Semantic-Based Authorization: Need for Context

Mouiad AL-Wahah, Csilla Farkas
University of South Carolina, USA

Abstract

In this paper, we present a context-based access control authorization framework. Our approach is suitable to incorporate dynamically changing access control requirements. We express authorization requirements and contextual information as ontologies. The context ontology, denoted as CTX-Lite, serves as a core ontology for context handling. The authorization ontology, denoted as CBAC ontology, is used for modeling access control policy requirements. We use Description Logic (DL) and Logic Programming (LP) technologies for implementing Context-Based Access Control (CBAC). In our framework, access authorization decision is made based on the context of the request and the resources. We show that semantic-based techniques can support adaptive and dynamic context-based authorization. We also show that our framework is expressive enough to incorporate the needs of emerging technologies such as Internet of Things (IoT). Furthermore, we develop a proof of concept implementation to demonstrate our work. Moreover, we provide the complexity analysis of the framework and contrast the complexity against possible optimization that can be applied on the framework.

1. Introduction

Traditional authorization models are limited to support environments such as ubiquitous and Internet of Things (IoT). Sensing capabilities embedded in computing devices offer users the ability to share, retrieve, and update resources anytime and anywhere. These advances have come with security challenges due to the variety and mobility of the networked devices and the changing context. In addition, there are unique end-user needs. Traditional access control policies, such as Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Role-Based Access Control (RBAC), are insufficient for supporting these advances. These access control models assume a prior knowledge of collaborating entities. Moreover, these models only support static access control policies. Attribute-Based Access Control (ABAC) solves some of the distributed networks security problems, but it does not provide a comprehensive solution. For example, all participants must agree on a set of attributes and their meaning when using ABAC. This is error-prone and difficult to accomplish in heterogeneous environments like IoT [1]. Context-based authorization methods have been proposed to address the above limitations. Current approaches for Context-Based Access Control (CBAC) can be divided into three categories depending on how the context is used. In the first category, the context is associated with roles in RBAC [2], [3], [4]. Based on that context and the user's roles together, the authorization decision is made. However, this approach does not solve the problem of exponential expansion of the number of roles. The second category uses the context as a set of attributes for ABAC [5], [6], [7]. The approaches in this category need the attributes to be known in advance which is not possible in dynamic environments. The third category approaches [8], [9], [10] are the closest to our work. These approaches are similar to our approach, use the context as the main component by which the access the authorization decision is made. However, they have high complexity that makes them impractical to be used in dynamic environment. For example, Resource Description Framework (RDF)-based semantics that are proposed by authors of [5] cannot cope with large vocabulary of the current environments. Moreover, current semantic-based tools rely on Web Ontology Language (OWL) and therefore RDF)-based semantics is too incapable to express the semantic needs of the evolving environments. Contextual information is dynamic during communication sessions. Mobility of the users and devices will further complicate the authorization process. In such environments, an effective access control (AC) approaches must handle context changes. Consider the following scenario that we will use as a running example throughout the rest of this paper. " Martha is a 55-year old woman and she has health problems, such as hypertension and heart arrhythmia. She uses a wearable smart device with heartbeat and blood pressure monitors embedded in it. The device transmits its data along with the time of sampling and the device's geo-spatial coordinates as continuous streams to Martha's smart phone. The health monitoring application running on Martha's smart phone transmits the received data to the health agency's server. The health agency server uses these data items to continually monitor Martha's health status. In case of a situation that may be an emergency, the health agency will 1.) alert 911 and transmits Martha's location and status. Assume that Martha has a severe increase of her heartbeat. During exercising, this heartbeat could have been normal. However, in the context of resting at home, this may

be an emergency." In this paper, we propose a flexible and adaptive context-based access control model. Our framework uses contextual information to derive access decisions. We have developed two OWL ontologies. The first one, CTX-Lite ontology, is used for representing and reasoning over the contextual information. The second ontology, CBAC, is used to represent and reason over the Knowledge Base (KB) to derive the access control decisions. We use incremental ontological reasoning, by employing Pellet reasoner, to ensure that our approach is feasible for real world applications. This process forms the bulk of reasoning in our approach and is used for checking ontology satisfiability and consistency, conceptual subsumption, and instance realization. Logic programming reasoning is only used for modeling access control rules and for situations that need the use of variables. More specifically, CTX-Lite ontology is used to annotate the raw contextual data. The annotation is then used by CTX-Lite to derive inferred high-level context from the raw observations. This high-level context is then used by the policy engine to derive a decision. CTX-Lite ontology serves as a core ontology for context handling operations. The approach described in this paper is implemented using semantic web technologies, such as OWL ontologies, Pellet reasoner, Jena inference rules, and SPARQL (SPARQL Protocol And RDF Query Language) queries. The remainder of this paper is structured as follows: in section 2, we present our formal representation of the context. Section 3 is dedicated to CBAC modeling, and in section 4 we show our implementation results. Complexity analysis of the framework is presented and contrasted to optimized version of the approach in section 5. Related works are described in section 6, and in section 7 we conclude with suggestions for future work.

