows that the hospitals may provide multiple applications for sensor synchronization, *ex:hospital* provides only the application *ex:ssa*. The application *ex:ssa* is fed by sensor data from the sensors *ex:s1* and *ex:s2* owned by its users *ex:bob* and *ex:alice*, respectively. These users are hospital's patients with treatments *ex:t1* and *ex:t2* conducted by the doctors *ex:john* and *ex:ben*, respectively. The treatment *ex:t3* shows that *ex:john* is a patient of *ex:ben*. The fact that *ex:john* is a doctor and patient depending on the occasion, enables modeling different levels of separation of duty, as discussed in [43]. The locations *_:l1* and *_:l2*, and the hospital's network IP address enable defining geo-spatial contextual requirements, while the duration of the treatments provides temporal contextual requirements.

Figure 9 shows an example Intent *_:i*, where a requester *ex:john* tries to create a graph *ex:ssa* using a software agent *_:ag* through an intended action *_:act*. The SPIN ontology[20] models the SPARQL syntax and is used to represent the action in this example. It describes that the intended action is a SPARQL CREATE query (*sp:Create*) and the property *sp:graphIRI* represents the graph intended for creation. The *int:* prefix models the classes and the properties used to indicate the expected data structure for the *Intent*. The *int:requester* property and the class *int:Requester* specify who the requester is. The resource *_:ag* represents the software agent used to interact with the LDA platform.

1) POLICY's PROTECTED DATA $\mathbb{D}^{(\mathbb{I},p)}$

Listing 10 shows an example policy that allows the doctors to access their patients' observations from the hospital network only. In this example, the body of the *GRAPH <http://intent>* element (lines 5-8) corresponds to the predicate function φ_i that validates the policy applicability for the Intent. In our case, we are using separate graph for maintaining the Intent's data. The policy utilizes the *GRAPH <http://intent>* element (line 4) to validate that the Intent meets the activation requirements. The policy is activated and considered in the enforcement process when the Intent binding function φ_i extracts the required resources from the Intent. The data selection function φ_d is represented by the content of the lines 10-19. We use SPARQL's *WHERE* element (lines 4-19) since its processing corresponds to the variable evaluation function $\sigma(\varphi_i, \mathbb{I}, \varphi_d, \mathbb{D})$ and returns the evaluated variable bindings. These results are the variable evaluations that match the policy's predicate functions, but they do not define what should be protected by the policy. The projection function π is used to create the quads defined with the variables in q that are protected by the policy from the variable bindings

[19]http://www.w3.org/2003/01/geo/wgs84_pos#
[20]http://spinrdf.org/sp#

```
1 ALLOW READ {
2 ?r ?p ?o ?app
3 } WHERE {
4  GRAPH <http://intent> {
5   ?doc a int:Requester.
6   ?ag a int:Agent.
7   ?ag int:address ?ip.
8   ?ip int:network ?n
9  }
10 GRAPH ?app {
11  ?r a sm:Observation.
12  ?r sm:sensor ?s.
13  ?r ?p ?o
14 }
15 ?doc sm:works_at ?h.
16 ?s sm:owner ?pat.
17 ?t sm:for_patient ?pat.
18 ?t sm:has_doctor ?doc.
19 ?h sm:network_address ?n
20 }
21 PRIORITY 1
```

Listing 10. E1: Example policy.

returned by σ. The SPARQL's *SELECT* query form projects the desired variables and is suitable implementation of the projection function π. The quad pattern *?r ?p ?o ?app* (line 2) will construct the protected data $\mathbb{D}^{(\mathbb{I},p)}$.

Lets assume that the data in Figure 7 is protected only by the policy in Listing 10, and *ex:john* submits the Intent from Figure 9 to execute the *sp:Create* action that requires read operation to be executed before the graph creation. This scenario is valid since each action may require multiple operations to be executed for its completion. The Intent binding predicate function function φ_i is evaluated against the Intent's graph <http://intent> and separate binding is created for each resource combination that matches all triple patterns of φ_i (marked bold in Figure 9). In our case, $vars(\varphi_i)$ =*(?doc, ?ag, ?ip, ?n)* and only the variable binding *(?doc→ex:john, ?ag→_:ag, ?ip→_:ip, ?n→"192.168.100.0/24")* matches all patterns in φ_i when evaluated against the Intent graph $\mathbb{I}$. Similarly, the predicate function φ_d is used to evaluate the variable bindings shown in Table 3 (without the columns *?ag* and *?ip*) from the guarded data $\mathbb{D}$. The final variable bindings are obtained by joining the two results that have the same value for the variables that are common both in φ_i and φ_d, i.e., *?doc* and *?n*. The user *ex:john* is the only doctor that has a patient with observations (for the patient *ex:bob*) and works at the hospital *ex:hospital* with network IP address *"192.168.100.0/24"*, which is why all of the variable bindings evaluated for φ_d are in the final result in Table 3. These bindings contain the resources, literals and blank nodes that match all triple and quad patterns from the policy's *WHERE* element. The projection quad q is then used to construct the protected data shown in Figure 10. The first line of the policy in Listing 10 describes that the read operations for the data in Figure 10 will be allowed.

TABLE 3. Protected data selection binding.

?doc	?ag	?ip	?n	?s	?pat	?t	?h	?r	?p	?o	?app
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o1	rdf:type	sm:Observation	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o1	sm:sensor	ex:s1	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o1	sm:time	1500386600319	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o1	sm:val	66	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o2	rdf:type	sm:Observation	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o2	sm:sensor	ex:s1	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o2	sm:time	1500386690319	ex:ssa
ex:john	_:ag	_:ip	192.168.100.0/24	ex:s1	ex:bob	ex:t1	ex:hospital	ex:o2	sm:val	57	ex:ssa

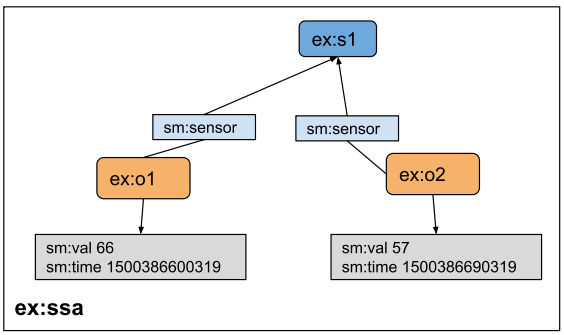

Figure 10. Protected data.

```
SELECT DISTINCT ?doc ?n
WHERE {
  GRAPH ?app {
    ?r a sm:Observation; sm:sensor ?s; ?p ?v
  }
  ?doc sm:works_at ?h.
  ?h sm:network_address ?n.
  ?t sm:has_doctor ?doc; sm:for_patient ?pat.
  ?s sm:owner ?pat.
}
```

Listing 11. Minimal Intent Bindings extraction example.

