Author:	Kayes, A. S. M., Han, J. & Colman, A.
Title:	An ontology-based approach to context-aware access control for software services
Conference name:	14th International Conference on Web Information System Engineering (WISE 2013)
Conference location:	Nanjing
Conference dates:	13-15 October 2013
Series title and volume:	Lecture notes in computer science, vol. 8180
Place published:	Berlin
Publisher:	Springer
Year:	2013
Pages:	410-420
URL:	http://doi.org/10.1007/978-3-642-41230-1_34
Copyright:	Copyright © Springer-Verlag Berlin Heidelberg 2013.

An Ontology-Based Approach to Context-Aware Access Control for Software Services

A. S. M. Kayes, Jun Han, and Alan Colman

Faculty of Information and Communication Technologies
Swinburne University of Technology, VIC 3122, Australia
{akayes,jhan,acolman}@swin.edu.au

Abstract. In modern communication environments, the ability to provide access control to services in a context-aware manner is crucial. By leveraging the dynamically changing context information, we can achieve context-specific control over access to services, better satisfying the security and privacy requirements of the stakeholders. In this paper, we introduce a new *Context-Aware Access Control (CAAC) Framework* that adopts an ontological approach in modelling dynamic context information and the corresponding CAAC policies. It includes a *context model* specific to access control, capturing the relevant low-level context information and inferring the high-level implicit context information. Using the context model, the *policy model* of the framework provides support for specifying and enforcing CAAC policies. We have developed a prototype and presented a healthcare case study to realise the framework.

Keywords: Context-Awareness, Context-Aware Access Control, Context Model, High-Level Context, Access Control Policy.

1 Introduction

The rapid advancement of computing technologies has led to the computing paradigm shift from fixed desktop to context-aware environments, as described by Weiser [12]. Such a shift brings with it opportunities and challenges. On the one hand, users demand access to services in an anywhere, anytime fashion. On the other hand, such access has to be carefully controlled due to the additional challenges coming with the dynamically changing environments, so as not to compromise the relevant security and privacy requirements. Such new challenges require new Context-Aware Access Control (CAAC) approaches. In general, the information about the changing environment, called context information [7], needs to be taken into account when making access control decisions.

A number of context-sensitive access control approaches have highlighted the importance and use of context information in the access control processes (e.g., [3],[5],[6],[8],[11]). However, the existing approaches to access control have only considered specific types of context information. There is still a lack of a general context model specific to access control, to capture the basic context information, and infer richer information from other information in a comprehensive manner. In addition, an appropriate access control policy model that incorporates contexts into access control decision making is required.

Towards this goal, in this paper we introduce a new Context-Aware Access Control (CAAC) approach to provide context-aware access control support for software services. It makes the following key contributions.

First, we propose a *context model*, in order to represent and capture different types of context information in a systematic way; and to reason about high-level implicit context information that is not directly available but can be derived from other information. The context model uses the ontology language OWL, extended with SWRL for inferring implicit context with user-defined rules. ***Second***, we introduce a *policy model* for defining and enforcing access control policies that take into account relevant contexts from the context model. The policy model also uses OWL and SWRL. The main novel point in our policy model is that it provides context-aware access control decisions. In particular, it has reasoning capability which grants the right access to the appropriate parts of a resource (fine-grained access control to resource) by the appropriate users, considering the different granularity levels of the resource. Other than the above two major contributions, we have developed a *prototype framework* implementing the approach and carried out a *case study* in the healthcare domain.

The rest of this paper is organized as follows. Section 2 presents an application scenario to motivate our work. Section 3 discusses the design of our context model for access control. Section 4 presents a policy model for context-aware access control. The applicability of our approach is investigated in Section 5. Section 6 discusses related work. Finally, Section 7 concludes the paper.

2 Motivation and General Requirement

Motivating Scenario. Let us consider an application scenario from the healthcare domain, requiring context-aware access control.