2. Context Representation

The framework has two main components, Figure 1, context manager and authorization engine. We separate context operations from access authorization operations. This is to reduce processing cost for network devices and to make the framework more modular. The context manager receives requests from the policy engine asking for a specific context. In response, the context manager generates the corresponding context and sends it to the policy engine. Based on the received context and its own Knowledge Base (KB), the policy engine derives an access control decision.

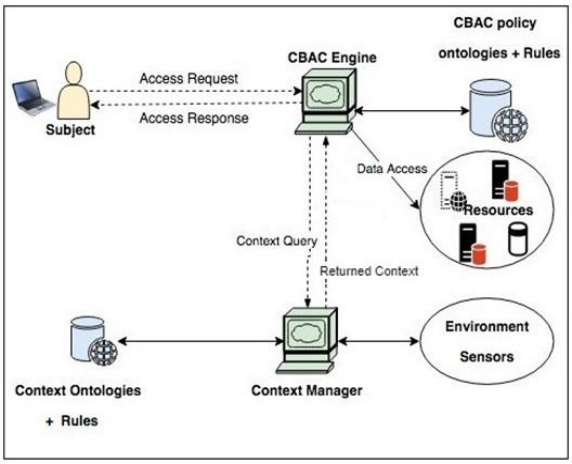

Figure 1. Architecture of the Context-Based Access Control

We use Pellet reasoner for checking ontology satisfiability and consistency, conceptual subsumption, and instance realization. Logic programming reasoning is used to represent access control rules and for situations that need the use of variables. Ontologies are represented using Web Ontology Language (OWL). OWL is a practical implementation of the Description Logic (DL) family known as SROIQ(D). It allows one to define classes, properties, and individuals. An OWL ontology includes a set of class axioms to describe classes, which constitute the Terminological Box (TBox); a set of property axioms to describe properties, which constitute a Role Box (RBox); and a set (may be zero) of assertions to describe individuals, which constitute an Assertion Box (ABox). Properties can be either object properties or data properties. Classes can be viewed as formal descriptions of sets of objects (taken from a nonempty universe), and individuals can be viewed as names of objects of the universe. A class is either an atomic class or a complex class, the later built via several available constructors that express Boolean operations and different types of restrictions on the members of the class. Basic DL constructors are:

- $\{a, b, c, \dots\}$ is the set of individuals that belong to a concept C.
- $C \sqcap D$, is the set of individuals that belong to both concepts C and D.
- $C \sqcup D$, is the set of individuals that belong to concept C or concept D.
- $C \sqsubseteq D$, concept D subsumes concept C, i.e., all individuals that are members of concept C are also members of concept D.
- $C \equiv D$, Concept C is equivalent to concept D.
- $\forall R.C$ is the set of individuals (if any) that are in relationship R with individuals that belong to concept C.
- $\exists R.C$ is the set of individuals (at least one) that are in relationship R with individuals that belong to concept C.

- $\top$ is the set of all individuals.
- $\bot$ is the empty set.

We adopt the formalism of [11], [12], [13] to define the syntax and semantics of our model. A DL (or its OWL representation) model is a set of axioms which make statements about how concepts (and hence the individuals belonging to these concepts) and roles are related to each other [4].

In our framework [14], we differentiate between two context's building blocks, the *reference context* and *active context*. The *reference context* is used as a generic template that holds all the high-level characterizing specifications of specific context of an entity. This high-level context will be used later as a reference when we need to instantiate the active context of that entity. The *active context* holds the entity context at a specific instant of time. For example, when an entity requests an access to a resource. Active contexts are like their reference contexts counterparts. However, they differ in that they do not have range values in their definitions. Active context reflects a real snapshot of an entity's context at a specific time instant. For example, the following DL axiom is used to represent the reference context for *InEmergency* context of a user for our running example, Figure 2.

$$InEmergency \equiv User \sqcap \exists owns.(Device \sqcap \exists hasDevice.(SensingDevice \sqcap \exists madeObservation.(Observation \sqcap \exists observedProperty.(Property \sqcap \exists featureofinterest.(FeatureOfInterest \sqcap \exists observation\ result.(Sensor\ Output \sqcap \exists hasValue.(Obsevation\ Value \sqcap \exists hasDataValue.\{xsd:float[\geq 180.0, \leq 55.0]\} \sqcap \exists hasUnit.Unit \sqcap hasTime.Time \sqcap hasLocation.Location \sqcap \exists mappedTo.(Place))))))))$$

Figure 2. Patient's *InEmergency* context

An instance (*active context*) of the above *reference context* is shown in Figure 3.