Once a policy is entered and saved, the *policy management module* (Figure 6) aligns it as described in §IV-B1. The "Parse" button parses the policy and creates a standalone testing query, as explained in §IV-C1. Additionally, during this action, the *policy management module* parses the variables from the Intent binding predicate function $vars(\varphi_i)$ and builds the "Simulate intent" form that can be used to simulate the policy's protected data for a given Intent variable bindings. The *policy management module* also shows the results of the data selection actions.

2) POLICY's MINIMAL INTENT BINDINGS

An example Intent for the *E1* policy in Listing 10 is the Intent $\mathbb{I}$={*ex:sam a int:Requester. _:ag a int:Agent; int:address _:ip. _:ip int:network "192.168.100.0/24"*}. However, when applied to our sample dataset, this Intent will not result in any protected data $\mathbb{D}^{(\mathbb{I},E1)}$ when executed against the guarded data $\mathbb{D}$ shown in Figure 7. In order to find a scenario with non-empty protected data, it is necessary to find the Minimal Intent Bindings that will activate the policy.

The policy's *E1* Intent binding predicate function φ_i contains the variables $vars(\varphi_i)$ =*(?doc, ?ag, ?ip, ?n)*, while its protected data predicate function φ_d contains the variables $vars(\varphi_d)$ =*(?app, ?r, ?s, ?p, ?o, ?doc, ?h, ?pat, ?t,?n)*. The intersection of these sets are the variables $vars(\varphi_i) \cap vars(\varphi_d)$ =*(?doc, ?n)*, which are used in the query in Listing 11 that extracts the Minimal Intent Binding from Definition 15.

If the query in Listing 11 is executed against our sample data, the only Minimal Intent Binding that is extracted is *(ex:john, "192.168.100.0/24")*. The Intents that will result in non-empty protected data $\mathbb{D}^{(\mathbb{I},E1)}$ can be synthesized from this Minimal Intent Binding. In this process, the policy's φ_i variables can be replaced with the resulting Minimal Intent Bindings, and the remaining variables can be replaced with blank nodes. If we apply this replacement in our example, we obtain that the Intent $\mathbb{I}$=*{ex:john a int:Requester. _:ag a int:Agent; int:address _:ip. _:ip int:network "192.168.100.0/24"}* is the only synthesized Intent that can activate the policy *E1* and protects the data $\mathbb{D}^{(\mathbb{I},E1)} \neq \emptyset$.

3) POLICY's COVERAGE

The transformation algorithm $T_{policy\_coverage}$ in Listing 4 produces the query in Listing 12. This query extracts the data shown in Figure 10 from the dataset in Figure 7. The results of the action "Coverage per Intent", obtained by transforming the policy according to Listing 6 and executing the obtained query can be seen in Figure 6. This query extracts the policy's protected data $\mathbb{D}^{(\mathbb{I},p)}$ per its Minimal Intent Bindings. In this way, the data owner can obtain, review and validate the protected data per Intent on a button click.

4) CONFLICT DETECTION

Lets assume that the security requirements from Table 6 should be implemented as policies that protect our sample data from Figure 7. The *policy management module* provides comparison of the currently inserted policy with the remaining policies defined in the platform, showing the conflicting Intents together with the conflicting data. The conflict detection process for the policies in Table 6 finds 3 conflicting policy pairs: A2 – P1, A2 – U2 and D1 – D2. The output

```
SELECT DISTINCT ?r ?p ?v ?app
WHERE {
  GRAPH ?app {
    ?r a sm:Observation; sm:sensor ?s; ?p ?v
  }
  ?doc sm:works_at ?h.
  ?h sm:network_address ?n.
  ?t sm:has_doctor ?doc; sm:for_patient ?pat.
  ?s sm:owner ?pat.
}
```

Listing 12. E1 protected data query.

TABLE 4. Conflict detection for A2 – P1.

?s	?p	?o	?g	?r
ex:ben	sm:phone	075 555 555		ex:alice
ex:john	sm:phone	070 111 111		ex:alice
ex:ben	sm:phone	075 555 555		ex:bob
ex:john	sm:phone	070 111 111		ex:bob
ex:ben	sm:phone	075 555 555		ex:john
ex:john	sm:phone	070 111 111		ex:john

TABLE 5. Read operation unprotected data.

?s	?p	?o	?g
_:b0	geo:lat	"42.004"8sd:double	
_:b0	geo:long	"21.409"8sd:double	
_:b1	geo:lat	"42.010"8sd:double	
_:b1	geo:long	"21.410"8sd:double	
ex:o1	rdf:type	sm:Observation	
ex:o1	sm:sensor	ex:s1	
ex:o1	sm:val	66	
ex:o1	sm:time	1500386600319	
ex:o2	rdf:type	sm:Observation	
ex:o2	sm:sensor	ex:s1	
ex:o2	sm:val	57	
ex:o2	sm:time	1500386690319	
ex:o3	rdf:type	sm:Observation	
ex:o3	sm:sensor	ex:s2	
ex:o3	sm:val	28	
ex:o3	sm:time	1500386690319	

of the conflict between A2 – P1 is shown in Table 4. This information enables the data owner to properly resolve the detected conflicts.

C. UNPROTECTED DATA EXTRACTION

A common ambiguity that arises in the authorization platforms is whether the unprotected data should be allowed or denied. Table 5 shows that the policies in Table 6 are not sufficient for complete protection of the sample dataset. This information can prevent exposure of private data (such as the observations *ex:o1, ex:o2* and *ex:o3*) or hiding data that is intended for public good (such as the location of the hospital *_:b0*). The unprotected data transformation alerts the data owner that additional policies should be created for complete data protection. The *policy management module* reveals this information on a button click, showing the unprotected data for each atomic operation. Although not shown in the example, there are even more unprotected quads in our example for the *INSERT* and *DELETE* operations (43 out of 59 quads).