Scene #1: *The scenario begins with patient Bob who is in the emergency room due to a heart attack. While not being Bob's usual treating physician, Jane, a medical practitioner of the hospital, is required to treat Bob and needs to access Bob's emergency medical records from the emergency room.*

Scene #2: *After getting emergency treatment, Bob is shifted from the emergency room to the general ward of the hospital and has been assigned a registered nurse Mary, who has regular follow-up visits to monitor his health condition. Mary needs to access several types of Bob's records (daily medical records and past medical history) from the general ward.*

Basic Analysis. The context information describing the context entities relevant to access control in the above scenario includes: the *identity/role* of the Users, the *location* of the Users, the *relationship* between the User and Patient, the *health status* of the Patient, etc. To provide fine-grained access control and grant the right access to the appropriate parts of a resource by the appropriate users, the resource (patient medical records) needs to be considered in a hierarchical manner. Furthermore, access to the resource and its components at different levels of granularity can be managed in a service oriented manner, i.e., service wrapper operations can be defined them and invoked by the users as permission allows. For example, the write operation on the emergency medical

Table 1. An Informal Access Control Policy for Our Example Application

No	Policy
1	A medical practitioner, who is a treating physician of a patient, is allowed to read/write the patient's emergency medical records in the hospital. However, in an emergency situation (like the above), all medical practitioners should be able to access the emergency medical records from the emergency room of the hospital.

records can be defined as *writeEMR()*. In this way, fine-grained access control to resources can be easily realized by managing the access to the service operations.

Concerning the above two scenes, one of the relevant access control policy is shown in Table 1. The policy is based on a set of constraints on the user role and service (a *service* can be seen as a *pair* $<a, r>$ with r being a requested resource and a being the action/operation on the resource), and the policy refers to need to be evaluated in conjunction with the relevant contexts.

General Requirement. As different types of contexts are integrated into the access control processes, some important issues arise.

(R1) ***Representation of context entities and information:*** *What access control-specific context entities and information should be considered as part of building a context model specific to CAAC?* Furthermore, *how to model and capture the context entities and information in an effective way?*

(R2) ***Inferring high-level context information:*** *How to express user-defined reasoning rules, in order to capture knowledge which is only implicitly present, thereby extending the context model?*

(R3) ***Enforcement of access control policies:*** *How to specify and evaluate the access control policies based on the relevant contexts to realize a flexible and dynamic access control scheme?*

3 Context Model - COAC Context Ontology

Many researchers have attempted to define the concept of context. According to Dey [4], the context entities are *Person*, *Place* and *Object* and he defines context as *any information that can be used to characterize the situation of an entity.* We specialize Dey's definition of context to cover access control applications.

Definition: Context Information and Context-Awareness. *Context Information* used in an access control decision is defined as any relevant information about the state of a relevant entity or the state of a relevant relationship between different relevant entities at a particular time. *Context-awareness* relates to the use of this context information for access control decision making.

Experience from existing research (e.g., [9]) shows that ontologies are very suitable for modelling context for pervasive computing applications. To meet requirements (1) and (2), we in this section propose an ontology-based context model, named Context Ontology for Access Control (COAC).

Design Considerations. Our COAC ontology, representing *context entity* and *information* as ontology elements, is capable of: *representing general (upper) concepts*, *representing domain-specific concepts*, and *supporting reasoning according to user-defined rules (to obtain high-level context information).*

To model context information, we adopt the OWL language, which has been the most practical choice for most ontological applications because of its considered trade-off between computational complexity and expressiveness [9].

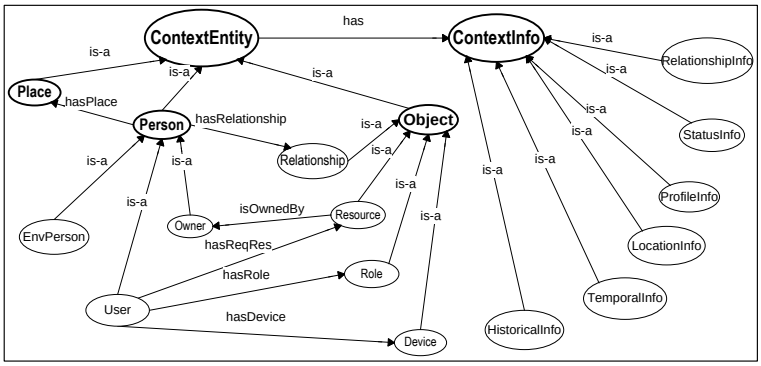

Fig. 1. COAC Upper Context Ontology - *Context Entity* and *Context Information*

Ontology-based reasoning using Description Logic (DL) may not always be sufficient to infer the high-level implicit contexts. The expressivity of OWL can be extended by adding SWRL rules to an ontology. We express the user-defined reasoning rules using the SWRL rule language which provide the ability to infer additional information in our COAC context ontology.