These DL axioms indicate that a user is InEmergency context if she owns a device (**AcceMIS2DH**) and this device has a sensing device (**HBSD100**) which has an observation ($\boldsymbol{o_1}$). This observation is associated with a property (pulse rate in Martha case, it is a general property to be observed) and this property has a feature of interest (heart beat rate, a special feature of the observed property). The observation result ($\boldsymbol{so}_1$) is an output made by the sensing device and it has an observation value ($\boldsymbol{ov_1}$), which in turn, has a data value that represents the raw data measured by the sensing device and it is of float type (**53**).

$$InEmergency \equiv User\{\textbf{Martha}\} \sqcap \exists owns.(Device\{\textbf{AcceMIS2DH}\} \sqcap \exists hasDevice.(SensingDevice\{\textbf{HBSD100}\} \sqcap \exists madeObservation.(Observation\{\textbf{o1}\} \sqcap \exists observedProperty.(Property\{\textbf{pulse rate}\} \sqcap \exists featureofinterest.(FeatureOfInterest\{\textbf{heart beat rate}\} \sqcap \exists observation\ result.(Sensor\ Output\{\textbf{so1}\} \sqcap \exists hasValue.(Obsevation\ Value\{\textbf{ov1}\} \sqcap \exists hasDataValue.\{xsd:float[\textbf{53}]\} \sqcap \exists hasUnit.Unit\{\textbf{bpm}\} \sqcap hasTime.Time\{\textbf{2:43 pm}\} \sqcap hasLocation.Location\{\boldsymbol{\ell_1}\} \sqcap \exists mappedTo.(Place\{\textbf{gym}\}))))))))$$

Figure 3. *InEmergency* active context of Martha

The raw data has a unit (bpm, beats per minute) and a measurement sampling time (**2:43 pm**). The location of the observation is ($\boldsymbol{\ell_1}$) which is mapped to a place (**gym room**).

2.1 The Lightweight Context Ontology CTX-Lite

The context ontology, CTX-Lite, defines eight basic concepts as top-level concepts; these concepts are *Agent*, *Device*, *Service*, *Network*, *Location*, *Time*, *Activity* and *Context*. The ontology has the concept *Context* as the top concept. The *Context* concept represents the higher contextual concept and it has two subcontexts for representing atomic context and composite context, and these are *Atomicctx* and *Compositectx*, respectively. The *Compositectx* has seven subconcepts that represent our four fundamental high-level contexts as well as other three concepts for handling unnamed contexts, personal contexts, and spatiotemporal contexts. These sub concepts are: *Behavioral*, *Temporal*, *Spatial*, *Spatiotemporal*, *Environmental*, *Personal*, and *Unnamed* concept. The CTX-Lite ontology is only 148 RDF triples, and this is to make it as generic as possible.

3. Modeling CBAC Policy

We adopt the context as the core element around which all access authorization operations are based. In our approach, the access authorization depends on the visibility of certain contexts that are to be fed to the access control engine. The access control engine, depending on the provided context information, may deny or permit such an access authorization request. Access control policy requirements are represented using the ontology shown in Figure 4 and Jena rules.

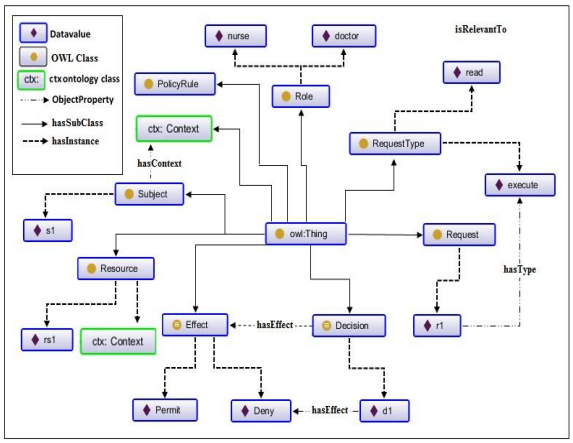

Figure 4. CBAC ontology

The context-based access control model, or CBAC model, consists of a set of rules and each rule is given by the following definition:
$rule = \langle r, rt, rs, sc, ro, rd, de, gc^* \rangle$, where
$rule \in Rule$, $r \in Request$, $rt \in RequestType$, $rs \in RequestSubject$, $sc \in SubjectContext$, $ro \in RequestObject$, $oc \in ObjectContext$, $rd \in RequestDesicion$, $de \in DesicionEffect$, and $gc^* \in GeneralContext$[1].

Each rule is triggered by an access request and is evaluated to a decision at runtime. The decision has an effect which, in our model, is either a "*Deny*" or a "*Permit*".

Now let's represent our access control model using DL (and hence in OWL-DL). Given the above clarification of how to represent rules, the rule axiom becomes straightforward and is given as shown in Figure 5.

$$rule \equiv Request \sqcap \exists hasType.RequestType \sqcap \exists hasSubject.(Subject \sqcap \exists hasContext.Context) \sqcap hasObject.(Object \sqcap \exists hasContext.Context) \sqcap \exists hasDecision.(Decision \sqcap \exists hasEffect.\{"Deny","Permit"\})$$

Figure 5. Request rule axiom

When all the access control rules are specified as we have done above, runtime evaluation of the access authorization is accomplished when a request is received by access control policy engine. From our running example, we build two access control authorization rules that are shown in Table I.

