An ontological framework for situation-aware access control of software services

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

School of Software and Electrical Engineering, Swinburne University of Technology, Victoria 3122, Australia

ARTICLE INFO

Available online 30 April 2015

Keywords:
Situation-awareness
Context information
Purpose
Situation model
Access control policy
Policy model
Situation-aware access control

ABSTRACT

Situation-aware applications need to capture relevant context information and user intention or purpose, to provide situation-specific access to software services. As such, a situation-aware access control approach coupled with purpose-oriented information is of critical importance. However, modelling purpose-oriented situations is a challenging task. Existing modelling approaches for situation-aware systems are not adequate to express purpose-oriented situations. Furthermore, existing context/situation-aware access control approaches are highly domain-specific and do not consider purpose-oriented information.

In this paper we consider purpose-oriented situations rather than conventional situations (e.g., user’s state) in proposing a generic situation-aware access control framework for software services. We take situation to mean the states of the entities and their relationships that are relevant to the purpose of a resource access request. Our framework includes a situation model specific to access control, identifying the relevant purpose-oriented situation information. Using the situation model, the policy model of the framework provides support for specifying and enforcing situation-aware access control policies. A software prototype has been developed to demonstrate the practical applicability of the framework. In addition, we demonstrate the general applicability of our framework through two case studies from different domains. Experiments are conducted to quantify the performance overhead of providing such situation-aware access control for software services.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

A security policy normally states that a particular service can be invoked based on the states of the relevant context entities and the specific purpose, which describes the reason for which the organizational resource is used [2]. By identifying the user’s intention in accessing software services, we can achieve purpose-oriented control over access to such services. In dynamic and context-aware environments, therefore, situation-aware access control (SAAC) applications need to take into account the relevant context information [3] and user intention or purpose, to provide situation-specific access to software services. In such environments, users demand access to software services in an anytime and anywhere fashion, as described by Weiser [4], and yet not to compromise the relevant privacy and security requirements of the stakeholders.

For example, a doctor’s request to access a patient’s medical records through a healthcare application may be
appropriate and allowed from the inside of the hospital when the patient is assessed by the doctor, but may not be from a public bus for reviewing patient cases. However, such service access request may also be granted for the emergency treatment purpose even if it is on a bus. In the medical domain the American Health Information Management Association (AHIMA) identifies 18 health care scenarios across 11 purposes (treatment, payment, research, etc.) for health information exchange [5]. Therefore, in order to specify situations for SAAC applications, on the one hand, it is required to capture the states of the relevant situation-specific context entities (e.g., user, resource, resource owner) and their relevant relationships (e.g., the interpersonal relationships between the user and the resource owner); on the other hand, it is required to identify the purpose or user’s intention in accessing the software services.

Access control is one of the fundamental security mechanisms needed to protect information resources and software services. It determines whether a request to access the resources and services provided by a system should be permitted or denied. A well-accepted traditional access control model based on the roles of the users is role-based access control (RBAC), which has been introduced to tackle the problem of identity-based access control for managing and enforcing security in large-scale domains [6,7]. However, the basic RBAC approach does not provide adequate functionality to incorporate and adapt to dynamically changing contextual information. On the other hand, the basic attribute-based access control (ABAC) approach grants accesses to resources and services based on attributes of entities (e.g., the attributes possessed by the requester) [8]. The ABAC approach has similar drawbacks in supporting the context-awareness. Besides, the attribute-based approaches are not suitable in large-scale domains because they do not scale well in large open systems [9].

Context-aware access control is one of the security mechanisms needed to provide flexible control for users’ access to resources and services according to the currently available contextual information [10]. During the past decades, a number of research efforts have extended the basic RBAC approach [7] by incorporating some specific types of contextual information: temporal information (e.g., [11,12]), spatial information (e.g., [13]), and both the time and location (e.g., [14]). Recently, Kulkarni et al. [15], He et al. [16], Huang et al. [17], and Schefer-Wenzl et al. [18] have adopted and extended the basic RBAC approach with some further contextual information from other than the temporal and spatial dimensions, including the resource and environment dimensions as well as the user dimension. Several research efforts (e.g., [19–22]) extend the basic ABAC approach with context-awareness by modelling contextual aspects of the user, resource and environment dimensions as attributes. These extended role-based and attribute-based access control approaches are however still limited in identifying situations.