VI. LDA PLATFORM EVALUATION

The main goal of our work is to build a flexible, maintainable and understandable LDA platform that covers all the design consideration described in §II. The *Flexibility* of the LDA platform compared to the related work is demonstrated by a set of diverse authorization requirements in Table 6. The requirements in this table are chosen to cover all *flexibility* design principles in respect to the use case scenario from §V. This table clearly shows that the LDA platform is the only one that fully covers all these requirements, i.e., all the *flexibility* design considerations. The policies for every single requirement, written in our policy language, and the data they protect are shown in Appendix.

The *context awareness* is present in the requirements *D1, D2* and *EM1*, since *D1* depends on the requester's agent IP address and on the accessing time, *D2* depends only on the accessing time and *EM1* is an emergency policy that depends on the data context. Even though the platforms [11], [13], [24] provide context aware policies, they only support read operations. Our LDA platform is unique in the support of context awareness for every operation since it provides complete *operation coverage* for the Linked Data operations. The *IntentProvider* component enables dynamic context representation through the Intent, while the *Enforcement Module* protects every operation in the provided contextual environment.

Another flexibility feature that makes our LDA platform distinguishable is the ability to define *aggregated data sharing* policy which is pointed out by the requirement *A3*. None of the reviewed authorization platforms supports this feature.

The LDA platform is also superior from maintainability point of view since it is the only one that provides one-to-one policy to requirement specification, conflict detection and protected and unprotected data extraction at the same time.

The extraction of the Minimal Intent Bindings described in §IV-C3 significantly reduces the number of required test cases by removing the ones that do not protect any data. The protected data per Minimal Intent Binding provides a preview of all possible outcomes for a policy, which is a significant information for policy correctness verification. The protected data per Minimal Intent Binding for each of the policies in Table 6 is shown in Appendix.

Apart from the uniqueness of our LDA platform, another contribution of our work is the *policy management module*. It is a powerful tool that considerably lowers the maintainability effort and time. It provides previewing of the results for versatile policy transformation algorithms on a button click, such as Minimal Intent Binding, conflict detection and extraction of the protected and unprotected data at design time. The results from the *policy management module* allow anomaly

TABLE 6. Protection requirements.

Requirement	[10]	[22]	[11]	[24]	[16]	[25]	[44]	[13]	[15]	[20]	LDA platform
A1: The hospital's and the application's properties are publicly available for everyone.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A2: Users' mobile and emergency phones are private.		✓		✓	✓	✓		✓	✓	✓	✓
A3: The average daily measurements from the sensors that are not sensitive (health) are public.											✓
U1: The users can access their own properties and the direct properties of the resources connected with them.			✓	✓		✓		✓		✓	✓
U2: The users can manage their name, phone and emergency phone or email.			✓	✓		✓		✓		✓	✓
P1: The patients can access everything about the doctors from *ex:hospital*.		✓	✓	✓		✓		✓	✓	✓	✓
D1: The doctors can modify their patients' measurements from their hospital's network during office hours.											✓
D2: The doctors can not modify the measurements out of their treatment timespan.											✓
TS1: Technical Staff can manage applications only for his/hers hospital.									✓		✓
SU1: The user ex:ben can generate reports.	✓		✓	✓							✓
EM1: The doctor can access their patients emergency phone number during abnormal measurements.			✓	✓				✓			✓

detection, while the flexibility of the proposed policy language enables writing policies that will resolve these security flaws.

The use of the W3C standardized SPARQL query language as a basis of our policy language simplifies the maintenance process by reducing the number of needed policies per security requirement. The large community and materials lower the adoption barrier of our language. There are already tools that enable non IT specialists to write SPARQL queries [45]–[47], which can be adopted for management of our policies with little effort and adjustments.

As the main goal of the LDA platform is to provide a flexible authorization of the datasets, we relay on the state of the art authentication protocols provided by the Spring library. The *Intent provider* module unites the authorization and authentication processes. It extracts the evidences provided by the authentication protocol and uses them to compose the *Intent* used for authorization in the *enforcement* module. The current implementation of the *Intent provider* module utilizes a Spring Security authentication provider for the WebID protocol, the most widely used authentication protocol in the Linked Data community. Depending on the use case, different protocol can be employed in order to extract the requester's properties and to secure their intents.

The LDA platform is robust in terms of *Query Injection* because it does not accept parameterized queries which are commonly used for this type of attacks. In essence, the query injection is an act of replacing the expected parameters with another malicious query, which executes multiple queries per request. The LDA platform is designed to ensure that there is single query per Intent and to modify the query in a way that it can interact with the allowed data only.

Since the data protection depends on the policies, it is crucial to ensure that there is no attack that will alter the configured policies. The current implementation of the LDA platform is conservative in terms of policy administration, and even though it allows policy testing and management, only the person that has control to the LDA platform instance can publish the policies. Another policy administration option is to liberate this process by storing the policies in RDF format and to authorize their maintenance through LDA platform policies. In this scenario, the policy management requirements must have highest priority so that nobody is able to override the policies, i.e., it is impossible to create a policy with a higher priority. This challenge is solved by limiting the maximal priority of the newly added or modified policies to a predefined value.

One of the highest security risks in every platform comes from internal human mistakes. In order to overcome the configuration mistakes and oversights, the *policy management* module (Figure 6) provides tools for extensive policy correctness testing, as described in §V. These tools enable the data owners to ensure the appropriate protection and to prevent improper data exposure. Considering that the policies are used to construct a temporal dataset with the allowed data without any external influence, we can ensure that the testing results are relevant in production environment. The fact that the temporal dataset contains only the allowed data, the LDA platform prevents obtaining data using multiple probing queries, and therefore, offers implicit data protection, as explained in §IV-B3.

A. PERFORMANCE EVALUATION

According to [23], the enforcement architecture of the LDA platform uses *partial data filtering* strategy where a temporal dataset is created per request. This strategy has the advantage of providing implicit data protection since the denied

data is not considered at all during the Intent processing. However, the evaluations presented in [15] and [16] show that a performance price has to be payed in return to this convenience.