Upper Context Ontology. Our *COAC Upper Ontology* is the main ontology having *classes* modelling *context entities* and *context information* and *relationships* between classes (*object* and *data type* properties).

The main part of the upper ontology is shown in Figure 1. We classify the access control-specific *context entities* into two groups: *core* and *environmental*. *User*, *Resource*, and resource *Owner* are the *core entities* as they are core concepts of access control. To offer the advantages of the RBAC role, which regulates access to services based on user roles rather than individual users, our model also has *Role* as a core entity. A *hasRole* object property is used to relate *User* and *Role* classes for representing the fact that a user has a role. *User* is linked to *Resource* by the *hasReqRes* object property for capturing a user's access interest in a resource. The relationship between a *Resource* and its *Owner* is captured by the property *isOwnedBy*. The *Relationship* between *Persons* is another class of core entity, has its own characterizing context information. An object property *hasRelationship* is used to link the *Persons* and the *Relationship*. Following Dey's general context entities [4], we consider *Role* and *Relationship* as special *Objects*.

The *environmental entities* are the other entities that are relevant to the access request. They include *EnvPerson* (a person who is neither a *User* nor an *Owner* but relevant), *Place*, and *Device*. The *Device* is the communication device that the *User* uses to issue the access request, and consequently the *User* is linked to *Device* through *hasDevice*. Furthermore, a *Person* is at a particular place and therefore is connected to *Place* with an object property *hasPlace*.

Focusing on the *context information* types that are relevant to making access control decisions, we have classified the context information into six categories, represented by six context information types (see Figure 1)): *RelationshipInfo*, *StatusInfo*, *ProfileInfo*, *LocationInfo*, *TemporalInfo*, and *HistoricalInfo* classes.

Based on the motivating scenario, we further classify relationship information into different types. For example, the *Person-Centric* relationship is a relation-

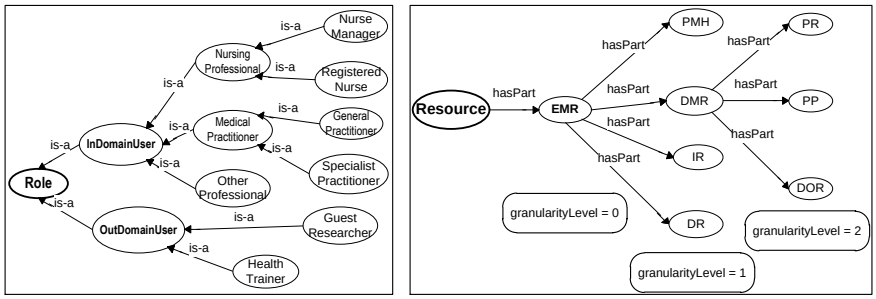

Fig. 2. Domain-Specific (a) *Role Ontology* (left), (b) *Resource Ontology* (right)

ship information type, which contains a data type property (*xsd:string* type) named *interRelationship*. It indicates the interpersonal relationship between the *Persons* concerned such as "doctor-patient". Due to page limit, the complete descriptions of other context types are not included in this paper.

Domain-Specific Context Ontologies. In the context of the motivating scenario, we define the *domain-specific ontologies* for the healthcare domain on the basis of the *COAC upper ontology*. In particular, we focus on the representation of the relevant *Role* and *Resource* ontologies due to space limitation.

The Role can be categorized as *InDomainUser* or *OutDomainUser* (Figure 2(a) shows a part of the *Role* ontology). This categorization is to facilitate different fine-grained control for different types (roles) of users. Additionally, this improved structure is beneficial from the reasoning viewpoint, as some services only relevant to the introduced generalizations of these roles, i.e., *InDomainUser* and *OutDomainUser* in the example. In the healthcare application, we model *InDomainUser* roles, relying on the Australian Standard Classification of Occupations (ASCO) of the health professionals. The *NursingProfessional*, *MedicalPractitioner*, and *OtherProfessional* are subclasses of *InDomainUser*; the *GeneralPractitioner* and *SpecialistPractitioner* are in turn subclasses of *MedicalPractitioner*; etc. Besides, some other members such as *GuestResearcher*, *HealthTrainer*, etc, are not healthcare members but need to access some of the patient information, and therefore classified under *OutDomainUser*. This superclass-subclass hierarchy can facilitate access control in a way similar to the RBAC's senior-junior role concept [10]. For example, a user playing the role *MedicalPractitioner* can access a patient's daily medical records, which means a user playing the role *GeneralPractitioner* or *SpecialistPractitioner*, is also permitted to access the patient's daily medical records. However, the converse is not true.