In the field of situation representation models, there have been several research works for modelling situation information. Endsley [23] defines the basic components to achieve situation-awareness, “the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future”. Some other research works describe situation as the states of the specific kinds of entities (e.g., [24–26]). Though these situation modeling approaches can specify conventional situations (e.g., the user’s state), they are not adequate to specify purpose-oriented situations.

Some situation-aware access control approaches have been proposed in the access control literature (e.g., [27,28]), with each of them having different origins, pursuing different goals and often, by nature, being highly domain-specific. They consider the specific types of context information (e.g., the user’s state) as policy constraints to control access to software services or resources. However, other than the relevant entity states, the states of the relevant relationships between entities are not considered. In addition, the purpose or user’s intention in accessing the services is not considered in these works. In this paper, we consider the basic elements of situation-aware access control: the relevant entity states, the relevant relationship states and the purpose or user’s intention, and their combination.

In general, the existing access control approaches for information resources and software services have only considered specific types of context information as policy constraints. As such, a new situation model for access control is required to identify purpose-oriented situation information. Furthermore, a policy model for access control that incorporates purpose-oriented situations into the access control process is also required, in order to provide more targeted service access permissions to users.

### 1.1. Our contributions

The above identified research issues and challenges motivate us to develop a new situation-aware access control framework for software services. In this paper, we present a novel framework, Purpose-Oriented Situation-Aware Access Control (PO-SAAC), to provide the capability to make access control decisions for software services by taking into account the purpose-oriented situation information. It makes the following key contributions:

(C1) **Purpose-oriented situation model**: Our framework uses the purpose-oriented situation information in determining situation-specific access to software services (authorization), where we present a situation model to represent and reason about the different types of situations. The purpose-oriented situation can be composed of the relevant states of the entity and states of the relationships between entities and the user’s intention or purpose.

(C2) **Situation-aware access control policy model**: Our framework presents a SAAC policy model to specify situation-aware access control policies. The policy model supports access control to the appropriate software services based on the relevant situations.

(C3) **Ontology-based framework implementation**: Based on
1.2. Paper outline

The rest of the paper is organized as follows. Section 2 presents a healthcare application scenario to motivate our work. In Section 3, we present our approach to purpose-oriented situation-aware access control (PO-SAAC) framework, including a situation model for specifying different situations and a policy model for specifying situation-specific access control policies. Section 4 presents an ontology-based development platform for our framework. We then present the prototype implementation of our framework in Section 5. In Section 6, we present two case studies from the healthcare and university domains. The experiments that quantify the performance of our framework are discussed in Section 7. Section 8 discusses the related work and presents a comparative analysis of our framework with the related access control frameworks. Finally, Section 9 concludes the paper and outlines future work.

2. Research motivation and general requirements

In this section we first outline a motivating scenario that illustrates the need for the incorporation of purpose-oriented situation information in access control policies. We then identify the general requirements for managing the access to software services in a situation-aware manner.

2.1. Motivating scenario

To exemplify the complexity of achieving situation-awareness in access control systems, let us consider the area of patient medical records management (PMRM) in the healthcare domain as a motivating scenario.

- **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 doctor, Jane, a general practitioner at the hospital, is required to treat Bob and needs to access Bob’s emergency medical records from the emergency room of the hospital.

RBAC [7] is the most widely used traditional access control technique that evaluates access permission through roles assigned to users by taking into account the static user-role and role-permission assignments. It does not have functionality to integrate dynamically changing contextual information in the access control process. The different types of information involved in this scenario are highly dynamic, and granting a user access without taking the dynamic information into account can compromise security. In a traditional setting, RBAC will not grant Jane the access to the requested service (access to Bob’s emergency medical records) because Jane is not Bob’s usual treating doctor. However, to save life, permission should be granted under such specific (dynamic) conditions (e.g., heart attack, the emergency room, general practitioner as requester). To do so, we need situation-aware access control policies. One of the relevant situation-aware access control policy is shown in Table 1 (see Policy 1).