In order to evaluate the performance of the LDA platform, we conducted an experiment of a system configured with the policies from Appendix, data modeled with the ontology shown in Figure 8 and initial guarded data as shown in Figure 7. The goal of the experiment is to examine the dependency of the processing time on the allowed data size. In order to achieve this goal, we execute 70 evaluation rounds, where each following round increases the size of the allowed data by 10000 quads, representing 2500 resources of type *sm:Observation*. After each round, we evaluate different operations using 10 warm up cycles and 20 evaluation cycles. The largest evaluated dataset is composed of 700.000 triples. Bigger datasets are not considered since there authorization will require unacceptably long time. In the experiment, we evaluate the following operations:

- *Standard LDA authorized query:* executes selection of all allowed quads in the current round. The LDA platform is used to authorize the operation using all configured policies for temporal allowed dataset creation.
- *Query-centric LDA authorized query:* executes selection of all allowed quads in the current round. The LDA platform is used to authorize the operation using only the policies that may affect the query results.
- *Hard-coded authorization query:* executes selection of all allowed quads in the current round using hard-coded query template for authorization in order to obtain the same results as the previous operations.
- *LDA authorized insert and delete*: An authorized requester executes insertion and deletion of resources. The LDA platform is used to authorize the intention.
- *Hard-coded authorization insert and delete:* An authorized requester executes insertion and deletion of resources. Hard-coded rules for the applicable requirements are used to authorize the intention using *SPARQL ASK* queries. If all query results are positive, the insertion or the deletion is executed.

The most common authorization approach used for fine grained data protection is the use of hard-coded rules in the service layer because the developer is able to optimize the performance. Therefore, we consider the hard-coded authorization approach as an optimal protection approach in respect to performance.

Figure 11 shows the dependence of the *insert/delete* operations execution time on the allowed data size. The *hard-coded authorization insert/delete* operations have almost constant execution time because they execute a separate *SPARQL ASK* query for each policy before allowing operation execution. Therefore, the performance mainly depends on the number of queries executed, i.e., the number of policies. On the other hand, the *LDA authorized* operations have linear increase in the execution time since the LDA platform creates a temporal dataset for each intent. The deceleration trend in this case is due to the linear dependence of the dataset creation time on its size.

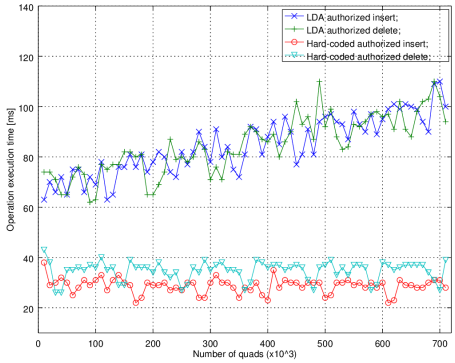

Figure 11. LDA modify performance results.

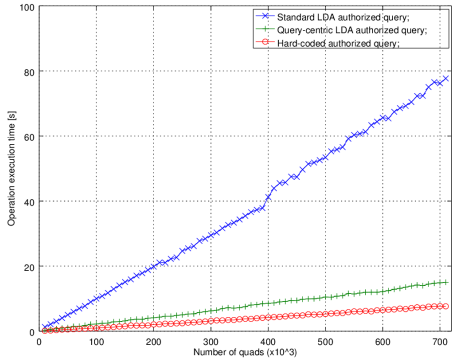

Figure 12. LDA query performance results.

Figure 12 shows that all query authorization operations have linear execution time dependence in respect to the allowed data size. However, the *standard LDA authorization query* is approximately an order of magnitude slower due to the complexity of the query for temporal dataset creation. Therefore, we optimized this process by using only the relevant policies for the query in the enforcement process. These policies are extracted using variable type inference from the quad patterns present in both the query and the policy. The selected policies are only those that have matching of their *projection quad pattern* with some of the query's quad patterns, considering the inferred variable types. This optimization results in fivefold performance improvement, which is near to the optimal performance, as shown with *query-centric LDA authorized query* results.

Even though the *hard-coded* authorization has superior performance, it is hard for testing and lacks flexibility and maintainability. Moreover, each requirement change requires code modification and additional testing. On the contrary, the LDA platform provides flexible requirement representation, simple maintenance and tools for policy validation and

testing. The only price that has to be paid for this convenience is partially sacrificing the performance. However, this price may be acceptable when a complete protection flexibility and maintainability is required.

VII. CONCLUSION

In this paper, we present a complete Linked Data Authorization platform on the top of a flexible policy language designed for Linked Data protection. The LDA platform offers full coverage of the *flexibility* authorization design principle, while the use of the W3C standardized SPARQL query language as a basis of the policy language increases the *understandability* and simplifies the *maintainability* process by reducing the number of needed policies per security requirement. The *policy management module* supports the *maintainability* and *correctness validation* principles in design time, while the *enforcement module* ensures the implicit data protection.

The LDA platform uses context dependent policies with a broad protection granularity, covering resources, triples, graphs, and datasets. The authorization context is modeled through the *Intent*, which is dynamically provided through the *Intent provider* module and contains all necessary requester and environment evidences, together with the intended action and its parameters. The *context awareness* enables designing emergency policies, while the flexible policy language is unique in its support of policies that can expose aggregated and impersonalized data that will serve the greater good.

The *policy management module* supports design time validation and significantly reduces the maintenance effort, time and cost. In this process, not only the owners can preview all the data protected by the policy, but they can also get suggestions for the possible test case scenarios that result in non-empty protected data. Moreover, the LDA platform enables conflict detection and resolution by using priorities. This paper also presents how the platform's unprotected data can be previewed in design time, which guides the data owners towards a complete data protection.

The LDA platform is implemented using the the widely accepted *Spring Framework* and *Apache Jena* library, and it has acceptable performance in comparison to the optimal hard-coded authorization. Nevertheless, in order to achieve better performance, we focus our future work on defining a suitable query rewriting algorithm for our policy language and introducing caching mechanisms for the temporal datasets.

With all these features in mind, the proposed LDA platform is a step forward towards wider acceptance of the Link Data as a common platform for heterogeneous data representation and interlinking. Furthermore, the *policy management module* can serve as a useful tool for design time protection validation even for platforms other than the LDA platform.

