Context Delegation for Context-Based Access Control

Mouiad Al-Wahah(✉) and Csilla Farkas

University of South Carolina, Columbia, SC 29208, USA
malwahah@email.sc.edu, farkas@cec.sc.edu

Abstract. The capability to delegate access privileges is an essential component of access control policies. We present an ontology-based context delegation approach for context-based access control. Our approach provides a dynamic and adaptive context delegation capability. The delegation does not cause any change to the underlying access control policy. We use Description logic (DL) and Logic Programming (LP) technologies for modeling contexts, delegation and CBAC privileges. We show how semantic-based techniques can be used to support adaptive and dynamic context delegation for CBAC policies. We provide the formal framework of the approach and show that it is decidable and consistent.

Keywords: Security · Access control · Authorization · Delegation · Description logics · OWL ontology

1 Introduction

Delegation of the privileges is an important mechanism to support dynamic and adaptive access control in real world applications. There is a significant previous work on Context-Based Access Control (CBAC) [1–6, 10, 15]. However, support to delegate CBAC privileges is limited. For example, approaches described in [1–4, 15] do not provide any delegation services. Most of the existing delegation methods are based on traditional access control models, such as Role-Based Access Control (RBAC) models [6, 7]. Methods such as attribute-based delegation [8, 9] and capability-based delegation [10, 11] require that the underlying access control policy is changed. Moreover, none of the methods address the issue of context delegation when the access authorization is a context-dependent.

We propose a context delegation approach for CBAC policies. Our approach is grounded in semantic web technologies, specifically, Web Ontology Language (OWL) ontologies [17, 18], Semantic Web Rule Language (SWRL) [12] and Pellet reasoner [13]. The main advantages of using OWL-based technologies to represent access control are as follows: OWL ontologies provide formal framework since they are based on Description Logics. XML documents [14], for example, lack the formal semantics. OWL ontologies can encompass any XML representation or a Resource Description Framework (RDF) ontology. Finally, OWL-DL ontologies have the expressivity of DLs and the properties of completeness

C. Alzate et al. (Eds.): ECML PKDD 2018 Workshops, LNAI 11329, pp. 201–210, 2019.
https://doi.org/10.1007/978-3-030-13453-2_16

and decidability. OWL-DL reasoning can be provided by open-source reasoners, such as Pellet [13]. Using SWRL rules permits the use of dynamic variables that can not be determined during ontological policy specification. In our approach, SWRL rules are used to instantiate and validate the value of these variables at runtime.

The main contributions of our approach are: (1) Our method provides dynamic and adaptive context delegation that does not modify the original access control policy. (2) Our approach can be adopted by existing CBAC systems which do not provide delegation services. (3) Our semantic-based delegation model supports capabilities such as checking the access control and delegation policies for conflict and consistency, explaining inferences and helping to instantiate and validate the variables in dynamic environments.

The remainder of this paper is organized as follows: in Sect. 2, we present the context-based access control system modeling. Section 3 is dedicated to semantic-based context delegation, and in Sect. 4 we conclude with suggestions for future work.

2 Context-Based Access Control System Modeling

In this section, we give a brief overview of the Context-Based access control.

"Context" has been defined by Dey et al. [16] as "any information that is useful for characterizing the state or the activity of an entity or the world in which this entity operates." In CBAC, the system administrator (or resource owner) specifies a set of contexts and defines for each context the set of applicable privileges. When an entity (a user) operates under a certain context, (s)he acquires the set of privileges (if any) that are associated with the active context. When (s)he changes the active context, the previous privileges are automatically revoked, and the new privileges acquired [5]. Hence, the Context plays a crucial role in evaluating the access privileges.

2.1 Context-Based Access Control Model

Access requests are evaluated based on the contexts associated with the subject and the requested. The request is matched with context metadata that specify and activate the policy rule that to be enforced. We use rule-based Logic Programming (LP) to encode context and policy rules.

(**Access Control Policy (ACP) Rules**): Access control policy rule is given as a 6-tuple $\langle s, sc, r, rc, p, ac\rangle$, where $s \in$ **Subject**, $r \in$ **Resource**, sc, $rc \in$ **Context**, where sc is the subject's context and rc is the resource context, $p \in$ **Permission**$=\{$"*Deny*", "*Permit*"$\}$, and $ac \in$ **Action** $= \{read, write, delegate, revoke\}$. Each rule is instantiated by an access request, using the model ontologies and rules, and is evaluated at runtime to reach a decision.