<table>
<thead>
<tr>
<th>No.</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A general practitioner who is a treating doctor of a patient, is allowed to read/write the patient’s emergency medical records in the hospital for an emergency treatment purpose. However, in an emergency situation (like Scene 1), all general practitioners should be able to access the patient’s emergency medical records in the hospital (by playing the emergency doctor role).</td>
</tr>
</tbody>
</table>

The situation and policy models, we introduce an ontology-based platform for modelling and identifying purpose-oriented situations, and for specifying and enforcing situation-aware access control policies that take into account the relevant situations. Our ontology-based framework represents the basic elements using the ontology language OWL, extended with SWRL for identifying and reasoning about relevant situations and the corresponding access control policies.

Other than the above three major contributions, this paper also includes the following research aspects:

1. **Prototype implementation**: In order to demonstrate the practical applicability of our approach, we have presented a prototype implementation for our framework, providing the basis for developing individual situation-aware access control applications.

2. **Case studies**: We have carried out a case study using our prototype framework for a patient medical records management system. We have also presented another case study from the university domain, to demonstrate the general applicability of our framework.

3. **Performance evaluation**: In addition, we have conducted a number of experiments in a simulated healthcare environment, to quantify the performance overhead of our framework by measuring the system’s response time. The experimental results have shown the satisfactory performance of our proposed framework.

4. **Comparative assessment**: In addition to the case studies and performance evaluation, we have presented a comparative analysis of the existing access control frameworks. The comparative assessment has shown that our framework offers a range of new capabilities for situation-aware access control.
The different types of dynamically changing information are also involved in this scenario (e.g., the 'patient-registered nurse' relationship, in the general ward, and on duty). The corresponding situation-aware access control policy is shown in Policy 2 in Table 2.

The above scenario and their related policies involve a number of constraints, including the user role (e.g., emergency doctor, registered nurse), the relevant environmental information (e.g., the emergency room, the interpersonal relationship between doctor and patient), the resource/service (e.g., emergency medical records, daily medical records), and the purpose/user's intention in accessing the services (e.g., emergency treatment, daily operation). Furthermore, the policies refer to the need for these constraints to be evaluated in conjunction with these relevant information.

The above-mentioned PMRM (patient medical records management) application is an example of how to realize situation-aware access control decisions. In this application scenario, only the situation-aware access control policies for the general practitioners and registered nurses (2 roles) are considered. In the overall PMRM application, the number of policies can reach up to 500 with respect to 138 different health professional roles [29]. The detail analysis of the PMRM application and case study are presented in Sections 5 and 6.

2.2. General requirements

Looking at the above-mentioned application scenario and their related policies, we can make some important observations concerning situation-aware access control.

To support situation-aware access control in a computer application like the patient medical records management system, we need to consider the 3Ws: who (the appropriate users by playing the appropriate roles) wants to access what (the appropriate services), and when (the relevant states and the purpose or user's intention in accessing the services). In particular, a general purpose-oriented situation-aware access control framework is required to manage the access to services in such applications by taking into account the different types of relevant situations. As different types of information are integrated into the access control processes, some important issues should be considered as part of building situation-aware access control? In general, there is a need to identify the user's intention or purpose information using the available context information. How to model and identify the basic purpose-oriented situations in an effective way? There is a need to formulate a dynamic situation model to represent the atomic purpose-oriented situations.

Reasoning about composite situations: How to express user-defined reasoning rules to infer a new composite situation from the atomic situations, thereby extending the dynamic situation model? There are some complex situations that are involved in the access control process, are not directly obtainable but can be inferred from the one or more already identified situations. Thus, there is a need to express user-defined reasoning rules to infer the complex and composite situations, and extend the dynamic situation model with these newly created rules.

Specification of situation-aware access control policies: How to specify the access control policies that take into account purpose-oriented situations to realize a flexible and dynamic access control scheme? In the application scenario, for example, when the situation changes (e.g., Bob has come out of emergency and moved to a general ward), decisions on further access control to resources (e.g., Jane access to Bob's emergency medical records by playing the general practitioner role) may change accordingly (e.g., denied). Thus, there is a need to specify access control policies based on the relevant situation information.