APPENDIX
EXAMPLE POLICIES

Protection requirements listed in Table VI are shown in this section.

```
ALLOW READ { ?s ?p ?o ?g }
WHERE {
 ?s ?p ?o .
 { {
   ?s a sm:Hospital
  } UNION {
   ?s a sm:SensorSyncApplicaton
  } }
} PRIORITY 1
```

Coverage per minimal intent:

?s	?p	?o	?g
ex:hospital	rdf:type	sm:Hospital	
ex:hospital	sm:network_address	192.168.100.0/24	
ex:hospital	sm:location	_:b0	
ex:ssa	rdf:type	sm:SensorSyncApplicaton	
ex:ssa	sm:provided_by	ex:hospital	

A1: The hospital's and the application's properties are publicly available for everyone:

```
DENY READ { ?s ?p ?o ?g }
WHERE {
 ?s a sm:User; ?p ?o.
 FILTER (?p=sm:phone || ?p=sm:emergency_phone)
} PRIORITY 3
```

Coverage per minimal intent:

?s	?p	?o	?g
ex:john	sm:phone	070 111 111	
ex:ben	sm:phone	075 555 555	
ex:bob	sm:emergency_phone	075 123 456	
ex:alice	sm:emergency_phone	075 987 654	

A2: User's mobile and emergency phones are private:

```
ALLOW READ {?s ?p ?o ?g}
WHERE {
 BIND(sm:avg_value as ?p)
 ?s sm:owner ?v3.
 { SELECT (AVG(?v1) as ?o) ?s
  WHERE {
   GRAPH ?g {
    ?v2 sm:sensor ?s; sm:val ?v1; sm:time ?v4
   }
  } group by ?s ceil(?v4/86400000)
 }
} PRIORITY 9
```

Coverage per minimal intent:

?s	?p	?o	?g
ex:s2	sm:avg_value	28	

A3: The average daily measurements from the sensors that are not for health are public:

```
ALLOW READ { ?s ?p ?o ?g }
WHERE {
 GRAPH <http://intent> {
      ?r rdf:type int:Requester
 }
 ?s ?p ?o.
 {
  {
    ?r rdf:type sm:User; ?v6 ?s
  } UNION {
    ?r rdf:type sm:User.
    ?s ?v6 ?r
  }
 }
} PRIORITY 4
```

Coverage per minimal intent:

?s	?p	?o	?g	?r
ex:alice	sm:uses	ex:ssa		ex:alice
ex:alice	rdf:type	sm:User		ex:alice
ex:alice	sm:emergency_phone	075 987 654		ex:alice
ex:ssa	sm:provided_by	ex:hospital		ex:alice
ex:ssa	rdf:type	sm:SensorSyncApplicaton		ex:alice
ex:t2	sm:for_patient	ex:alice		ex:alice
ex:t2	sm:has_doctor	ex:ben		ex:alice
ex:t2	rdf:type	sm:Treatment		ex:alice
ex:t2	sm:from	2017-04-13		ex:alice
ex:t2	sm:to	2017-04-23		ex:alice
ex:ben	sm:works_at	ex:hospital		ex:ben
ex:ben	rdf:type	sm:User		ex:ben
ex:ben	sm:phone	075 555 555		ex:ben
ex:hospital	sm:location	_:b0		ex:ben
ex:hospital	rdf:type	sm:Hospital		ex:ben
ex:hospital	sm:network_address	192.168.100.0/24		ex:ben
ex:t2	sm:for_patient	ex:alice		ex:ben
ex:t2	sm:has_doctor	ex:ben		ex:ben
ex:t2	rdf:type	sm:Treatment		ex:ben
ex:t2	sm:from	2017-04-13		ex:ben
ex:t2	sm:to	2017-04-23		ex:ben
ex:t3	sm:has_doctor	ex:ben		ex:ben
ex:t3	sm:for_patient	ex:john		ex:ben
ex:t3	rdf:type	sm:Treatment		ex:ben
ex:t3	sm:from	2017-07-01		ex:ben
ex:t3	sm:to	2017-09-20		ex:ben
ex:bob	sm:uses	ex:ssa		ex:bob
ex:bob	rdf:type	sm:User		ex:bob
ex:bob	sm:emergency_phone	075 123 456		ex:bob
ex:s1	sm:owner	ex:bob		ex:bob
ex:s1	rdf:type	sm:HealthSensor		ex:bob
ex:s1	sm:regular_to	140		ex:bob
ex:s1	sm:regular_from	60		ex:bob
ex:s1	sm:stype	Pulse		ex:bob
ex:s1	sm:unit	bpm		ex:bob
ex:ssa	sm:provided_by	ex:hospital		ex:bob
ex:ssa	rdf:type	sm:SensorSyncApplicaton		ex:bob
ex:t1	sm:for_patient	ex:bob		ex:bob
ex:t1	sm:has_doctor	ex:john		ex:bob
ex:t1	rdf:type	sm:Treatment		ex:bob
ex:t1	sm:from	2017-07-20		ex:bob
ex:t1	sm:to	2017-09-20		ex:bob
ex:hospital	sm:location	_:b0		ex:john
ex:hospital	rdf:type	sm:Hospital		ex:john
ex:hospital	sm:network_address	192.168.100.0/24		ex:john
ex:john	sm:works_at	ex:hospital		ex:john
ex:john	rdf:type	sm:User		ex:john
ex:john	sm:phone	070 111 111		ex:john
ex:s2	sm:location	_:b1		ex:john
ex:s2	sm:owner	ex:john		ex:john
ex:s2	rdf:type	sm:Sensor		ex:john
ex:s2	sm:unit	C		ex:john
ex:s2	sm:stype	Temperature		ex:john
ex:t1	sm:for_patient	ex:bob		ex:john
ex:t1	sm:has_doctor	ex:john		ex:john
ex:t1	rdf:type	sm:Treatment		ex:john
ex:t1	sm:from	2017-07-20		ex:john
ex:t1	sm:to	2017-09-20		ex:john
ex:t3	sm:has_doctor	ex:ben		ex:john
ex:t3	sm:for_patient	ex:john		ex:john
ex:t3	rdf:type	sm:Treatment		ex:john
ex:t3	sm:from	2017-07-01		ex:john
ex:t3	sm:to	2017-09-20		ex:john

