A Semantic Policy Framework for Context-Aware Access Control Applications

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—Due to the rapid advancement of communication technologies, the ability to support access control to resources in open and dynamic environments is crucial. On the one hand, users demand access to resources and services in anywhere, anytime fashion. On the other hand, additional challenges arise when ensuring privacy and security requirements of the stakeholders in dynamically changing environments. Conventional Role-based Access Control (RBAC) systems evaluate access permissions depending on the identity/role of the users who are requesting access to resources. However, this approach does not incorporate dynamically changing context information which could have an impact on access decisions in open and dynamic environments. In such environments, an access control model with both dynamic associations of user-role and role-permission capabilities is needed. In order to achieve the above goal, this paper proposes a novel policy framework for context-aware access control (CAAC) applications that extends the RBAC model with dynamic attributes defined in an ontology. We introduce a formal language for specifying our framework including its basic elements, syntax and semantics. Our policy framework uses the relevant context information in order to enable user-role assignment, while using purpose-oriented situation information to enable role-permission assignment. We have developed a prototype to realize the framework and demonstrated the framework through a healthcare case study.

Keywords—Privacy and Security; Context; Situation; User-Role Assignment; Role-Permission Assignment; Policy Framework

I. INTRODUCTION

Access control is one of the fundamental security mechanisms needed to protect computer resource, verifying whether a user is allowed to carry out a specific action on that resource. A recent study [1] shows that Role-based Access Control (RBAC) [2] has become the most widely used access control model; it typically evaluates access permission through roles assigned to users and each role assigns a collection of permissions to users who are requesting access to the resources. That is, RBAC simplifies the management of access control policies by creating user-role and role-permission mappings. In small-scale RBAC systems, the existence of an administrator to manage all the user-role and role-permission assignments in the system is realistic. However, large-scale RBAC systems may have hundreds or even thousands of roles/permissions and hundreds of thousands of users. In such cases, it is impractical for an administrator to manage the role assignments for all the users. Therefore, an access control model with dynamic associations of user-role and role-permission capabilities is needed.

In general, context-awareness can enable more effective access control decision making in dynamically changing environments. Even though some access control models have already been proposed in context-aware computing environments, a comprehensive policy framework, flexible and dynamic enough to deal with dynamic context-aware user-role and role-permission assignments, is still missing. On the one hand, users demand access to resources and services in 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 privacy and security requirements of the stakeholders. For instance, consider the roles of patient and nurse in a hospital context. A nurse Mary may be allowed to access the medical records of a patient Bob for daily operational purposes, if she has been assigned to look after Bob but only when she is located in the general ward. Therefore, an access controller needs to evaluate such dynamic attributes when enabling user-role and role-permission assignments.

The overall goal of this paper is to introduce a Context-Aware Access Control (CAAC) policy framework, an extension of the basic RBAC framework. The novel feature of this framework is an expressive formal language for specifying the elements of our framework, and its syntax and semantics, specifically addressing the following aspects:

- **Context-aware user-role assignment**: Our model uses the relevant context information in order to enable context-aware user-role assignment. We take context to mean any information that can be used to characterize the state of a relevant entity or the state a relevant relationship between different entities.

- **Context-aware role-permission assignment**: Our model uses the purpose-oriented situation information in order to enable context-aware role-permission assignment. A purpose-oriented situation is defined as a specific subset of the complete state of the universe of the relevant entities that are relevant to a certain goal or purpose of a resource access request.

- **Context/situation expression and context/situation specification language (CSSL)**: A context expression is used to express the context constraints in order to describe the user-role assignment policies. A situation expression is used to express the situation constraints to describe the
role-permission assignment policies. The CSSL language is used to specify the context or situation constraints.

Based on the above aspects, we introduce a semantic policy framework for defining and enforcing access control policies that take into account the relevant dynamic attributes. Our policy framework represents the basic elements using the ontology language OWL, extended with SWRL for reasoning about dynamic attributes that are not obtainable directly but can be inferred from the other available attributes. To demonstrate the effectiveness of our framework we have developed a prototype and carried out a healthcare case study.

The rest of this paper is organized as follows. Section II presents an application scenario. In Section III, we present a formal model for specifying the elements of our framework. Section IV provides a semantic policy framework by expressing the ontology and represents rules for a proper access decision. Section V explains the prototype implementation of the framework along with a healthcare case study. We compare our work with the related work in Section VI. Finally, Section VII concludes the paper and outlines future work.

II. RESEARCH MOTIVATION

Before presenting a formal definition of our policy framework, let us consider a scenario from the healthcare domain, requiring 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 at 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.

Two of the relevant access control policies related to the above scenes are shown in Table I. The policies are based on a set of constraints on the user role and resource, and the relevant context information as well.