The different components at various granularity levels of a patient's medical record (*Resource*) are individually identifiable, so as to achieve fine-grained control over access to them. As such, there is the healthcare *Resource* (patient data) hierarchy in the domain ontology (Figure 2(b) shows a part of the *Resource* ontology). In formulating the structure of the patient medical record, we follow the Health Level Seven (HL7) standard. Emergency Medical Records (*EMR*, at *granularityLevel* 0) includes a patient's complete medical records: a patient's Daily Medical Records (*DMR*, at *granularityLevel* 1), which includes Physiological Records (*PR*), Physician Prescriptions (*PP*), Daily Observations Reports (*DOR*), etc., at *granularityLevel* 2; a patient's Past Medical History (*PMH*);

Table 2. User-Defined Reasoning Rules

No	Rule ($C_1 \wedge C_2 \wedge ... \wedge C_n \rightarrow C_1 \wedge C_2 \wedge ... \wedge C_r$, where each C_i is a rule concept.
Rule #1(a)	**Owner**(?o) ∧ StatusInfo(?hs) ∧ has(?o, ?hs) ∧ ProfileInfo(?hp) ∧ has(?o, ?hp) ∧ **bodyTemperature**(?hp, "normal") ∧ **heartRate**(?hp, "abnormal") → **healthStatus**(?hs, "critical") **// Rule #1(b)** ... ∧ ... → **healthStatus**(?hs, "normal")
Rule #2(a)	**User**(?u) ∧ **Role**(?rol) ∧ hasRole(?u, ?rol) ∧ swrlb:equal(?rol, "MedicalPractitioner_1") ∧ Resource(?r) ∧ hasReqRes(?u, ?r) ∧ **Owner**(?o) ∧ isOwnedBy(?r, ?o) ∧ **Relationship**(?re) ∧ hasRelationship(?u, ?re) ∧ hasRelationship(?o, ?re) ∧ **RelationshipInfo**(?rel) ∧ has(?re, ?rel) ∧ **ProfileInfo**(?pp) ∧ has(?u, ?pp) ∧ **userIdentity**(?pp, ?uID) ∧ **roleIdentity**(?pp, ?rolID) ∧ **ProfileInfo**(?sp) ∧ has(?o, ?sp) ∧ **conPeopleID**(?sp, ?cpID) ∧ **conPeopleRoleID**(?sp, ?cpRID) ∧ swrlb:notEqual(?cpID, ?uID) ∧ swrlb:notEqual(?cpRID, ?rolID) → **interRelationship**(?rel, "nonTreatingPhysician") **// Rule #2(b)** ... ∧ ... → **interRelationship**(?rel, "treatingPhysician")

Identification Records (*IR*); Demographic Records (*DR*); and so on. The *granularityLevel* is an important data type property (*xsd:int* type) in our model that regulates access to different parts of a resource individually. In the motivating scenario (Scene #1), for example, Jane can access Bob's *EMR*, i.e., including all other sub-components of the *EMR*, while Mary can only access *DMR*.

Context Reasoning. Our COAC ontology is extended with user-defined reasoning rules (specified in the SWRL language) to infer high-level implicit contexts (requirement (2)). This aims to improve the request for specific services through automated inference of implicit contexts from limited contexts.

A *user-defined reasoning rule* has the form: $\boldsymbol{C_1} \wedge \boldsymbol{C_2} \wedge \boldsymbol{...} \wedge \boldsymbol{C_n} \rightarrow \boldsymbol{C_1} \wedge \boldsymbol{C_2} \wedge \boldsymbol{...} \wedge \boldsymbol{C_r}$, where C_1, C_2, ... C_n are the input concepts (*body* of the rule) and C_1, C_2, ... C_r are the resultant/derived concepts (*head* of the rule).