U1: The users can view their own properties and the direct properties of the resources connected with them:

```
ALLOW MODIFY {?s ?p ?o ?g}
WHERE {
 GRAPH <http://intent> {
   ?r a int:Requester
 }
 BIND (?r as ?s)
 ?s ?p ?o
 FILTER (?p=sm:name || ?p=sm:phone ||
     ?p=sm:email || ?p=sm:emergency_phone)
}
PRIORITY 5
```

Coverage per minimal intent:

?s	?p	?o	?g	?r
ex:alice	sm:emergency_phone	075 987 654		ex:alice
ex:ben	sm:phone	075 555 555		ex:ben
ex:bob	sm:emergency_phone	075 123 456		ex:bob
ex:john	sm:phone	070 111 111		ex:john

U2: The users can manage their name, phone, emergency phone or email:

```
ALLOW READ {?s ?p ?o ?g}
WHERE {
 GRAPH <http://intent> {
   ?r a int:Requester
 }
 ?v7 sm:for_patient ?r.
 ?v8 sm:has_doctor ?s .
 ?s ?p ?o
}
PRIORITY 2
```

Coverage per minimal intent:

?s	?p	?o	?g	?r
ex:ben	sm:works_at	ex:hospital		ex:alice
ex:ben	sm:works_at	ex:hospital		ex:alice
ex:ben	rdf:type	sm:User		ex:alice
ex:ben	rdf:type	sm:User		ex:alice
ex:ben	sm:phone	075 555 555		ex:alice
ex:ben	sm:phone	075 555 555		ex:alice
ex:john	sm:works_at	ex:hospital		ex:alice
ex:john	rdf:type	sm:User		ex:alice
ex:john	sm:phone	070 111 111		ex:alice
ex:ben	sm:works_at	ex:hospital		ex:bob
ex:ben	sm:works_at	ex:hospital		ex:bob
ex:ben	rdf:type	sm:User		ex:bob
ex:ben	rdf:type	sm:User		ex:bob
ex:ben	sm:phone	075 555 555		ex:bob
ex:ben	sm:phone	075 555 555		ex:bob
ex:john	sm:works_at	ex:hospital		ex:bob
ex:john	rdf:type	sm:User		ex:bob
ex:john	sm:phone	070 111 111		ex:bob
ex:ben	sm:works_at	ex:hospital		ex:john
ex:ben	sm:works_at	ex:hospital		ex:john
ex:ben	rdf:type	sm:User		ex:john
ex:ben	rdf:type	sm:User		ex:john
ex:ben	sm:phone	075 555 555		ex:john
ex:ben	sm:phone	075 555 555		ex:john
ex:john	sm:works_at	ex:hospital		ex:john
ex:john	rdf:type	sm:User		ex:john
ex:john	sm:phone	070 111 111		ex:john

P1: The patients can see everything about the doctors:

```
ALLOW MODIFY { ?s ?p ?o ?g }
WHERE {
 GRAPH <http://intent> {
  ?r a int:Requester.
  ?ag a int:Agent; int:address ?ip.
  ?ip int:network ?n
 }
 ?r sm:works_at ?v8.
 ?v8 sm:network_address ?n.
 ?v9 sm:has_doctor ?r; sm:for_patient ?v11.
 ?v10 sm:owner ?v11.
 GRAPH ?g {
  ?s sm:sensor ?v10; ?p ?o
 }
} PRIORITY 7
```

Coverage per minimal intent:

?s	?p	?o	?g	?r	?n
ex:o3	sm:sensor	ex:s2	ex:ssa	ex:ben	192.168.100.0/24
ex:o3	rdf:type	sm:Observation	ex:ssa	ex:ben	192.168.100.0/24
ex:o3	sm:time	1500386690319	ex:ssa	ex:ben	192.168.100.0/24
ex:o3	sm:val	28	ex:ssa	ex:ben	192.168.100.0/24
ex:o1	sm:sensor	ex:s1	ex:ssa	ex:john	192.168.100.0/24
ex:o1	rdf:type	sm:Observation	ex:ssa	ex:john	192.168.100.0/24
ex:o1	sm:time	1500386600319	ex:ssa	ex:john	192.168.100.0/24
ex:o1	sm:val	66	ex:ssa	ex:john	192.168.100.0/24
ex:o2	sm:sensor	ex:s1	ex:ssa	ex:john	192.168.100.0/24
ex:o2	rdf:type	sm:Observation	ex:ssa	ex:john	192.168.100.0/24
ex:o2	sm:time	1500386690319	ex:ssa	ex:john	192.168.100.0/24
ex:o2	sm:val	57	ex:ssa	ex:john	192.168.100.0/24

D1: The doctors can manage their patients' measurements from their hospital's network during office hours:

```
DENY MODIFY { ?s ?p ?o ?g }
WHERE {
 GRAPH <http://intent> {
  ?r a int:Requester
 }
 ?v19 sm:has_doctor ?r; sm:for_patient ?v20;
  sm:from ?v22; sm:to ?v23  .
 ?v21 sm:owner ?v20.
 GRAPH ?g {
  ?s sm:sensor ?v21.
  ?s ?p ?o
 }
 BIND (xsd:date(now()) as ?v24)
 FILTER (xsd:date(?v22) > ?v24
  || xsd:date(?v23)<?v24)
} PRIORITY 8
```

Coverage per minimal intent[a]:

?s	?p	?o	?g	?r	?n
ex:o1	sm:sensor	ex:s1	ex:ssa	ex:john	192.168.100.0/24
ex:o1	rdf:type	sm:Observation	ex:ssa	ex:john	192.168.100.0/24
ex:o1	sm:time	1500386600319	ex:ssa	ex:john	192.168.100.0/24
ex:o1	sm:val	66	ex:ssa	ex:john	192.168.100.0/24
ex:o2	sm:sensor	ex:s1	ex:ssa	ex:john	192.168.100.0/24
ex:o2	rdf:type	sm:Observation	ex:ssa	ex:john	192.168.100.0/24
ex:o2	sm:time	1500386690319	ex:ssa	ex:john	192.168.100.0/24
ex:o2	sm:val	57	ex:ssa	ex:john	192.168.100.0/24