(**Access Request (AR)**): Access request is given as a triple $\langle s, r, ac\rangle$, where $s \in$ **Subject**, $r \in$ **Resource**, $ac \in$ **Action**.

For example, an access request denoted as $ar = \langle s, r, \text{“}read\text{”}\rangle$, represents the case when subject s is requesting a “*read*” access to a resource r. The policy engine requests the contexts of s and r, and evaluates the permission p for the request ar. Assume the contexts of s and r are sc and rc, respectively. If using the contexts sc and rc, the policy engine can derive a permission, i.e., p is $+$, and there is no conflict, it grants the access permission for the request. Otherwise, it denies the request.

2.2 Ontology-Based Context Model

To model the context, we adopt a Description Logic (DL)-based method that partially resembles the method adopted by Bellavista and Montanari [15]. However, our context representation differs than that adopted by [15]. They have tightly coupled the subject's context (they call it the requestor context), the resource's context, the environmental context and the time context in one context (protection context). In our model, the subject's context and resource's context are separated. To support context delegation, we modify the subject's context only. We represent our model using the OWL-DL ontologies, the reader is referred to [17] and [18] for additional description on the current OWL standard.

Our context model is built around the concept of contextual attribute, information which models contextual attributes of the physical/logical environment such as location and temperature. Specific context subclasses can be represented under Generic Concept *Context*. Each subcontext class consists of attribute values and constants. In our model, the generic context of the subject is given by the following DL axiom:

$$\begin{aligned} SContext \equiv\ & Context \sqcap (User \sqcap \exists hasID.IDentity \sqcap \exists hasRole.Role \\ & \sqcap \exists hasGroup.Group) \sqcap (Environment \sqcap \exists hasLocation.Location) \\ & \sqcap (TElement \sqcap \exists hasTime.Time\_Interval) \sqcap \exists hasID.Identifier \end{aligned}$$

A context of *OnDutyNurse*, is represented as follows:

$$\begin{aligned} & OnDutyNurse \equiv Context \sqcap (User \sqcap \exists hasID.IDentity \sqcap \exists hasRole \\ & .Role\{Nurse\} \sqcap \exists hasGroup.Group\{InShiftNurses\}) \sqcap \\ & (Environment\{WorkingEnvironment\} \sqcap \exists hasLocation.Location \\ & \{Hospital\}) \sqcap (TElement\{WorkingTime\} \sqcap \exists hasTime\{xsd : dateTime \\ & [\geq 2018-04-06T09:00:00, \leq 2018-04-06T17:00:00]\}) \sqcap \exists hasID.\{0\} \end{aligned}$$

Note that the concept *OnDutyNurse* includes all the characteristics specifications of the generic concept *SContext*. We call this context a *reference context*. It holds the high-level context of an entity which 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 similar to 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 describes a certain user context at 2018-04-06T14:23:00, which represents 2:23 pm on April 6, 2018:

$$OnDutyNurse\{Ann\} \equiv Context \sqcap (User\{Ann\} \sqcap \exists hasID.IDentity\{Nurse505\}$$
$$\sqcap \exists hasRole.Role\{Nurse\} \sqcap \exists hasGroup.Group\{InShiftNurses\})\sqcap$$
$$(Environment\{WorkingEnvironment\} \sqcap \exists hasLocation.Location\{Hospital\})$$
$$\sqcap (TElement\{WorkingTime\} \sqcap \exists hasTime.Time\_Instance\{xsd : dateTime$$
$$[2018-04-06T14:23:00]\}) \sqcap \exists hasID.\{0\}$$

This concept states that Ann is $OnDutyNurse$ at time 2:23 pm on April 6, 2018, if she is a user, has a role of $Nurse$, belongs to a group that is called $InShiftNurses$, within a $WorkingEnvironment$, at location $Hospital$ and during the $WorkingTime$.

The context ontology is flexible. It can be extended or shrinked by adding or removing subcontexts or by adding or removing contextual attributes to the subcontexts.