[1] This is for general context usage. The star on the *gc* is to indicate that this field is optional while the other fields are mandatory.

TABLE I: access authorization rules for the running example

No.	Rule
Rule-1:	An owner of a device can access the device's resources irrespective of her/his context.
Rule-2:	A paramedic personnel who is on duty at the time of emergency can access patient device's resources in case that patient suffers from that emergency case.

The first rule does not encompass any context and hence there is no need for complex rule. For the second rule of the policy we need two contexts, one associated with the subject (Joe in our case) and the second is associated with the resource's owner (not the object). The *InEmergency* context for Martha's health condition using our approach is given in Figure 4. While Joe's *OnDuty context* is given Figure 6.

$$OnDuty \equiv Subject\{\mathbf{Joe}\} \sqcap \exists hasActivity.(Activity\{\mathbf{Working}\} \sqcap \exists hasTime.(TemporalEntity\{t_1\}.\exists hasBeginning\{\mathbf{08:00\ am}\} \sqcap \exists hasEnd\{\mathbf{04:00\ pm}\}) \sqcap hasLocation.(Location\{\ell_2\} \sqcap \exists mappedTo.Place\{\mathbf{Emergency\ Center}\})) \sqcap \exists hasRole.Role\{\mathbf{paramedic}\} \sqcap \exists hasID.\{\mathbf{MSS56006}\}$$

Figure 6. OnDuty context of Joe

4. Implementation Details

All experiments were conducted using two laptops, PC1 is a Windows 8.1, 1.9 GHz CPU, 4 GB RAM and 500 GB HDD, while PC2 is Windows 10 3.8 GHz CPU, 32 GB RAM, and 2 TB HDD (The source code and the data sets are uploaded to https://github.com/Mouiad1975/CBAC). Ten sets of data have been generated using on-line data generator[2] and been processed to r emove duplicates. Then we used Cellfie plug-in for Protégée[3] to assert these instances to the model ontologies. The sets are 10000, 20000, 30000, 40000, 50000, 60000, 70000, 82000, 90000 and 100000 triples in their size. We have conducted tests on the CBAC engine via submitting SPARQL queries to populated samples of the CTX-Lite and CBAC ontologies in our application program. The engine successfully returned only the authorized data specified by the sample access control rules. The reasoning process is accomplished using DL reasoner Pellet while LP is

[2] freedatagenerator. (Date last accessed 27-December-2017).
[3] Protégée. (Date last accessed 13-August-2017).

accomplished using forward chaining Jena inference engine and user-defined rules. The DL reasoning has been used for checking ontology satisfiability and consistency, conceptual subsumption, and instance realization. The bulk of reasoning process in this paper is done by DL, while LP is used only for simple situations like in case of need to variables.

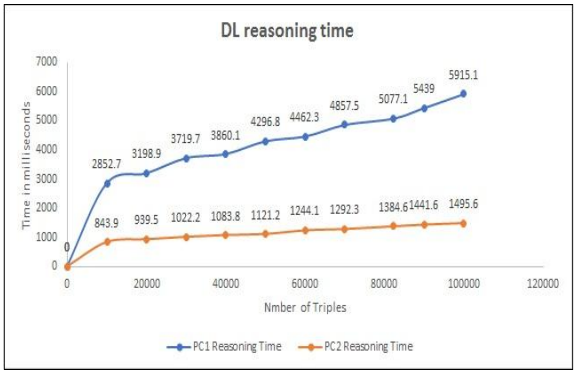

Figure 7. DL reasoning time

The reasoning process has been repeated ten times for every one of the mentioned data sets on each one of the two PCs. Then we take the average time computed for these ten repetitions for reaching concrete values for the reasoning process execution time. The same thing is repeated in calculating SPARQL query response time but for bigger data sets because these are the results of inference upon the original data set models. Figures 7, 8, and 9 represent the results of DL reasoning time, DL+PL reasoning time, and SPARQL query response time, respectively.