Example Rules. The SWRL Rule #1(a) in Table 2 states that ***if*** the patient's *bodyTemperature* is "normal" and its *heartRate* is "abnormal" ***then*** his *healthStatus* is "critical". Rule #2(a) in Table 2 states that ***if*** a User (playing the *MedicalPractitioner* role) has requested access to a *Resource* which is owned by an *Owner* (a Patient) and the *Owner* is not connected with the *MedicalPractitioner* through a *interRelationship* ***then*** the association between the *User* and *Owner* is a "nonTreatingPhysician" relationship. Note that this rule has used the required (basic) information, the user's *personal profile* information and the patient's *social profile* information. For the scenario (Scene #1), based on the context ontology, this rule can determine that Jane and Bob has a "nonTreatingPhysician" relationship, because Jane is not assigned as the treating physician of patient Bob. Similarly, the SWRL rules can be used to derive *Location-Centric* relationships (e.g., *colocatedRelationship* between *User* and *Owner*).

4 Policy Model - CAPO Policy Ontology

The access control policies specify whether a *subject* is permitted or denied access to a set of target *objects* for their specific *action* or sequence of *actions*, when a set of *conditions* are satisfied. We refer to the user's role rather than explicitly name each user to reflect the way users can be grouped as the subject.

Various policy languages have been proposed in the literature. Our goal in this research is to provide a way in which CAAC policies can be specified by incorporating dynamic contexts. To be of practical use, it must be expressive enough to specify the policies in an easy and natural way. In particular, it needs to use the concepts from the context model, to specify under which conditions the requested resource is accessible. To do so, we use the same ontology-based

Table 3. An Example Access Control Policy

AccessPolicy (hasSubject, hasObject, hasAccessPerm, hasAction, hasAccessConds)
{
Subject.Role.roleID = "MP00X"; & // MedicalPractitioner's role identity
Object.granularityLevel = 0; & **Object**.privacyAttribute = 1; & // Emergency Records (EMR)
AccessCond.intRelationship(User, Owner) = "anyPhysician"; &
AccessCond.healthStatus(Owner) = "critical"; &
AccessCond.locAddress(User) = **AccessCond**.locAddress(Owner) = "ER00X"; &
→ **AccessPermission**.permission = "granted"; &
Action.actionType = "read" *or* "write".
}

languages (the OWL and SWRL) as the policy language. In the following, we present the two main parts of our policy model, supporting the requirement (3).

Policy Specification. Our *context-aware policy ontology (CAPO)*, as depicted in Figure 3, has the following concepts: *AccessPolicy*, *Subject*, *Role*, *Object*, *AccessPermission*, *Action*, and access conditions (i.e., *AccessCond*). Our policy model also uses the concepts (*ContextInfo*, shown in shaded ellipse) from the COAC context model, to identify and capture the relevant contexts at runtime.

A *CAAC Policy* captures the ***who/what/when*** dimensions which can be read as follows: an *AccessPolicy* specifies that a *Subject* (who is playing a *Role*) has *AccessPermission* ("granted" or "denied") to which parts (*privacy attributes*) of an *Object* (at which *levels of granularity*) for a specific *Action* ("read" or "write") or sequence of actions under which context-dependent *access conditions*.

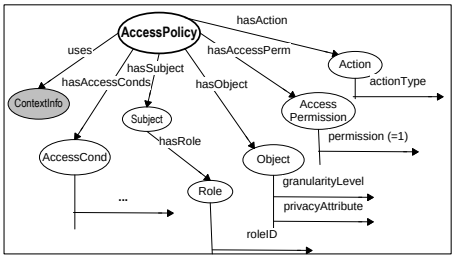

Fig. 3. CAPO Policy Ontology

Let us consider an access control policy for the medical practitioners (Policy #1 in our application scenario). The access decision is based on the following policy constraints: ***who*** the user is (subject's *role*), ***what*** resource being requested (object's *privacy attribute* and *granularity level*), and ***when*** the user sends the request (the *interpersonal relationship* between user and owner, their *locations*, the *health status* of the patient). The template of the policy in a readable form is shown in Table 3 and the specific policy states that all medical practitioners can access a patient's complete medical records when his health status is critical.

One of the main features of our policy model is its ability to specify access permissions at different levels of resource granularity. For example, in the above policy, all medical practitioners can access a patient's emergency records (EMR) at granularity level 0 (*highest* level), which means they also can access all other records at the *lower* granularity levels (i.e., all the sub-components of the EMR).