3 Semantic-Based Context Delegation

The purpose of delegation is to grant/transfer access privileges from one entity, the delegator, to another entity, the delegatee. We require that the delegator must have the access privilege that is associated with context to be delegated. Delegating a subset of contextual attributes may result in a number of problems. These problems:

- Colluding [8], i.e., two entities may satisfy a policy that they could not if they acted individually. We do not address this problem in this paper.
- Inconsistent policy, i.e., the delegated privileges are conflicting the user's original privileges. Our approach avoids inconsistent policies by evaluating delegator's context together with the delegatee's context.

At the time of delegation, the delegator must have the context c that is to be delegated to the delegatee. After the delegation is successfully completed, delegatee can use the delegated context and the privilege(s) associated with it to access to a resource r Our approach imposes constrains on context delegation. The constraints may be specified by the delegator or the system security officer. These constraints further restrict the delegation. Intuitively, if the delegatee's context satisfies the constraints, then the delegation is permitted. Otherwise, the delegation will be aborted. Our model architecture is shown in Fig. 1.

(**Delegation Request (DR)**): Delegation request is given as a 6-tuple $\langle s_1, s_2, r, ac, \mathbf{DCs}, \mathbf{Par}\rangle$, where s_1, $s_2 \in$ **Subject** and they represent the delegator and delegatee, respectively. $r \in$ **Resource**, the resource to make the

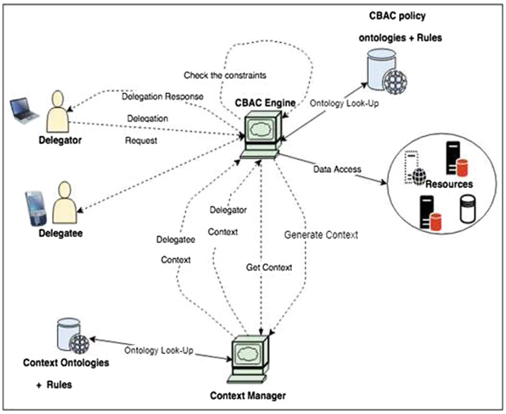

Fig. 1. The proposed system architecture.

delegation over, $ac \in$ **Action**, the action and must be equal to "*delegate*", **DCs** $\subseteq$ **Constraint** represents the set of constraints imposed by the delegator on delegatee's context, and **Par** is a finite set of delegation parameters, other than the delegation constraints, which are specified by the delegator. Delegation parameters, **Par**, are given by:

$$\mathbf{Par} = (n_1, v_1), ..., (n_m, v_m) \tag{1}$$

where n_i represents the parameter $name_i$ and v_i is the value of this parameter.

The **DCs** are represented as a set of pairs:

$$\mathbf{DCs} = (CA_1, Cons_1), ...(CA_n, Cons_n) \tag{2}$$

where CA_i represents an attribute i and $Cons_i$ is the delegation constraints set i (if any) that is imposed over CA by the delegator and must be satisfied by the delegatee's contextual attributes.

3.1 Delegation Policies

Every delegation operation is subject to predefined delegation policies. Delegation policies are rules that restrict the delegation. We represent our delegation policies in a predicate form as follows:

$can\_delegate(s_1, c_1, s_2, c_2, Grant, \mathbf{DCs})$: subject s_1 can delegate context c_1 to subject s_2 if s_2's context satisfies delegation constraints **DCs**.

$can\_delegate(s_1, c_1, s_2, c_2, Transfer, \mathbf{DCs})$: subject s_1 can delegate context c_1 to subject s_2 if s_2's context (the reference context) satisfies delegation constraints **DCs**.

$can\_revoke(s_1, s_2, c_1, casCaded)$: subject s_1 can revoke the delegated context c_1 from s_2 if s_1 is authorized to do so, i.e., it was the delegator of c_1. Note that, the issue of cascading revoke has been studied extensively and we do not address this issue in this paper.

3.2 Delegation Operations

We assume that each delegation operation delegates only one context at a time. If the delegator has multiple contexts (one is the instantiated context and the others may be gained by previous delegations) and (s)he wishes to delegate more than one context to the same delegatee, (s)he can do that in multiple delegation operations. The delegation operation takes the form $delegate(s_1, c_1, s_2, c_2, Grant, \mathbf{Par})$.