<table>
<thead>
<tr>
<th>No</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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 for emergency treatment purpose. However, in an emergency situation (like the Scene #1), all medical practitioners should be able to access the emergency medical records from the emergency room of the hospital (by playing the emergency doctor role).</td>
</tr>
<tr>
<td>2</td>
<td>A registered nurse (who is assigned for a patient) within a hospital is granted the right to read the patient’s daily medical records from the location where the patient is located, during her ward shift time for daily operational purpose. Moreover, a registered nurse can access the patient’s past medical history, if a medical practitioner is present in the same location.</td>
</tr>
</tbody>
</table>

TABLE I
INFORMAL ACCESS CONTROL POLICIES

General Requirements. The above example scenario illustrates many of the key ideas of the research presented in this paper. As different types of dynamic attributes (context information) are involved in the scenario, some important issues arise. These issues and their related requirements are:

- The dynamic attributes which are relevant to access control 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, the purpose of resource access, etc. In general, there is a need to identify the relevant dynamic attributes which are integrated into the access control process.

- Normally, only a patient’s treating physician is able to access the patient’s emergency medical records. In the above emergency scenario, Jane, while not being the treating doctor, can access Bob’s emergency medical records from the emergency room of the hospital by playing emergency doctor role. That is, Jane can play emergency doctor role when he is present in the emergency room. He also can play medical practitioner role; these are, however, not at the same time (for security concern). Thus, rather than having static user-role assignment it is required to dynamically activate and revoke role according to the context.

- In general, a registered nurse is granted the right to read a patient’s daily medical records within a hospital. But, she should only be able to read the medical records from the location where the patient is located, during her ward shift time. In the above scenario, Mary can access Bob’s daily medical records during her ward shift time because she is present with Bob in the general ward. Furthermore, when the situation changes (e.g., Bob’s health condition critical again), decisions on further access requests by Mary to Bob’s daily medical records, may change accordingly (e.g., denied). Therefore, it is necessary to consider the relevant dynamic attributes for role-permission assignment.

III. CORE POLICY FRAMEWORK

We introduce a novel policy framework which enables dynamic privileges assignment at two levels. On the one hand, the roles are dynamically assigned to the users based on the relevant context constraints. Towards this end, we introduce a new concept context-dependent role activation/revocation. On the other hand, when a role is activated, then it can be used to apply access permissions assigned to that role based on the relevant situation constraints. To this goal, we introduce another new concept named purpose-oriented situation.

Definition 1 (Core Policy Model). Our policy model (M) can be formally described as a tuple (see Equation 1):

\[ M = (M_S, M_R) \]

\[ M_S = (U, R, CE, RS, OP, P, SE, Pol) \]

\[ M_R = (RH, CAUA, RSH, OPA, CAPA) \]

Figure 1 shows our policy model and the relationships between its thirteen elements. In this paper, we use first order logic to make formal descriptions of these elements.

Based on the formalization of the traditional role-based access control (RBAC) model [2], we present a formal definition of our policy model. First of all, we define following eight elements (sets) \( (M_S, \text{see Equation 1}) \) of our policy model:
**Users (U):** A set of users $U := \{u_1, ..., u_m\}$. In our model, a user is a human-being (who is a resource requester) interacting with a computing system whose access request is being controlled.

**Roles (R):** A set of roles $R := \{r_1, ..., r_n\}$. A role reflects user’s job function. It is associated with a set of permissions.

**Context Expressions (CE):** A set of context expressions $CE := \{ce_1, ..., ce_o\}$ specified by using the context/situation specification language (CSSL) (see Subsection III.C for CSSL). A context expression is used to express the context constraints in order to describe the user-role assignment policies.

**Resources (RS):** A set of component parts of resources $RS := \{rs_1, ..., rs_p\}$. Resources are the objects protected by access control. Based on our scenario, a resource represents the data/information container (the different component parts of a patient’s medical records).

**Operations (OP):** A set of operations on the resources $OP := \{op_1, ..., op_q\}$. An operation is an action that can be executed on the resources, for instance, read and write.

**Permissions (P):** A set of permissions $P := \{p_1, ..., p_r\} := \{(rs, op)|rs \in RS, op \in OP\}$. Permission is an approval to perform certain operations on resources, by the users who are originating access request.

**Situation Expressions (SE):** A set of situation expressions $SE := \{se_1, ..., se_s\}$ specified by using the CSSL language. A situation expression is used to express the situation constraints in order to describe the role-permission assignment policies.

**Policies (Pol):** A set of policies $Pol := \{pol_{CAUA} \cup pol_{CAPA}\}$. Our model has two sets of policies, the context-aware user-role assignment policies and context-aware role-permission assignment policies.

Originating from the above sets, we derive five other elements (using the relationships between sets) $(M_R, \text{Definition 1})$ of our model which are defined formally as follows:

- **Role Hierarchy (RH):** $RH \subseteq R \times R$ is a partial order on $R$ to serve as the role hierarchy, which supports the concept of role inheritance. The role is considered in a hierarchical manner in that if a permission assigned to a junior role, then it also be assigned to all the senior roles of that role; or if a user assigned to a junior role, then he also be assigned to all the senior roles of that role.

- **Context-Aware User-Role Assignment (CAUA):** $CAUA \subseteq U \times R \times CE$ is a context-aware user-role assignment relation, which is a many-to-many mapping between a set of users and roles, associated with certain (relevant) context constraints.