[a]The query is executed on 2017-08-04

D2: The doctors can not manage the measurements out of their treatment timespan:

```
ALLOW MANAGE
WHERE {
 GRAPH <http://intent> {
  ?r a int:Requester.
  {
   {
    ?a rdf:type sp:Create
   } UNION {
    ?a rdf:type sp:Drop
   }
  }
  ?a sp:graphIRI ?sp
 }
 ?sp a sm:SensorSyncApplication; sm:provided_by ?h.
 ?r sm:works_at ?h; sm:dtype "TechnicalStaff"
}
PRIORITY 10
```

TS1: Technical Staff can manage applications only for his/hers hospital:

```
ALLOW MANAGE
WHERE {
 GRAPH <http://intent> {
  ?i int:requester ex:john; int:action ?a.
  ?a a ex:GenerateReport.
 }
} PRIORITY 12
```

SU1: The user ex:ben can generate reports:

```
ALLOW READ {?s ?p ?o ?g}
WHERE {
 GRAPH <http://intent> {
    ?r a int:Requester
 }
 BIND(sm:emergency_phone as ?p)
 ?v12 sm:for_patient ?s; sm:has_doctor ?r .
 ?v13 sm:owner ?s; sm:regular_from ?v14;
  sm:regular_to ?v15 .
 GRAPH ?v16 {
  ?v17 sm:sensor ?v13; sm:val ?v18
 }
 ?s ?p ?o
 FILTER (?v18 > ?v15 || ?v18 < ?v14)
}
PRIORITY 12
```

Coverage per minimal intent:

?s	?p	?o	?g	?r
ex:bob	sm:emergency_phone	075 123 456		ex:john

EM1: The doctor can access their patients emergency phone number during abnormal measurements:

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic Web," *Sci. Amer.*, vol. 284, no. 5, pp. 28–37, 2001.

[2] T. Heath and C. Bizer, "Linked data: Evolving the Web into a global data space," *Synth. Lectures Semantic Web, Theory Technol.*, vol. 1, no. 1, pp. 1–136, Feb. 2011.

[3] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data—The story so far," in *Semantic Services, Interoperability and Web Applications: Emerging Concepts*. Hershey, PA, USA: IGI Global, 2009, pp. 205–227.

[4] R. Cyganiak and A. Jentzsch, "Linking open data cloud diagram," vol. 12, Sep. 2011. [Online]. Available: http://lod-cloud.net/versions/2011-09-19/lod-cloud.png

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A nucleus for a Web of open data," in *The Semantic Web*. 2007, pp. 722–735.

[6] G. Andrienko, A. Gkoulalas-Divanis, M. Gruteser, C. Kopp, T. Liebig, and K. Rechert, "Report from Dagstuhl: The liberation of mobile location data and its implications for privacy research," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 17, no. 2, pp. 7–18, Apr. 2013.

[7] G. Klyne and J. J. Carroll, "Resource description framework (RDF): Concepts and abstract syntax," W3C, W3C Rec., Feb. 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[8] S. Das, S. Sundara, and R. Cyganiak, "R2RML: RDB to RDF mapping language," W3C, W3C Rec., Sep. 2012. [Online]. Available: http://www.w3.org/TR/2012/REC-r2rml-20120927/

[9] C. Bizer and R. Cyganiak, "D2R server—Publishing relational databases on the semantic Web," in *Proc. 5th Int. Semantic Web Conf.*, vol. 175. 2006, pp. 1–2.

[10] J. Hollenbach, J. Presbrey, and T. Berners-Lee, "Using RDF metadata to enable access control on the social semantic Web," in *Proc. Workshop Collaborative Construction, Manage. Linking Struct. Knowl. (CK)*, vol. 514. 2009, pp. 1–10.

[11] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila, "Proteus: A semantic context-aware adaptive policy model," in *Proc. 8th IEEE Int. Workshop Policies Distrib. Syst. Netw. (POLICY)*, Jun. 2007, pp. 129–140.

[12] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *Proc. IEEE 4th Int. Workshop Distrib. Syst. Netw. (POLICY)*, Jun. 2003, pp. 63–74.

[13] F. Abel, J. L. De Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla, "Enabling advanced and context-dependent access control in RDF stores," in *The Semantic Web*. New York, NY, USA: Springer, 2007, pp. 1–14.

[14] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994.

[15] S. Kirrane, "Linked data with access control," Ph.D. dissertation, Nat. Univ. Ireland, Galway, Ireland, 2015.

[16] G. Flouris, I. Fundulaki, M. Michou, and G. Antoniou, "Controlling access to RDF graphs," in *Proc. Future Internet Symp.*, 2010, pp. 107–117.

[17] S. Franzoni, P. Mazzoleni, S. Valtolina, and E. Bertino, "Towards a fine-grained access control model and mechanisms for semantic databases," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2007, pp. 993–1000.

[18] S. Oulmakhzoune, N. Cuppens-Boulahia, F. Cuppens, and S. Morucci, "*f*Query: SPARQL query rewriting to enforce data confidentiality," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy*, 2010, pp. 146–161.

[19] W. Chen and H. Stuckenschmidt, "A model-driven approach to enable access control for ontologies," in *Proc. Wirtschaftsinformatik*, vol. 1. 2009, pp. 663–672.

[20] A. Padia, T. Finin, and A. Joshi, "Attribute-based fine grained access control for triple stores," in *Proc. 3rd Soc., Privacy Semantic Web-Policy Technol. Workshop, 14th Int. Semantic Web Conf.*, 2015, pp. 1–15.

[21] L. Costabello, S. Villata, O. R. Rocha, and F. Gandon, "Access control for HTTP operations on linked data," in *Proc. Extended Semantic Web Conf.*, 2013, pp. 185–199.

[22] O. Sacco and J. G. Breslin, "*PPO & PPM 2.0*: Extending the privacy preference framework to provide finer-grained access control for the Web of data," in *Proc. 8th Int. Conf. Semantic Syst.*, 2012, pp. 80–87.

[23] S. Kirrane, A. Mileo, and S. Decker, "Access control and the resource description framework: A survey," *Semantic Web*, vol. 8, no. 2, pp. 311–352, 2017.

[24] H. Mühleisen, M. Kost, and J.-C. Freytag, "SWRL-based access policies for linked data," in *Proc. SPOT*, vol. 80. 2010, pp. 1–12.