Figure 1 shows our approach architecture. Delegator s_1 delegates context c_1 to delegatee s_2. After checking delegation constraints satisfaction as we have illustrated in the previous subsection, the delegation algorithm (see Algorithm 1.) creates a delegation instance with an identifier del_{id}. The delegation instance gets part of its values from the delegation request, namely from **Par** and **DCs**. We define the following parameters, $MaxDepth$ is the depth of the delgation. It specifies the number of times the context can be delegated. This value is set by the first delegator ($isSoA = true$, see Fig. 3). The $isDelegatable$ is a Boolean value that determines whether the context is delegatable. If $isDelegatable = false$, then the algorithm automatically sets $MaxDepth$ to 0.

3.3 Delegation Constraints

We represent delegation constraints, denoted as $Cons$, using Semantic Web Rule Language safe rules (SWRL-safe). SWRL combines OWL ontologies with Horn Logic rules, extending the set of OWL axioms to include Horn-like rules. SWRL rules have the syntax **Antecedent**− >**Consequent**, where each **Antecedent** and **Consequent** consists of atoms. These atoms can be of the form $C(x)$, $P(x, y)$, $sameAs(x, y)$ or $differentFrom(x, y)$, where C is an OWL class, P is an OWL property, and x, y are either variables, OWL individuals or OWL data values. The **Consequent** atom will be true if all atoms in the **Antecedent** are true.

For example, suppose that Ann has $OnDutyNurse$ as a reference context as has been shown in Sect. 2.2. Now suppose Ann wants to set delegation constraint on the $time$ contextual attribute before delegating her context (her reference context) to another user, $Alice$. $Alice$ is a lab analyst and she has the following reference context:

$$OnDutyAnalyst \equiv Context \sqcap (User \sqcap \exists hasID.IDentity \sqcap \exists hasRole$$
$$.Role\{LabAnalyst\} \sqcap \exists hasGroup.Group\{InShiftAnalysts\})\sqcap$$
$$(Environment\{WorkingEnvironment\} \sqcap \exists hasLocation.Location$$
$$\{Lab\}) \sqcap (TElement\{WorkingTime\} \sqcap \exists hasTime\{xsd : dateTime$$
$$[\geq 2018-04-06T09:00:00, \leq 2018-04-06T17:00:00]\}) \sqcap \exists hasID.\{0\}$$

The delegation constraint is ($01:00pm \geq time \geq 10:00am$), that is, it can only be delegated between 10:00 am and 01:00 pm. At the time of delegation, $Alice$ has an active context as shown below:

$$OnDutyAnalyst\{Alice\} \equiv Context \sqcap (User\{Alice\} \sqcap \exists hasID.IDentity\{Analyst705\}$$
$$\sqcap\, \exists hasRole.Role\{LabAnalyst\} \sqcap \exists hasGroup.Group\{InShifAnalyst\}) \sqcap$$
$$(Environment\{WorkingEnvironment\} \sqcap \exists hasLocation.Location\{Lab\})$$
$$\sqcap\, (TElement\{WorkingTime\} \sqcap \exists hasTime.Time\_Instance\{xsd : dateTime$$
$$[2018-04-06T12:30:11]\}) \sqcap \exists hasID.\{0\}$$

The policy engine checks, then, if the delegation constraints are satisfied or not. The policy engine uses the following SWRL rule to check the time constraint:

$$TimeCons(?t_3) \wedge notBefore(?t_3, ?cons1) \wedge swrlb : greater$$
$$ThanOrEqual(?cons1, 10:00) \wedge notAfter(?t_3, cons2) \wedge swrlb :$$
$$lessThanOrEqual(?cons2, 01:00) -> satisfied(?t_3),$$

where $t_3 = Time\_Instance$ is extracted from Alice's active context and is equal to $12:30:11\ pm$ (on April 6, 2018), and the constraints $cons1 = 10:00\ am$ and $cons2 = 01:00\ pm$ from the delegation constraints set by Ann.