- **Resource Hierarchy (RSH):** $RSH \subseteq RS \times RS$ is a partial order on $RS$ to serve as the resource hierarchy, which supports a user to access the different granularity levels of resources. The resource is considered in a hierarchical manner in that if a user has the right to access a resource with the highest granularity level, then he also has the right to access the lower granularity levels of that resource.

- **Operation Assignment (OPA):** $OPA \subseteq RS \times OP$ is a many-to-many operation-to-resource mapping. Each operation could be associated with many resources, and for each resource could be granted to many operations. A set of operation assignment relations, $OPA := \{(rs, op)|rs \in RS, op \in OP\}$. Each operation could be associated with many resources, and

- **Context-Aware Role-Permission Assignment (CAPA):** $CAPA \subseteq R \times P \times SE$ is a context-aware role-permission assignment relation, which is a many-to-many mapping between a set of roles and permissions, associated with certain (relevant) situation constraints.

In the following, we give some further formal description of context, situation, CSSL, context-dependent role activation/revocation, policy specification, and policy enforcement.
A. Context

In the context-aware literature, many researchers have attempted to define the concept of context. According to Dey, the general context entities are person, place or object and a context represents the information that can be used to characterize the situation of an entity [3]. However, this definition does not adequately and explicitly identify the access control-specific context entities and context information. Following Dey’s general context definition, we propose an access control-specific context definition [4].

Definition 2 (Context Information Type - CT). Context information used in an access control decision is defined as any relevant information about the state of a relevant access control-specific context entity (E) or the relevant relationship between different relevant entities at a particular time.

For example, the identity/role of user and the interpersonal/colocated relationship between user and owner are the context information.

Focusing on the context entities and information relevant to making access control decisions, we classify the access control-specific context entities (E) into two groups: core (EC) and environmental (EE).

\[ E := \{E_C \cup E_E\} \]  

User, Resource, and resource Owner are the core access control entities as they are core concepts of access control. In order to offer the advantages of the RBAC role, which regulates access to services based on user roles rather than individual users, we also consider Role as a core entity. The Relationship between different persons is another class of core entity. For example, in our application scenario (Scene #1), the interpersonal Relationship between Jane and Bob is "nonTreatingPhysician".

\[ E_C := \{User \cup Role \cup Resource \cup Owner \cup Relationship\} \]  

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. In the example scenario (Scene #2) where a registered nurse Mary can access a patient Bob’s past medical history if Jane (by playing medical practitioner role) is present in the same location with Mary, for instance, Mary is a User, Bob is an Owner, and Jane is an envPerson.

\[ E_E := \{envPerson \cup Place \cup Device\} \]  

In our model, we consider two different types of context information: namely low-level context and high-level context, i.e., context information type (CT) is the set of all low-level context (LCT) and high-level context (HCT).

\[ CT := \{LCT \cup HCT\} \]  

Definition 3 (Low-level Context - LCT). A low-level context represents a context attribute related to an entity (E). It depends on the raw context fact that is used to characterize the state of an entity.

In our model, we include the low-level context information such as identity, granularityLevel, locationAddress, requestTime, heartRate, bodyTemperature, roomTemperature, humidity, weatherCondition, etc.

Definition 4 (High-level Context - HCT). A high-level context (i.e., derived or inferred context) is a context attribute related to an entity or a subset of entities (e.g., relationship between entities). It depends on the values for low-level contexts or other high-level contexts that is used to characterize the state of one or more entities.

In our model, we include the high-level context information such as interRelationship, colocatedRelationship, socialProfile, healthStatus, etc.

B. Situation

In the context-aware literature, existing situation definitions typically describe the state of the user [5]. However, these definitions are limited when considering access control in open and dynamic environments, where a user wants to access specific resources from a particular environment (e.g., a patient is in a critical health condition) for a certain purpose. In addition to the state of the user, the states of the other context entities are also important considerations. Moreover, there exists a goal or purpose in every situation concerning the resource access. For example, in the medical domain the American Health Information Management Association (AHIMA) prepared 18 health care scenarios ranging across 11 purposes (treatment, payment, research, etc.) for health information exchange [6]. Based on the above discussion, we propose the purpose-oriented situation definition [4].

Definition 5 (Situation). A situation is defined as a specific subset of the complete state of the universe of access control-specific context entities that are relevant to a certain goal or purpose of a resource access request.

Here, the universe is formed or defined by the context entities. A situation is a set of values whose types are defined by the domain-specific context predicates at a particular time. These values are determined by what the system needs to know given its current state in order to make the context-aware access control decision.

A security policy normally states that the particular resource(s) can be accessed only for the specific purpose(s) under specific environmental context conditions; and it describes the reason for which organizational resources are used [7].

Definition 6 (Purpose). A set of purposes PR := \{pr1, ..., prt\}. In our model, a purpose is the user’s intention to access the resources. It is a special high-level context.

Purpose is a domain dependent concept, and can be derived based on an access request (i.e., based on the relevant context values that are present in which a user is requesting access to the resources). If there is no purpose can possible to derive, then our model by default considers purpose pr := ("no").
C. Context/Situation Specification Language - CSSL

Our proposed model includes a simple language for expressing context and situation constraints named CSSL. Based on this language, it is possible to specify the more complex constraints in access control policies.