Policy Evaluation. During the evaluation phase, an *access query* is used to process the user's *access request*. We use the SWRL rules to evaluate the policies. In particular, the query language SQWRL, which is based on OWL and SWRL, is adopted to process service access requests.

To determine the access permissions on the requested services, an *Access Query* is formulated based on the access control policies, by capturing the *policy constraints* and *relevant contexts* that are currently in effect.

Table 4. A Simplified Version of an Access Query

AccessPolicy(?policy) ∧ Subject(?subject) ∧ Role(?role) ∧ Object(?resource) ∧ Action(?action) ∧ AccessPermission(?permission) ∧ AccessCond(?condition) ∧ ContextInfo(?context) ∧ swrlb:equal(?condition, ?context) → **sqwrl:select(?role, ?resource, ?permission, ?action)**

Table 5. Access Query Result

	?role	?resource	?permission	?action
1	MP00X(MedicalPractitioner)	EMR	granted	read
2	MP00X(MedicalPractitioner)	EMR	granted	write
3	MP00X(MedicalPractitioner)	DMR	granted	read

A *Service Access Request* is defined as a tuple, <*pass*, *service*>, where *pass* is the user's access pass *(identity, password)*, and *service* is the *(action, resource)* pair. When a *service access request* comes, the user is first identified based on his provided *pass*. Then, the policy ontology identifies the relevant access control policies. It also identifies the low-level and high-level context information using the context ontology. Then the defined access query is used to process the user's request to access the services using both the policy constraints and the relevant contexts. For the application example (*Scene #1*), a simplified version of the access query (see Table 4) is used to match the relevant policy constraints against the relevant contexts, in order to determine whether the access is *granted* (if matching result is *true*) or *denied* (otherwise). Table 5 presents query results.

5 Prototype Implementation and Case Study

Framework Prototype. We have developed a prototype implementing CAAC approach in J2SE. We have used the Protégé-OWL API to implement the ontologies. We have used Java and the Jess Rule Engine to implement a context reasoner for executing the SWRL rules. We have implemented a set of $APIs$, which can support the software engineers to develop CAAC applications using this framework. The $ContextManager$ provides functionalities to manipulate the context ontologies (capturing low-level context facts and inferring high-level contexts). We have developed a number of *context providers* and the *context reasoner* as parts of the $ContextManager$. The $PolicyManager$ allows CAAC application developers to add, edit and delete access control policies. The $CAACDecisionEngine$ checks the user's request to access the software services/resources and makes access control decisions using the relevant context information. In addition, the *query manager* (part of the $CAACDecisionEngine$) allows application developers to add, edit and delete the access queries.

Application Prototype. Following the motivation scenario, we have developed a demo CAAC application, called *Patient Medical Record Management (PMRM)*, as an example of how to realise CAAC decisions. The application allows different users to invoke different operations on the requested services to access specific patient records in a context-aware manner.

For Scene #1, when Jane wants to access the requested service $writeEMR()$, a service *AccessRequest* is submitted to the $CAACDecisionEngine$ for evaluation. The defined query (see Table 4) is used to retrieve the access control decision (access *permission* is "granted" or "denied").

The COAC ontology contains the relevant low-level context facts (from the *context providers*). It also captures the relevant high-level implicit contexts using the *context reasoner* based on the basic information in the ontology and

the reasoning rules (e.g., Rule #1(a) and Rule #2(a) in Table 2). The CAPO ontology captures the relevant policy constraints (e.g., the policy in Table 3) applicable to the requested service, i.e., (*action*, *resource*) pair. The relevant contexts and the policy constraints, captured at the time of access request, are provided to the *CAACDecisionEngine* as part of the request processing. The current contexts are then matched against the constraints of the policy as part of making the access control decision. Based on this information, the *CAACDecisionEngine* returns an access control decision for the submitted access request. One of the entries in the access query results (Line #2 of Table 5) satisfies Jane's *AccessRequest*. That is, ***if*** Jane and Bob both are *located in* the "emergency room" of the hospital ***and*** Bob's current *health status* is "critical" ***and*** the *interpersonal relationship* between Jane and Bob is "any physician", ***then*** Jane is authorized to access the service *writeEMR*(), because the available contexts and policy constraints indicate that the access conditions are satisfied.