3.4 Processing Delegation Request

Algorithm 1. illustrates the process of context delegation. The approach proceeds as follows:

- The delegator prepares a delegation request and sends it to the policy engine.
- The policy engine parses the request and starts the delegation process.
- The policy engine extracts the delegation constraints, asks the context manager for the delegator's context, and checks if the delegator has the delegation right.
- If the delegator is authorized, the policy engine asks the context manager for the delegatee's (s_2) context and checks for satisfiability of the delegation.
- If the delegation is satisfiable, the policy engine creates a delegation instance, see Fig. 2, using the delegation ontology and the parameters specified in the delegator's delegation request.
- The policy engine sends a request to the context manager, accompanied with a delegation identifier, del_{id}, to construct a generated context for s_2. This context is a copy of the delegator reference context but it is associated with the delegatee.
- The context manager creates the generated context for s_2 and associates it with the identifier del_{id} provided by the policy engine with the request.
- The delegatee has two contexts, the instantiated context and the generated context.

```
input  : CBAC, Del, Ctx are CBAC, delegation, and context Ontologies. RQ is an
         Access Request
output : UCtx, UDel /* Updated context and Delegation onologies              */
RT ← parse(RQ);
if RT = AR then
    /* It is an access request                                               */
    eval(RT);
    exit();
end
else
    /* It is a delegation request*/;
    ⟨s1, s2, r, ac, DCs, Par⟩ ← dismantle(RT);
    sc1 ← getContext(s1);
    if isAuthorized(s1, sc1, r) = false then
        output("s1 is not authorized to access r");
        exit();
    end
    sc2 ← getContext(s2);
    CAs ← extractCAs(sc2);
    T ← checkSatisfiability(DCs, CAs);
    if T = false then
        output("The context is not delegatable");
        exit();
    end
    else
        UDel ← createDelegationinstance(Del, ⟨s1, s2, r, ac, DCs, Par⟩, del_id);
        UCtx ← createContext(Cx2, Ctx, del_id);
        return(UDel, UCtx);
        exit();
    end
end
```

Algorithm 1. Context Delegation

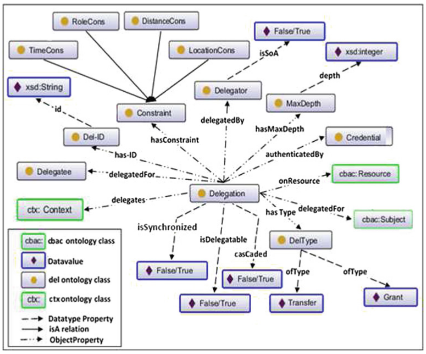

Fig. 2. Delegation ontology.

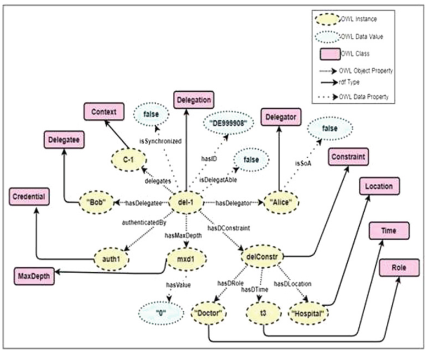

Fig. 3. Delegation instance for Bob.

Example. Suppose that we have the following policy rule: $\langle s$,c_1, "*Ann Health Record*", *Nil*, +, "*read*"$\rangle$ and that c_1 is given by the DL axiom:

$$c_1 \equiv User(Alice) \sqcap \exists hasRole(Analyst) \sqcap \exists hasTime(t_1) \sqcap \exists hasLocation (HosptialLab) \sqcap \exists hasActivity(Working)$$

Assume also that the contextual attribute t_1 has a constraint, **Constraint**, $(08:0\,am \leq t_1 \leq 05:0\,pm)$ and *Alice*'s context satisfies this constraint. Assume now *Alice* intends to delegate her context c_1 to *Bob* from 10:00 *am* to 01:00 *pm* and this context is not delegatable. Bob has the following context:

$$c_2 \equiv User(Bob) \sqcap \exists hasRole(Doctor) \sqcap \exists hasTime(t_2) \sqcap \exists$$
$$hasLocation(Hosptial) \sqcap \exists hasActivity(Working)$$

The contextual attribute t_2 has the constraint $(09:0AM \leq t_2 \leq 03:0PM)$. To delegate context c_1 to *Bob*, *Alice* prepares a delegation request which has the form:

$\langle Alice, Bob, Ann's\ Health\ Record,$ "*delegate*", $\langle Time, (10:0AM \leq t_3 \leq 01:0PM)\rangle\rangle$

Alice sends the delegation request to the policy engine. The policy engine asks the context manager for *Bob*'s context and checks for satisfiability of the delegation. If the delegation is satisfiable, the policy engine creates a delegation instance del_1 with the entities shown in Fig. 3. The new context is similar to *Alice*'s context except that it is associated with *Bob*.