Definition 7 (Atomic Constraint). Let E be the set of context entities, and CT be the set of context types, then we define an atomic constraint \( ac \) as the 4-tuple
\[
ac := (e, ct, rel.op, v)
\]
where \( e \) is a context entity; \( ct \) is a context information type; \( rel.op \) is a relational operator (the set of \( rel.op \) can be extended to accommodate user-defined operators (e.g., entering) as well; and \( v \) indicates the value assigned to the context entity \( e \) for the context type \( ct \). For the sake of simplicity, an atomic constraint is represented in the form of
\[
ac := (< e.ct > < rel.op > < v >)
\]
where \( e.ct \) represents a context attribute.

Example 1. A patient or owner’s heart rate is less than 65 (or is abnormal), which is represented as,
\[
ac := (Owner.heartRate < 65) \text{ or } "\text{Abnormal}"
\]
An atomic constraint is a simple constraint of the CSSL language. It is possible to construct more complex expressions by performing Boolean operations like conjunction (\( \land \)), disjunction (\( \lor \)), and negation (\( \neg \)) over atomic constraints.

Definition 8 (Context Expression language). By definitions 2 and 7, we can define a context expression \( ce \) as an atomic constraint or a boolean combination of the relevant atomic constraints. The following is a context expression language.
\[
ce := ce | ce \land ce | ce \lor ce | \neg ce
\]
where \( ce := ac \).

Definition 9 (Situation Expression Language). By definitions 5, 6 and 7, we can define situation expression \( se \) as a boolean combination of a purpose \( pr \) and the relevant atomic constraints. The following is a situation expression language.
\[
se := se \land se | se \lor se | \neg se
\]
where \( se := pr \land ac \).

Example 2. Based on the Policy #2 in our example scenario,
\[
ce := ((User.locationAddress = "GeneralWard") \land
(User.requestTime = "DutyTime"))
\]
D. Context-Dependent Role Activation/Revocation

Let \( CE \) be the set of context expressions, and \( R \) be the set of roles, then we define a function named context-dependent role activation/revocation
\[
CRAR : CE \times R \rightarrow BOOL
\]
such that, given a context expression \( ce \) and a role \( r \), \( CRAR(ce, r) := TRUE \), if the relevant context constraints are satisfied. Otherwise, \( CRAR(ce, r) := FALSE \).

If a role \( r \) is activated, then all the ancestors of \( r \) in the hierarchy are activated. Activated roles are the basis for determining whether to allow or deny access permissions. If an activated role is invalid or needed to be deactivated (called role revocation) when related context state is changed. If a role is deactivated, then the descendant roles in the hierarchy are also deactivated. Figure 2 shows an example of a hierarchy of role for our example application. Based on the Policy #2 in our application, the RegisteredNurse role is activated when the nurse present in the general ward and at duty time. Based on the role hierarchy (see Figure 2), we have partial ordering.

NurseManager \( \preceq \) RegisteredNurse
Nurse \( \preceq \) RegisteredNurse

If a role RegisteredNurse is activated, then Nurse and NurseManager are activated as well. If RegisteredNurse role is deactivated (when the nurse leaves from the general ward or after ward shift time), then Nurse and NurseManager are deactivated as well.

E. Resource Hierarchy

To provide fine-grained access control and grant the right access to the appropriate parts of a resource by the appropriate users, the resource needs to be considered in a hierarchical manner. In such a way, access to the resource and its different components can be managed. As such, there is the

![Role Hierarchy Diagram](image-url)
healthcare Resource (the different component parts of the patients’ medical health records) hierarchy in the domain ontology (Figure 2 shows a part of the Resource hierarchy). The complete medical records would be obtained by joining all the different component parts, which is identified by a class Emergency Medical Records (EMR). The categorization of the EMR class is important, because in emergency situation (a patient is in a critical health condition, see Scene #1 in the motivating scenario), for example, all general practitioners can access all the component parts of a patient’s medical records. Emergency Medical Records has the following subcomponents: Daily Medical Records (DMR), which includes Physiological Records (PR), Physician Prescriptions (PP), Daily Observations Reports (DOR), etc.; a patient’s Past Medical History (PMH); and so on. 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.

F. Policy Specification

1) Context-Aware User-Role Assignment (CAUA) Policy: Our model extends the concept of common user-role assignment in RBAC, by introducing the context-dependent role activation, called as context-aware user-role assignment.

Traditional RBAC model defines user-role assignments simply as a mapping of users to roles, URA ⊆ U × R. Here, we extend this notion by integrating context constraints.

Definition 10 (CAUA). Let U be the set of users, R be the set of roles and CE be the set of context expressions, then:

\[ CAUA \subseteq U \times R \times CE \]  

(15)
a many-to-many mapping user-role assignment relation associated with certain context constraints (expression).

Context-aware user-role assignments can be expressed in tabular form (see Table II). The second one describes when Mary is present in the general ward, at ward shift time (duty time), then she can be assigned to the registered nurse role.