6 Related Work and Discussion

Several research efforts (e.g., [1],[5],[8],[11]) have adopted and extended the Role-based Access Control (RBAC) approach for access control to software services. Some of these efforts (e.g., [1]) incorporate specific types of contexts such as location and time. Kulkarni et al [8] have proposed a context-aware RBAC (CA-RBAC) model for pervasive applications. They consider user and resource attributes as the context constraints. He et al [5] have considered access control for Web service based on the user role and presented a CAAC policy model considering the user, resource and environment concepts. These approaches consider specific types of contexts which are not general enough in dynamic environments. In contrast, we have proposed a general and extensible ontology-based context model, in which we introduce several additional general concepts for context modelling, including resource owner and relationship between different persons. Toninelli et al [11] have proposed a semantic CAAC approach which provides resource access permission on the basis of context (resource availability, roles of user, location and time). It includes an ontology-based framework with context and policy models. Similar to the above approaches, however, its context model does not consider several concepts which are important for access control in today's dynamic environments, including owner, relationship between user and owner, the derived entity status (e.g., health status) information, etc. In addition to these concepts, our approach also supports resource hierarchy, and consequently user's access to resources at different levels of granularity.

A number of further research efforts (e.g., [2],[3],[6]) have extended the Attribute-based Access Control (ABAC) approach to provide access control to software services in a context-aware manner. Corradi et al [2] have proposed a CAAC model for ubiquitous environments, where permissions are directly associated with contexts: user location, user activities, user device, time, resource availability and resource status. Hulsebosch et al [6] have proposed a context-sensitive access control framework based on the user's location and access history. These approaches also have limitations in considering a limited set of contexts. A recent CAAC framework for the Web of data is grounded on two ontologies which deal

with the core access control policy concepts and the context concepts [3]. Even though it does consider three important dimensions of context: user, device and environment, it does not have the capability of inferring high-level implicit contexts. These attribute-based approaches have major limitations when applied in large-scale domains because of the huge number of attributes involved. In addition, they do not directly treat *roles* as first class entity.

7 Conclusion

In this paper, we have introduced a new ontology-based approach to context-aware access control for software services. It includes an extensible context model specific to access control, and a reasoning model for inferring high-level implicit contexts based on user-defined rules, and an access control policy model incorporating context information from the context model. In order to demonstrate the practical applicability of our approach, we have developed a prototype framework implementing the approach. We have also developed a context-aware access control application in the healthcare domain and have presented a healthcare case study. The case study has shown that our framework is effective in practice.

Acknowledgment. Jun Han is partly supported by the Qatar National Research Fund (QNRF) under Grant No. NPRP 09-069-1-009. The statements made herein are solely the responsibility of the authors.

References

1. Chandran, S.M., Joshi, J.B.D.: *LoT-RBAC*: A location and time-based rbac model. In: WISE. pp. 361–375 (2005)
2. Corradi, A., Montanari, R., Tibaldi, D.: Context-based access control management in ubiquitous environments. In: NCA. pp. 253–260 (2004)
3. Costabello, L., Villata, S., Gandon, F.: Context-aware access control for rdf graph stores. In: ECAI. pp. 282–287 (2012)
4. Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing 5(1), 4–7 (2001)
5. He, Z., Wu, L., Li, H., Lai, H., Hong, Z.: Semantics-based access control approach for web service. JCP 6(6), 1152–1161 (2011)
6. Hulsebosch, R.J., Salden, A.H., Bargh, M.S., Ebben, P.W.G., Reitsma, J.: Context sensitive access control. In: SACMAT. pp. 111–119 (2005)
7. Kayes, A.S.M., Han, J., Colman, A.: *ICAF*: A context-aware framework for access control. In: ACISP. pp. 442–449 (2012)
8. Kulkarni, D., Tripathi, A.: Context-aware role-based access control in pervasive computing systems. In: SACMAT. pp. 113–122 (2008)
9. Riboni, D., Bettini, C.: Owl 2 modeling and reasoning with complex human activities. Pervasive and Mobile Computing 7(3), 379–395 (2011)
10. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)
11. Toninelli, A., Montanari, R., Kagal, L., Lassila, O.: A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In: ISWC. pp. 473–486 (2006)
12. Weiser, M.: Some computer science issues in ubiquitous computing. Communications of the ACM 36(7), 75–84 (1993)