4 Conclusion and Future Work

In this paper we have proposed an approach for context delegation for context-based access control policies. The approach provides dynamic and adaptive mechanism for privilege delegation and does not cause any change to the underlying access control policy. The approach presented in this paper is modeled using semantic-based technologies and can be used by existing CBAC systems which do not provide delegation capability. We have implemented the model using real networks. We are working on extending our model by using RESTful web services with Java (Jersey/JAX-RS). The ontologies and some related preliminary coding can be found on (https://github.com/Mouiad1975/Context-Delegation).

References

1. Bhatti, R., Bertino, E., Ghafoor, A.: A trust-based context-aware access control model for web-services. Distrib. Parallel Databases **1**(18), 83–105 (2005)
2. Toninelli, A., Montanari, R., Kagal, L., Lassila, O.: A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In: Cruz, I., et al. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 473–486. Springer, Heidelberg (2006). https://doi.org/10.1007/11926078_34
3. Kulkarni, D., Tripathi A.: Context-aware role-based access control in pervasive computing systems. In: 13th ACM Symposium on Access Control Models and Technologies, pp. 113–122. ACM, Estes Park (2008)
4. Shen, H., Cheng, Y.: A semantic context-based model for mobile web services access control. Int. J. Comput. Netw. Inf. Secur. **3**(1), 18–25 (2011)

5. Corrad, A., Montanari, R., Tibaldi, D.: Context-based access control management in ubiquitous environments. In: 3rd IEEE International Symposium on Network Computing and Applications, pp. 253–260. IEEE Computer Society, Washington (2004)
6. Trnka, M., Cerny, T.: On security level usage in context-aware role-based access control. In: 31st Annual ACM Symposium on Applied Computing, pp. 1192–1195. ACM, Pisa (2016)
7. Zhang, L., Ahn, G.J., Chu, B.: A rule-based framework for role based delegation. In: Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies, pp. 153–162. ACM, Chantilly (2001)
8. Servos, D., Osborn, S.L.: Strategies for incorporating delegation into attribute-based access control (ABAC). In: Cuppens, F., Wang, L., Cuppens-Boulahia, N., Tawbi, N., Garcia-Alfaro, J. (eds.) FPS 2016. LNCS, vol. 10128, pp. 320–328. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51966-1_21
9. Servos, D., Osborn, S.L.: Current research and open problems in attribute-based access control. ACM Comput. Surv. **4**(49), 1–65 (2017)
10. Kagal, L., Berners-lee, T., Connolly, D., Weitzner, D.: Self-describing delegation networks for the web. In: 7th IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 205–214. IEEE Computer Society, Washington (2006)
11. Gusmeroli, S., Piccione, S., Rotondi, D.: A capability-based security approach to manage access control in the Internet of Things. Math. Comput. Model. **5**(58), 1189–1205 (2013)
12. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: a semantic web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium (2004)
13. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, A.: Pellet: a practical OWL-DL reasoner. Web Seman.: Sci. Serv. Agents World Wide Web **2**(5), 51–53 (2007)
14. Parmar, V., Shi, H., Chen, S.-S.: XML access control for semantically related XML documents. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pp. 10–19 (2003)
15. Bellavista, P., Montanari, A.: Context awareness for adaptive access control management in IoT environments. Secur. Priv. Cyber-Phys.Syst.: Found. Princ. Appl. **2**(5), 157–178 (2017)
16. Dey, A., Abowd, G., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum.-Comput. Interact. **2**(16), 97–166 (2001)
17. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman and Hall/CRC Press, New York (2009)
18. The W3C OWL Homepage. https://www.w3.org/OWL/. Accessed 4 Feb 2018