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
<th>Context Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>EmergencyDoctor</td>
<td>(User.locationAddress = &quot;EmergencyRoom&quot;)</td>
</tr>
<tr>
<td>Mary</td>
<td>RegisteredNurse</td>
<td>(User.locationAddress = &quot;GeneralWard&quot;) ∧ (User.requestTime = &quot;DutyTime&quot;)</td>
</tr>
</tbody>
</table>

Definition 11 (CAUA Policy). Let U be the set of users, R be the set of roles and CE be the set of context expressions. A CAUA policy \( \text{pol}_{CAUA} \) is defined as follows:

\[ (\forall u, r)(\text{state}^U(u) \land \text{state}^R(r) \land \text{state}^{CE}(ce) \land \text{enable_token} \rightarrow (u, r, ce) \in CAUA) \]  

(16)

where \( \text{state}^U(u) \) denotes a user state, referring to a user \( u \) (\( u \in U \)), e.g., User.identity = “Mary00X”; \( \text{state}^R(r) \) denotes a role state, referring to a role \( r \) (\( r \in R \)), e.g., Role.name = “RegisteredNurse”; \( \text{state}^{CE}(ce) \) denotes a context state, referring to a context constraint expression \( ce \) (\( ce \in CE \)) defined using the CSSL; and enable_token indicates whether the role \( r \) is enabled/activated (i.e., enable_token = TRUE), or deactivated (i.e., enable_token = FALSE).

In the above policy, all of the states are expressions in first order logic. The above policy rule states that for any user \( u \), satisfying a given user state \( \text{state}^U(u) \), for any role \( r \), satisfying role state \( \text{state}^R(r) \), any context constraint expression \( ce \), satisfying context state \( \text{state}^{CE}(ce) \), and enable_token is TRUE, which means that according to this policy rule, \( u \) can be assigned to \( r \) in context state \( ce \); if enable_token is FALSE, which means that according to this policy rule, \( u \) cannot be assigned to \( r \) in context state \( ce \). The justification is that let us suppose a role is activated for any specific context expression (e.g., a registered nurse in general ward), it might be deactivated when the nurse leaves from the general ward. For the sake of readability, a CAUA policy is also defined as in the format below:

\[ \text{pol}_{CAUA} = (u, r, ce, \text{enable_token}) \]  

(17)

2) Context-Aware Role-Permission Assignment (CAPA) Policy: Similar to context-aware user-role assignment, our model for this part extends the concept of role-permission assignment in RBAC with situation constraints.

Definition 12 (CAPA). Let \( R \) be the set of roles, \( P \) be the set of permissions, and \( SE \) be the set of situation expressions, then:

\[ \text{CAPA} \subseteq R \times P \times SE \]  

(18)
a many-to-many mapping role-permission assignment relation associated with certain situation constraints (expression).

Definition 13 (CAPA Policy). Let \( R \) be the set of roles, \( P \) be the set of permissions, and \( SE \) be the set of situation expressions. A CAPA policy is defined as follows:

\[ (\forall r, p)(\text{state}^R(r) \land \text{state}^P(p) \land \text{state}^{SE}(se) \land \text{permission_token} \rightarrow (r, p, se) \in \text{CAPA}) \]  

(19)

where \( \text{state}^R(r) \) denotes a role state, referring to a role \( r \) (\( r \in R \)), \( \text{state}^P(p) \) denotes the format of a permission \( p \) (\( p \in P \)); \( \text{state}^{SE}(se) \) denotes a situation state, referring to a situation constraint expression \( se \) (\( se \in SE \)), defined using CSSL; and permission_token indicates whether the access authorization is permitted (permission_token = ALLOW), or prohibited (permission_token = DENY).

In the above policy, all of the states are expressions in first order logic. The above policy rule states that for any role \( r \), satisfying role state \( \text{state}^R(r) \), for any permission \( p \), satisfying the format of a permission \( \text{state}^P(p) \), for any situation constraint expression \( se \), satisfying situation state \( \text{state}^{SE}(se) \), and permission_token is ALLOW, which means that according to this policy rule, \( r \) is assigned with permission \( p \) in situation state \( se \) where the access authorization is permitted; if permission_token is DENY, which means that according to this policy rule, \( r \) is assigned with permission \( p \) in situation state \( se \) where the access authorization is prohibited. For the
sake of readability, a \( \text{CAPA} \) policy is also defined as in the format below:

\[
\text{pol}_{\text{CAPA}} = (r, p, se, \text{permission\_token}) \tag{20}
\]

G. Policy Enforcement

**Definition 14 (Access Request).** We define an access request \( ar \) as a tuple in the form of

\[
ar := (<\usr>, <\perm>, <\con>) \tag{21}
\]

where \( \usr \) is a user who issues this resource access; \( \perm = (rs, o) \) is a permission that this user wants to acquire (where \( rs \) is the requested resource, and \( o \) is an operation that can be executed on that resource); and \( \con \) is a set of dynamic attributes at request time.