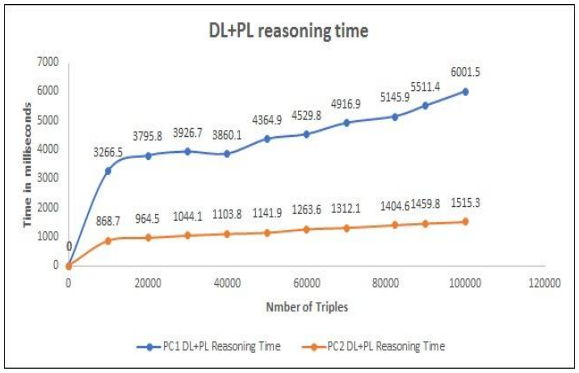

Figure 8. DL+PL reasoning time

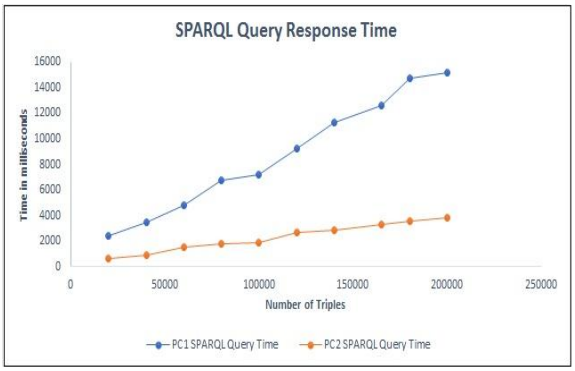

Figure 9. Response time of SPARQL query upon inference model

OnDuty context states that a user (**Joe**) is on duty when (s)he has an activity of Working that has a temporal entity. This temporal entity has a start time (**8:0 am**) and an end time (**4:0 pm**). The activity has a location (ℓ_2) also and this location is mapped to a place (**Emergency Center**). Associating the InEmergency context for Martha with *OnDuty* context for Joe will lead to a "*Permit*" effect for Joe's request to access Martha's health records using rule shown in Figure 10.

$$
\begin{aligned}
&[rule2: \\
&\quad (?r \quad rdf:type \quad cbac:Request) \\
&\quad (?r \quad ctx:hasType \quad app:access) \\
&\quad (?r \quad cbac:hasSubject \quad ?s) \\
&\quad (?s \quad rdf:type \quad app:OnDuty) \\
&\quad (?r \quad cbac:hasResource \quad ?rs) \\
&\quad (?rs \quad ctx:hasOwnwer \quad ?w) \\
&\quad (?w \quad rdf:type \quad app:InEmergency) \\
&\quad (?r \quad cbac:hasDecision \quad ?d) \\
&->(?d \quad cbac:hasEffect \quad cbac:Permit)]
\end{aligned}
$$

Figure 10. Access rule association

This rule states that if a request *r* of type access is issued to the CBAC engine by a subject *s* that has a context *OnDuty* on a resource *rs* such that the owner of the resource has a context *InEmergency* and request has a decision *d*, then insert a *Permit* effect for *d* into the KB.

In this rule, it is important to note that we do not need to specify any identity or role for the subject requesting the access to resource to decide if (s)he can access the resource or not. Also, we do not need the identity of the resource (or the owner of the resource). The whole access authorization decision

making is based on the context of the request r (the contexts of the subject and resource altogether). Furthermore, if the CBAC engine needs to know the identities of the subject and the resource, it simply queries its KB store to know that. Our model uses "*denial takes precedence*" as a conflict resolution strategy. That is, if "*Deny*" and "*Permit*" decisions are derived for an access request, then the request is denied. An access permission to a resource is granted only when the subject's context, the resource's context (or the resource owner's context as it is the case in our running example), and/or any other context that is specified by the access control policy and match a particular "*Permit*" rule. Algorithm 1 is used to compute authorization decision, Figure 11. The function evaluate is given by Algorithm 2, Figure 12. The function $evaluate(s,sc,r,rc,CBAC,RS)$ takes the access control ontology CBAC as input and achieves the DL-based reasoning to get the inferred model $infmodel$.

Algorithm 1 Access Authorization.

Input: *CBAC*, CBAC ontology and *RQ* is an Access Request. *RS* is Jena rule-set
Output: Access decision, either "Deny" or "Permit".

1: $RT \leftarrow parse(RQ)$ ▷ $RT = \langle s,r,ac \rangle$
2: **if** $RT = access$ **then**
3: $sc \leftarrow getContext(s)$;
4: $rc \leftarrow getContext(r)$;
5: $p \leftarrow evaluate(s,sc,r,rc,CBAC,RS)$;
6: **if** $p =$ "*Permit*" AND $noConflict(p,s,sc,r,rc,CBAC,RS)$ **then**
7: $return$("*Permit*");
8: $exit()$;
9: **end if**
10: **else**
11: $return$("*Deny*");
12: $exit()$;
13: **end if**

Figure 11. Access authorization algorithm

The access control policy rules, represented as Jena forward inference rules, will be applied on $infmodel$ to derive an access decision. The function $getrulesnumber(RS)$ returns the number of rules in the access control policy rule-set RS. The Boolean function $noConflict(p,s,sc,r,rc,CBAC,RS)$ given in Algorithm 3 works just like $evaluate(s,sc,r,rc,CBAC,RS)$ function but it returns a Boolean value if two conflicting rules ("*Permit*" and "*Deny*") are fired at the same time, Figure 13.