[25] S. Dietzold and S. Auer, "Access control on RDF triple stores from a semantic Wiki perspective," in *Proc. ESWC Workshop Scripting Semantic Web*, 2006, pp. 1–9.

[26] P. Bonatti and D. Olmedilla, "Driving and monitoring provisional trust negotiation with metapolicies," in *Proc. 6th IEEE Int. Workshop Policies Distrib. Syst. Netw.*, Jun. 2005, pp. 14–23.

[27] J. Li and W. K. Cheung, "Query rewriting for access control on semantic Web," in *Proc. Workshop Secure Data Manage.*, 2008, pp. 151–168.

[28] O. Sacco and A. Passant, "A privacy preference manager for the social semantic Web," in *Proc. SPIM*, 2011, pp. 42–53.

[29] N. Lopes, S. Kirrane, A. Zimmermann, A. Polleres, and A. Mileo, "A logic programming approach for access control over RDF," in *Proc. Tech. Commun. 28th Int. Conf. Logic Program.*, vol. 17. 2012, pp. 381–392, 10.4230/LIPIcs.ICLP.2012.381.

[30] M. Howard and S. Lipner, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Beijing, China: Publishing House of Electronics Industry, 2008.

[31] T. Ryutov, T. Kichkaylo, and R. Neches, "Access control policies for semantic networks," in *Proc. IEEE Int. Symp. Policies for Distrib. Syst. Netw. (POLICY)*, Jul. 2009, pp. 150–157.

[32] J. L. De Coi, D. Olmedilla, P. A. Bonatti, and L. Sauro, "Protune: A framework for semantic Web policies," in *Proc. Int. Semantic Web Conf. (Posters Demos)*, vol. 401. 2008, p. 128.

[33] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic Web rule language combining OWL and RuleML," in *W3C Member Submission*, vol. 21. 2004, p. 79.

[34] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, W3C Rec., Jan. 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[35] S. Harris, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 query language," W3C, W3C Rec., Mar. 2013, vol. 21. [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[36] V. Kolovski, J. Hendler, and B. Parsia, "Analyzing Web access control policies," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 677–686.

[37] R. Cyganiak, "A relational algebra for SPARQL," Digit. Media Syst. Lab. HP Lab., Bristol, U.K., Tech. Rep. HPL-2005-170, 2005, vol. 35.

[38] C. J. Date, *An Introduction to Database Systems*. Noida, India: Pearson Education, 2006.

[39] B. McBride, "Jena: A semantic Web toolkit," *IEEE Internet Comput.*, vol. 6, no. 6, pp. 55–59, Nov./Dec. 2002.

[40] C. Scarioni, *Pro Spring Security*. New York, NY, USA: Apress, 2013.

[41] M. Sporny, T. Inkster, H. Story, B. Harbulot, and R. Bachmann-Gmür, "Webid 1.0: Web identification and discovery," in *Proc. W3C*, 2011.

[42] H. Story, B. Harbulot, I. Jacobi, and M. Jones, "FOAF+SSL: RESTful authentication for the social Web," in *Proc. 1st Workshop Trust Privacy Social Semantic Web (SPOT)*, 2009, pp. 1–12.

[43] T. Finin *et al.*, "*ROWL*BAC: Representing role based access control in *OWL*," in *Proc. 13th ACM Symp. Access Control Models Technol.*, 2008, pp. 73–82.

[44] L. Costabello, S. Villata, N. Delaforge, and F. Gandon, "Linked data access goes mobile: Context-aware authorization for graph stores," in *Proc. LDOW-5th WWW Workshop Linked Data Web*, 2012, pp. 1–8.

[45] S. Ferré, "Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language," *Semantic Web*, vol. 8, no. 3, pp. 405–418, 2017.

[46] A. Bernstein, E. Kaufmann, and C. Kaiser, "Querying the semantic Web with ginseng: A guided input natural language search engine," in *Proc. 15th Workshop Inf. Technol. Syst.*, Las Vegas, NV, USA, 2005, pp. 112–126.

[47] A. Bernstein and E. Kaufmann, "GINO—A guided input natural language ontology editor," in *Proc. Int. Semantic Web Conf.*, 2006, pp. 144–157.

RISTE STOJANOV received the Diploma degree in computer science and the M.Sc. degree in content-based retrieval from Saints Cyril and Methodius University, Skopje, where he is currently pursuing the Ph.D. degree in semantic web security with the Faculty of Computer Science and Engineering with focus to authorizing access to the data in Linked Data business applications. He is currently a Teaching and Research Assistant with Saints Cyril and Methodius University. His research interests include data science, data security, semantic web, enterprise application architectures, and natural language processing.

SASHO GRAMATIKOV received the bachelor's degree in computer science, information technologies and automation, the master's degree in computer science and computer engineering degree from Saints Cyril and Methodius University, Skopje, Macedonia, in 2005 and 2009, respectively, and the Ph.D. degree from the Universidad Politécnica de Madrid, Madrid, Spain. He is currently an Assistant Professor with the Faculty of Computer Science and Engineering, Saints Cyril and Methodius University. His research interests are multimedia distribution systems, computer networks, and semantic web.

IGOR MISHKOVSKI was born in Skopje, Macedonia, in 1981. He received the master's degree in computer science and engineering from Saints Cyril and Methodius University, Skopje, in 2008, and the Ph.D. degree from the Politecnico di Torino, Torino, Italy, in 2012. After he received the Ph.D. degree, he was elected as an Associate Professor with the Faculty of Computer Science and Engineering, Saints Cyril and Methodius University, in 2012. His research interests include complex networks and modeling dynamical processes, network science, computer networks, semantic web, and operating systems.

DIMITAR TRAJANOV was the Dean of the Faculty of Computer Science and Engineering, Saints Cyril and Methodius University, Skopje, from 2011 to 2015, where he is currently the Head of Department of Information Systems and Network Technologies, Faculty of Computer Science and Engineering. He is the Leader of the Regional Social Innovation Hub established as a co-operation between UNDP and the Faculty of Computer Science and Engineering. He has authored over 130 journal and conference papers and seven books. He has been involved in more than 40 international and national scientific and applicative projects as a project lead or a participant. His research interests include data science, semantic web, big data, open data, social innovation, e-commerce, entrepreneurship, technology for development, mobile development, ad hoc networks, parallel processing, reliability, and system-on-chip design.