When a user wants to access resources, then user’s access request \( ar \) is granted (by evaluating the access control policies), if there exists a context-aware user-role assignment policy \( \text{pol}_{\text{CAUA}} = (u, r, ce, \text{enable\_token}) \) such that \( \text{enable\_token} = \text{TRUE} \) when \( \usr = u \), and context expression \( ce \) evaluates to true under \( \con \) (i.e., when all atomic constraints \( ac \) in context expression \( ce \) are replaced with dynamic attributes from access context \( \con \), then the resulted Boolean expression is \( \text{True} \)); and also if there exists a context-aware role-permission assignment policy \( \text{pol}_{\text{CAPA}} = (r, p, se, \text{permission\_token}) \) such that \( \text{permission\_token} = \text{ALLOW} \) when \( \perm = p \), and situation expression \( se \) evaluates to true under \( \con \) (i.e., when all \( ac \) in situation expression \( se \) are replaced with dynamic attributes in access context \( \con \), then the resultant Boolean expression is \( \text{True} \)).

Otherwise, the user’s request is denied.

In this paper, we describe the use of relevant contexts and situations for dynamic assignments of user-role and role-permission, and rather focus on the detailed context/situation management framework for managing and updating dynamic attributes at run time. A simplified version of the context ontology is depicted in Figure 4 (Section IV), in order to capture and represent relevant contexts/situations at run time.

IV. ONTOLOGY-BASED FRAMEWORK IMPLEMENTATION

In this section we propose an ontology-based framework specific to context-aware access control, in which we present the two main parts of our framework: policy specification (integrating relevant attributes into the policies) and policy enforcement (determining access control decisions).

**Domain-Specific Design Considerations.** Our policy framework is a hierarchical model that is extended by a set of business rules that are application specific (healthcare domain). An ontology representing this framework elements may fulfill the following requirements: representing all the framework elements (general and domain-specific) and the relationships between these elements; and incorporating the business rules as OWL language constructs where possible, otherwise, express the business rules using SWRL rules to obtain high-level knowledge and extend the ontology with these newly created rules. To simplify the management of access control policies, various languages have been proposed in the literature. Experience from existing research (e.g., [8],[9]) shows that ontologies are very suitable for modeling context elements for pervasive computing applications. The expressivity of OWL [10] can be extended by adding SWRL rules [11] to an ontology.

Our goal in this research is to provide a way in which the two sets of policies can be specified by incorporating dynamic attributes (basic and implicit context information).

![Fig. 3. A Simplified Version of the Policy Ontology](image-url)

**A. Ontology Structuring using OWL and Extending the Ontology using SWRL.**

The simplified version of our policy ontology is depicted in Figure 3, where different elements are Policy, User, Role, Resource, Assignment, Operation, and Condition (which has Context and Situation classes). The following example in OWL shows the class Policy has an object property hasUser, which is used to link the classes Policy and User.

```owl
<owl:Class rdf:ID="Policy"/>
<owl:Class rdf:ID="User"/>
<owl:ObjectProperty rdf:ID="hasUser"/>
<rdfs:domain rdf:resource="#Policy"/>
<rdfs:range rdf:resource="#User"/>
</owl:ObjectProperty>
```

Similarly the following example shows that the class Resource has an attribute resourceID (string type property).

```owl
<owl:DatatypeProperty rdf:ID="resourceID"/>
<rdfs:domain rdf:resource="#Resource"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

The following example specifies the cardinality of the class Assignment on the property permission_token (every policy has only one permission_token value (“Allow” or “Deny”)).

```owl
<owl:Class rdf:ID="Assignment"/>
<owl:Restriction/>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegative Integer">1</owl:cardinality>
</owl:Restriction>
</owl:Class>
```
Let us consider an access control policy for the registered nurses (Policy #2 in our application scenario). The access decision is based on the following policy constraints: who the user is (user’s role), what resource being requested (resource’s identity), and when the user sends the request (the interpersonal relationship between user and owner, the health status of the owner/patient, the locations of the user and owner). The template of a user-role assignment policy in a readable form is shown in Table III and the specific policy states that a user can play the registered nurse role during ward shift time and when she is located in the ward.

**Table III**

**AN EXAMPLE USER-ROLE ASSIGNMENT POLICY**

<table>
<thead>
<tr>
<th>Policy (hasUser, hasRole, hasCondition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>User.userID = “Mary00X”; &amp; // User’s identity (from RDBMS)</td>
</tr>
<tr>
<td>User.Role.roleID = “RN00X”; &amp; // Registered Nurse’s role identity</td>
</tr>
<tr>
<td>Condition.locationAddress(User) = “GW00X”; &amp; // General Ward</td>
</tr>
<tr>
<td>Condition.requestTime(User) = “DutyTime”</td>
</tr>
<tr>
<td>→ Assignment.enable_token = “true”.</td>
</tr>
</tbody>
</table>

Table IV shows a role-permission assignment policy in a simplified form and the specific policy states that all registered nurse can access a patient’s daily medical records for daily operational purpose when the patient’s health status is normal and she is present with the patient.

**Table IV**

**AN EXAMPLE ROLE-PERMISSION ASSIGNMENT POLICY**

<table>
<thead>
<tr>
<th>Policy (hasRole, hasResource, hasCondition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>User.Role.roleID = “RN00X”; &amp; // Registered Nurse’s role identity</td>
</tr>
<tr>
<td>Resource.resourceID = 2; &amp; // Daily Medical Records (DMR)</td>
</tr>
<tr>
<td>Operation.operationType = “Read”; &amp;</td>
</tr>
<tr>
<td>Condition.locationAddress = “GW00X”; &amp; // General Ward</td>
</tr>
<tr>
<td>Condition.requestTime = “DutyTime”</td>
</tr>
<tr>
<td>→ Assignment.permission_token = “Allow”;</td>
</tr>
</tbody>
</table>

During the policy enforcement 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 [12], which is based on OWL and SWRL, is adopted to process user’s access requests (see next Section).

![A Simplified Version of the Context Ontology](image)

In our policy framework, we also define some classes to capture the dynamic attributes (relevant contexts and situations) at run time. The class Context is used to capture access control-specific relevant context information at runtime. A part of the context ontology is shown in Figure 4. The ontology defines the object property hasRelationship that is used to link the Person (User and Owner) and the Relationship. Based on the motivating scenario, we classify relationship information into interRelationship (xsd:string type property) and colocatedRelationship (xsd:string type property). We also define a location context type class SpatialInfo, which has a data type property (xsd:string type) named locationAddress.

The ontology is extended with SWRL reasoning rules for automated inference of implicit contexts from limited contexts. For example, the rule in Table V states that if the patient and nurse are located in the same place then they both are “colocated”. After inferring this types of relevant implicit context information, the ontology finally captures these again.

**Table V**

**USER-DEFINED REASONING RULES**