Algorithm 2 $evaluate(s,sc,r,rc,CBAC,RS)$

1: **function** $evaluate(s,sc,r,rc,CBAC,RS)$
2: $RulesNo \leftarrow getrulesnumber(RS)$;
3: $infmodel \leftarrow reason(CBAC)$;
4: **for** $i \leftarrow 1, RulesNo$ **do**
5: **if** RS[i] has the form:
$(?rq \quad rdf:type \quad cbac:Request)$
$(?rq \quad ctx:hasAction \quad cbac:ac)$
$(?rq \quad cbac:hasSubject \quad ?s)$
$(?rq \quad cbac:hasResource \quad ?r)$
$(?s \quad rdf:type \quad ctx:sc)$
$(?r \quad rdf:type \quad ctx:rc)$
$(?rq \quad cbac:hasDecision \quad ?d)$
$-> (?d \quad cbac:hasEffect \quad cbac:p)$ **then**
6: **if** $p =$ "*Permit*" **then**
7: $return(p)$;
8: $exite()$;
9: **end if**
10: **end if**
11: **end for**
12: $p =$ "*Deny*"
13: $return(p)$;
14: **end function**

Figure 12. evaluate function

Algorithm 3 $noConflict(p_1,s,sc,r,rc,CBAC,RS)$

1: **function** $noConflict(p_1,s,sc,r,rc,CBAC,RS)$
2: $RulesNo \leftarrow getrulesnumber(RS)$;
3: $infmodel \leftarrow reason(CBAC)$;
4: **for** $i \leftarrow 1, RulesNo$ **do**
5: Let RS_i has the form:
$(?rq \quad rdf:type \quad cbac:Request)$
$(?rq \quad ctx:hasAction \quad cbac:ac)$
$(?rq \quad cbac:hasSubject \quad ?s)$
$(?rq \quad cbac:hasResource \quad ?r)$
$(?s \quad rdf:type \quad ctx:sc)$
$(?r \quad rdf:type \quad ctx:rc)$
$(?rq \quad cbac:hasDecision \quad ?d)$
$-> (?d \quad cbac:hasEffect \quad cbac:p_1)$
6: **for** $j \leftarrow 1, RulesNo$ **do**
7: Let RS_j has the form:
$(?rq \quad rdf:type \quad cbac:Request)$
$(?rq \quad ctx:hasAction \quad cbac:ac)$
$(?rq \quad cbac:hasSubject \quad ?s)$
$(?rq \quad cbac:hasResource \quad ?r)$
$(?s \quad rdf:type \quad ctx:sc)$
$(?r \quad rdf:type \quad ctx:rc)$
$(?rq \quad cbac:hasDecision \quad ?d)$
$-> (?d \quad cbac:hasEffect \quad cbac:p_2)$
8: **if** ($p_1 =$ "*Permit*" And $p_2 =$ "*Deny*") OR ($p_1 =$ "*Deny*" And $p_2 =$ "*Permit*") **then**
9: $return(False)$;
10: $exite()$;
11: **end if**
12: **end for**
13: **end for**
14: $return(True)$;
15: **end function**

Figure 13. Conflict detection algorithm

5. Complexity

In this section, we present computational complexity analysis of the framework. We also investigate different solutions to mitigate the worst-case complexity. Our framework uses reasoning in two phases of CBAC, design time and runtime. At design time, we use it to determine possible conflicts and to check satisfiability of the desired static security properties. At runtime, we use reasoning to verify if an access request is permitted or denied by the system state. We also verify that the data instances are consistent with corresponding the OWL restrictions. Our CBAC engine uses SPARQL-DL queries to interact with context manager and to check its own decisions. The DL-based reasoning in our approach is achieved via Pellet reasoner and LP reasoning using Jena inference forward-chaining. The description logic we use is ALCRIQ(D), which is a fragment of SROIQ(D). ALCRIQ(D) is the extended version of ALC which supports axioms with the concept constructors ⊓, ⊔, ∃, and ¬. Our model extends ALC with role inclusions (R), inverse roles (I), qualified number of restrictions (Q) and concrete roles (D). We use Datalog Safe-rules for Jena customary rules. Jena forward-chaining engine uses RDF(S)-entailment rules for reasoning. RDF(S)-entailment is decidable and in NP-Complete(d). If the target graph does not contain blank nodes [15], [16], then RDF(S)-entailment is in PTime(d), where d is the number of nodes in RDF(S) graph. ALCRIQ(D) reasoning has two components: Taxonomic reasoning (performed at design time) and instance reasoning (performed at runtime). The computational complexity of Taxonomic reasoning is determined based on the number of axioms (e.g., C ⊔ D). It is NExpTime(n), where (n) is the number of axioms. The computational complexity of instance (data) reasoning is determined by the number of facts (e.g., Doctor(John); hasWeight(John; 70)) in the knowledge base. It is also in NExpTime(m), where (m) is the number of facts. Total computational complexity for ALCRIQ(D) is NExpTime(m)+NExpTime(n). In most practical applications the number of facts is far more than the number of axioms in the DL knowledge base, hence the computational complexity of ALCRIQ(D) is plausibly taken to be in NExpTime(m), where (m) is the number of facts. DL ALCRIQ(D) reasoning complexity dominates Jena inference complexity, hence the worst-case complexity of our framework is NExpTime(m). To enhance the performance of reasoning for real applications, we adopt incremental DL reasoning [17].

5.1 Using Incremental DL Reasoning

Incremental reasoning means the ability of the reasoner to process updates (additions or removals) applied to an ontology without having to perform all the reasoning steps from scratch. To account for incremental reasoning enhancement, our system ontologies are increased via inserting new axioms, making the total size of the system knowledge base 16000 axioms. This size is dependent as a basic size which then increased. Then, we repeatedly and randomly add 500 axioms to the system's ontologies. At each iteration (starting from 16000 axioms and after every 500 axioms addition), the DL Pellet regular reasoner is executed over the ontologies for ten times and the average time is calculated. For incremental reasoning, we first execute the incremental reasoning on the basic size (the 16000 axioms knowledge base) and then we add 500 axioms to the knowledge base without stopping the reasoner, synchronize the reasoner execution over the new knowledge base and so on. We calculate the average enhancement of incremental reasoning compared to the regular reasoning as shown in Table II and Figure 14. This is given by the following simple formula:

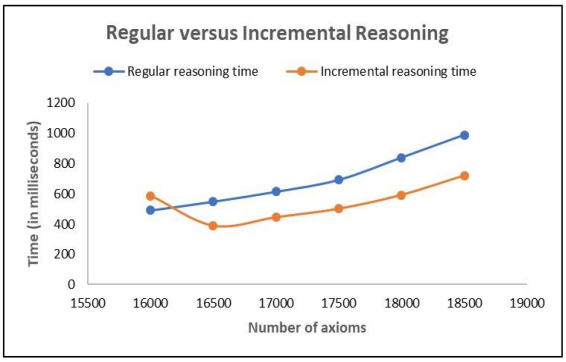

Figure 14. Regular reasoning compared to the incremental reasoning

TABLE II: Incremental versus regular Pellet reasoning performance enhancement for repeated 500 axioms addition to our model ontologies. All times are shown in milliseconds. Implementation is achieved on a PC with Windows 10 3.8 GHz CPU, 32 GB RAM, and 2 TB HDD

Axioms	Regular reasoning	Incremental reasoning	Enhancement percentage
16000	490.9	587.9	0.0%
16500	548.9	390.2	0.289%
17000	614.4	446.1	0.274%
17500	692.6	502.2	0.275%
18000	839.5	592.7	0.294%
18500	988.8	720.6	0.271%

To enhance Jena inference, we use safe rules (also called Datalog rules) on top of RDF facts. This is to capture, in a uniform way, most of the OWL

constraints useful in practice, while guaranteeing a polynomial data complexity of reasoning and query answering.

6. Related Works

A. Corradi et al. [9] have presented the first well-developed approach that uses ontologies to support context-based access control. The authors employed the context to be the main principle for security policy specification and enforcement. They adopted an RDF-based semantics for context representation to handle heterogeneity of data representation. However, the authors do not extend the RDF-based semantics to cope with large vocabulary of the current environments and the developing status of sophisticated semantic-based tools. This was a limitation because the approach they proposed cannot infer the semantic relationships of entities represented by OWL, for example. A. Toninelli et al. [10] introduce the Proteus framework. The framework uses the contexts represented as description logic (OWL ontologies) and logical programming rules to enable the dynamic adaptation of policies by linking the requests to the data and the context. The proposed framework aimed to work in pervasive and ubiquitous environments. However, our approach differs from Proteus in the underlying formalism and the semantics of requests. Based on Proteus framework, P. Bellavista and R. Montanari [8] proposed an implementation for IoT adaptive context-based access authorizations. However, using only OWL ontologies encounters difficulties with those policies that require the definition of variables. For example, the use case of ”same location policy”, that grants access to files only when location(user) = location(file) is a good example that illustrates the need for variables [15]. L. Seitz et al. [7] described a framework for authorization and access control on IoT, in the context of interconnected systems consisting of resource-constrained devices not directly operated by humans. Their approach, however, does not support context nor semantic-web technologies and works as an add-on for XACML which makes it difficult to adopt for dynamic environments because evaluating XACML policies is too heavyweight for constrained devices and this is why the authors of [7] made the authorization decision process external to XACML policies. M. Hilia et al. [18] present a semantic-based authorization approach for controlling access in collaborative cloud environments. Their approach is also based on the XACML architecture and makes access decision according to contextual situations. Authorization context is evaluated by XACML Engine. The contextual information is retrieved from several sources, i.e. by using attribute finders to dynamically search, and perform dynamic queries for the environmental values of these attributes. However, approaches in [6], [7], [18] highlight extending the well-known ABAC XACML framework with semantic-based context. M. J. Covington and M. R. Sastry [19] presented a contextual attribute-based access control (ABAC) model which was realized in mobile applications. They used contexts as add-ons for the ABAC. The closest to our work was proposed by P. Das et al. [5] They presented a method for context-sensitive access control for IoT that uses reasoning over contextual ABAC. One of the main advantages of ABAC is that requesters do not have to be known a priori by targets, providing a higher level of flexibility for open environments, compared to RBAC models. Nevertheless, in ABAC everyone must agree on a set of attributes and their meaning when using ABAC, which is not easy to accomplish, especially in distributed dynamic networks.