```
User(u) ∧ Owner(o) ∧ Place(p) ∧ hasPlace(u, p) ∧ hasPlace(o, p) ∧ SpatialInfo(?stu) ∧ has(?u, ?stu) ∧ SpatialInfo(?sto) ∧ has(?o, ?sto) ∧ locationAddress(?sto, ?lao) ∧ locationAddress(?stu, ?lau) ∧ swrlb:equal(?lau, ?lao) ∧ Relationship(?rel) ∧ hasRelationship(?u, ?rel) ∧ Relationship(?rel) ∧ hasRelationship(?o, ?rel) ∧ Relationship(?rel) ∧ hasRelationship(?o, ?rel) ∧ RelationshipInfo(?col) ∧ has(?rel, ?col) → colocatedRelationship(?col, “colocated”)```

Similarly, in our policy framework we define one another class named Situation, to capture relevant purpose-oriented situations at runtime (Situation Model).

We have described the development of the Role and Resource hierarchies in Figure 2, by following the healthcare domain concepts. We have also defined the Place ontology structure (Hospital, EmergencyRoom, GeneralWard, etc.).

**V. FRAMEWORK PROTOTYPE AND APPLICATION**

We have developed a prototype implementing out framework in J2SE. We have used the Protégé-OWL API [13] to implement the ontologies. We have used Java and the Jess Rule Engine [14] to implement a context reasoner for executing the SWRL rules. Due to space limitation, the architecture and full description of the prototype can not be included. We have implemented a set of APIs, which can support the software engineers to develop CAAC applications using this framework.

Currently a simple Java class CAACDecisionEngine is used to check the user’s request to access the resources and makes access control decisions. We have implemented PolicyEnforcementPoint as part of the CAACDecisionEngine. Once the CAACDecisionEngine receives the request for resource access, it queries the PolicyManager class for the relevant policies. The PolicyDecisionPoint, PolicyAdministrationPoint and ContextManager are implemented as parts of the class PolicyManager that are used to allow the engineers to add, edit and delete access control policies; and also used to provide functionalities to manipulate the context and situation ontologies (capturing low-level contexts, inferring high-level contexts, capturing purposes and capturing situations).
have developed a number of context providers and the context reasoner as parts of the ContextManager. We have also the knowledge bases which are stored in the form of the OWL ontologies, SWRL rules and SQWRL queries.

A. Application Development and Case Study

In this section, a context-aware application (based on the healthcare application scenario presented in Section II) is used to illustrate the operation of context-aware access control. The environment of our application is the patients’ medical records management (PMRM) system. The main goal that we aim with PMRM is to retrieve (read) and modify (write) different medical records of patients based on the dynamic attributes.

In evaluating our framework we have presented a case study. Consider our example scenario (for Scene #2), where a registered nurse Mary wants to access Bob’s daily medical records (DMR), an access request (see Definition 14) is submitted to the CAACDecisionEngine for evaluation.

The defined query (see an example query in Table VI) is used to retrieve the access control decision (“Allow” or “Deny”). The query says that an access request must satisfy the user-role and role-permission assignment policies (here “Context_1” token holds the relevant context constraints and “Situation_1” token holds the relevant situation constraints (see Figure 5)). Our ontology first captures the relevant policies (e.g., the access policies in Tables III and IV). It also captures the relevant low-level context facts (from the context providers) and the relevant high-level implicit contexts using the context reasoner based on the reasoning rules. The relevant contexts and the policy constraints, captured at the time of access request, are provided to the CAACDecisionEngine as part of the access request processing. The current contexts are then matched against the constraints of the policy using query, as part of making the access control decision.

![Fig. 5. Query Result (shown only one entry)](image)

Based on this information, the CAACDecisionEngine returns the access control decision, i.e., Mary’s access request is granted (see query results in Figure 5), because it satisfies the policies which are stored in the policy base. We can observe that if Mary is located in the general ward, she is authorized to access the DMR of the patient Bob, who is hosted in that ward in his normal health condition; conversely, Mary is not allowed to access such records. In general, at each time of an access request or when context changes, the CAACDecisionEngine sends automated request to the PolicyManager for the relevant policies.

VI. RELATED WORK

Despite considerable interest and research in context-aware environments (e.g., [8]), in this section, we briefly highlight several existing context-dependent access control approaches that employ dynamic attributes in the access control policies.

Several research efforts (e.g., [15],[16],[17],[18],[19]) have adopted and extended the basic Role-based Access Control (RBAC) solution [2], where authorizations to access resources are based on the user assigned role and specific types of dynamic contexts. Some of these efforts (e.g., [15],[16]) incorporate only location and time in the access control policies as policy constraints. The GEO-RBAC [15] model proposes the spatial extent (i.e., geographical location) of role in which the user is to be located for being enabled to play such a role. This model provides specific location-based user-role assignment that is applicable in location-aware environments. Chandran et al [16] proposes a location and time-based RBAC model, which uses temporal and location constraints for enabling and disabling of roles. Kulkarni et al [17] have proposed a context-aware RBAC (CA-RBAC) model for pervasive applications. They use context information in role admission policies, revocation of role memberships, permission activations, etc. They consider user and resource attributes as the context constraints. Users can activate personalized permissions in addition to their roles, thus having a dynamic role-permission assignment. Even though they consider dynamic role-permission assignment, they do not consider purposes and situations. In addition, they also have the limitation of manual user-role assignment.

He et al [18] have considered access control for Web service based on the user role and presented a policy model. This model is still limited in considering specific types of contexts as policy constraints. Similar to the above approaches, this model has limitation in considering dynamic user-role and role-permission assignments. Compared with their work, we use the relevant contexts for user-role assignment and the relevant purpose-oriented situations for role-permission assignment. In addition, we present a formal language for our policy framework. Huang et al [19] have proposed a new approach to integrating Attribute-based Access Control (ABAC) [20] with Role-based Access Control (RBAC) [2]. They consider attribute-based policies for user-role and role-permission assignments, and provide a formal language for their model. Our framework supports fine-grained access control requirements with reasoning capability, as the roles are dynamically assigned to the users based on the relevant context constraints, and the permissions are dynamically assigned to the roles based on the relevant purpose-oriented situation constraints. While the research efforts [18] and [19] provide useful insight to present user-role and role-permission assignment concepts, and these are related to our proposal, these two important approaches and other existing role-based
approaches do not consider several access control-specific concepts which are important for access control in today’s open and dynamic environments: the context-dependent role activation/revocation, the purpose of resource access, and the whole range of dynamic attributes relevant to access control. However, our framework supports these mentioned features. In addition, our framework also supports resource hierarchy, and the user’s access to resources at different levels of resource granularity which are also important aspects of access control for improving privacy and security.

A number of further research efforts (e.g., [21],[22],[23]) have extended the Attribute-based Access Control (ABAC) approach to provide access control to resources in a context-aware manner. Corradi et al [21] have proposed a CAAC model for ubiquitous environments, considering context information: user location, user activities, user device, time, resource availability and resource status. Hulsebosch et al [22] have proposed a context-sensitive access control (CSAC) framework based on the user’s location and access history. These approaches also have limitations in considering a limited set of context information. A recent CAAC framework for the Web of data is grounded on three important dimensions of context: user, device and environment [23]. These attribute-based approaches consider dynamic context information in the access control policies, where permissions are directly associated with contexts. Differently from these attribute-based approaches, we consider the relevant dynamic attributes in the user-role and role-permission assignment policies. These attribute-based approaches have major limitations when applied in open large-scale domains because of the huge number of attributes involved. In addition, they do not directly treat roles as first class entity, therefore they are not able to offer the advantages of role-based access control (i.e., the role hierarchy, role inheritance, etc.). In addition, we support resource hierarchy and different levels of resource granularity.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have explained the need for a policy framework for modern dynamic environments and presented a new policy framework, an extension of the RBAC framework, which satisfies such a need. We have presented an expressive formal language for specifying our framework including its basic elements, syntax and semantics. By introducing the concept of context-dependent role activation/revocation, the context-aware association of users to roles can be achieved. A role is activated or deactivated based on the relevant context information. Moreover, the concept of purpose-oriented situation has been introduced in order to achieve context-dependent roles to permissions assignment. By following the proposed formal model concepts, we have developed an ontology-based policy framework for defining and enforcing two sets of policies that take into account relevant contexts and situations: the context-aware user-role and role-permission assignment policies. The first set of policies specifies that users having roles are allowed to carry out an operation on the resources when a set of conditions are satisfied. When a user wants to access resources, policy enforcement determines an access request is granted or denied. We have also presented a prototype implementation of our framework and validated the framework through a case study in the healthcare domain.

As future work, a general system prototype implementing our framework with a user-friendly front-end will be developed so that it can be used by software engineers to build the context-aware access control applications.

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