Our approach, however, differs from their approach in that we decouple the context processing from access control operations. If our Context-Based Access Control (CBAC) engine needs a context related to an access request, it asks the context manager for it. The response from the context manager will impact the access authorization operations. In our approach, we aim at using light-weighted OWL ontologies for representing contexts and access control policies and to benefit from the full-blown OWL DL-based (incremental) reasoner Pellet [20] for getting inferences.

7. Conclusion

In this paper we have proposed an approach to support context-based access control in the dynamic environment like IoT. We have modeled the context-based access control using lightweight OWL ontologies CTX-Lite and CBAC. These ontologies represent the formal specifications of the contexts and the CBAC. We adopted DL and LP as a reasoning approach. We also have implemented a prototype of our approach to show its applicability. All access authorization operations proposed in this paper are specified based on the contexts of the communicating entities and determined at the time when the access requests are made. We give a complexity analysis of our framework and propose some enhancements for the framework.

Our ongoing work addresses incorporating obligation and trust management into CBAC.

8. References

[1] Y. Andaloussi, M. El Ouadghiri, Y. Maurel, J. Bonnin, and H. Chaoui. Access control in iot environments: Feasible scenarios. Procedia computer science, 130:1031–1036, 2018.

[2] A. S. Kayes, J. Han, and A. Colman. A semantic policy framework for context-aware access control applications. In 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp 753–762, July 2013.

[3] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT '08, pp 113–122, New York, NY, USA, 2008. ACM.

[4] M. Trnka and T. Cerny. Context-aware role-based access control using security levels. In Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems, RACS, pp 280–284, New York, NY, USA, 2015. ACM.

[5] P. K. Das, S. N. Narayanan, N. K. Sharma, A. Joshi, K. P. Joshi, and T. Finin. Context-sensitive policy-based security in internet of things. In 2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016, St Louis, MO, USA, May 18-20, 2016, pp 1–6, 2016.

[6] A. Dersingh, R. Liscano, and A. Jost. Context-aware access control using semantic policies. Ubiquitous Computing and Communication Journal (UBICC), pp 19–32, 2008.

[7] L. Seitz, G. Selander, and C. Gehrmann. Authorization framework for the internet-of-things. In IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks", WoWMoM 2013, Madrid, Spain, June 4-7, 2013, pp 1–6, 2013.

[8] P. Bellavista and R. Montanari. Context awareness for adaptive access control management in IoT environments. Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications, pages 157–178, 2017.

[9] Corradi, R. Montanari, and D. Tibaldi. Context-based access control for ubiquitous service provisioning. In 28th International Computer Software and Applications Conference (COMPSAC 2004), Design and Assessment of Trustworthy Software-Based Systems, 27-30 September 2004, pp 444–451, 2004.

[10] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. Proteus: A semantic context-aware adaptive policy model. In 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2007), 13-15 June 2007, Bologna, Italy, pp 129–140, 2007.

[11] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi. The description logic handbook: Theory, implementation and applications. Cambridge university press, 2003.

[12] A. Grigoris and H. Frank van. A Semantic Web Primer, 2Nd Edition (Cooperative Information Systems). The MIT Press, 2nd edition, 2008.

[13] P. Hitzler, M. Kr¨otzsch, and S. Rudolph. Foundations of Semantic Web Technologies. Chapman and Hall/CRC Press, 2010.

[14] M. AL-Wahah and C. Farkas. Context-aware IoT authorization: A dynamic and adaptive approach. In 13th International Conference for Internet Technology and Secured Transactions. (ICITST-2018), pp. 64-72.

[15] C. Gutierrez, C. Hurtado, A. Mendelzon, and J. P´erez. Foundations of semantic web databases. Journal of Computer and System Sciences, 77(3):520–541, 2011.

[16] H. Ter Herman. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. Web Semantics: Science, Services and Agents on the World Wide Web, 3(2-3):79–115, 2005.

[17] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. J. Web Sem., 5(2):51–53, 2007.

[18] M. Hilia, A. Chibani, T. Winter, and K. Djouani. Semantic based authorization framework for multi-domain collaborative cloud environments. volume 109, pp 718–724. Elsevier, 2017.

[19] M. Covington and M. R. Sastry. A contextual attribute-based access control model. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", pp 1996–2006. Springer, 2006.