Enforcement of access control policies: How to evaluate the access control policies based on the relevant situation information present to realize a flexible and dynamic access control scheme? There is a need for an access controller to evaluate the situation-aware access control policies based on the relevant purpose-oriented situation information.

Implementation framework: How to realize the relevant situations and the corresponding situation-specific access control policies in an effective way, in order to access/manage software services? A general software platform is required to enable the development of specific situation-aware access control applications.

<table>
<thead>
<tr>
<th>No.</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A registered nurse within a hospital is granted the right to read/write a patient's daily medical records during her ward duty time and from the location where the patient is located for a daily operational purpose.</td>
</tr>
<tr>
<td>2</td>
<td>A general software platform is required to enable the development of specific situation-aware access control applications.</td>
</tr>
</tbody>
</table>
arise. We examine these issues and their related requirements as follows:

In order to address the above-identified requirements, we introduce in the following sections (i) a formalization of the basic concepts of our purpose-oriented situation-aware access control (PO-SAAC) approach, including the situation awareness and policy models for access control and (ii) an ontology-based PO-SAAC framework that can provide access control support for software services in a situation-aware manner.

3. Purpose-oriented situation-aware access control

The basic steps of developing our PO-SAAC framework for providing situation-specific access to services are summarized in Fig. 1.

Level 1 PO-SAAC – Capture relevant information: It is the process of capturing elementary information from the environment [30], both the “state” of relevant entities and relationships between entities and the “purpose” of accessing services, brought together into an integrated whole to form purpose-oriented situations.

Level 2 PO-SAAC – Include situation semantics: The second step in achieving purpose-oriented situation-awareness is to assign meaning or include situation semantics, i.e., include the understanding of the significance of the elementary information (Level 1 data/information) as a network of relationships to form a holistic picture of the world (relevant environment).

Level 3 PO-SAAC – Infer new situation: It is the process of inferring new composite situations which are not directly obtainable. This is achieved by using the basic situation information from Level 1 and the knowledge from Level 2.

Level 4 PO-SAAC – SAAC policy: It is the ability to define and apply situation-aware access control policies to make access control decisions based on the relevant situations (where situations act as a condition), and consequently take necessary action in a timely and effective manner. This is the fourth and highest level of the purpose-oriented situation-awareness for access control.

In the following we present a formalization of the basic concepts of our purpose-oriented situation-aware access control (PO-SAAC) framework, including the purpose-oriented situation model and SAAC policy model for software services.

3.1. Purpose-oriented situation model

A situation consists of a set of elementary information concerning the states of relevant entities and relationships and the user’s intention or purpose. We classify situations that are used in specifying situation-specific access control policies, into simple situations (atomic situations) and complex situations (composite situations).

3.1.1. Representation of situation

Different atomic situations can be defined based on the domain-dependent data/information from the application domain concerned (e.g., the healthcare domain) through the development of computer-supported models.

**Definition 1** (Atomic situation, \( S_a \)). A Situation used in an access control decision is defined as the states of the relevant entities and the states of the relevant relationships between entities at a particular time that are relevant to a certain goal or purpose of a resource access request. A Purpose is the user’s intention in accessing software services. The Situation and Purpose are domain-dependent concepts, and their values can be obtained based on the access request (i.e., from the sensed contexts, inferred contexts, etc.):

\[
S_a = \{s_{a1}, s_{a2}, s_{a3}, \ldots, s_{a4}\}
\]

where ‘\( S \)’ is the set of atomic situations and ‘\( s_{a1}, s_{a2}, s_{a3}, \ldots, s_{a4} \)’ are atomic situations.

An atomic situation ‘\( s_{ai} \)’ is the logical conjunction of a purpose ‘\( p_j \)’ and a set of states ‘\( St \)’:

\[
s_{ai} = p_j \land St
\]

where ‘\( p_j \)’ (\( p_j \in P \)) is a purpose (see Definition 2) and ‘\( St \)’ denotes a set of relevant states (the states of the entities and/or the states of the relationships between entities).

Consider our application scenario Scene 1, which is presented in Section 2, where some of the relevant states are the state of an entity, e.g., ‘\( \text{location(Jane)} = \text{“Hospital”} \)’, the state of the relationship between entities, e.g., ‘\( \text{interpersonalRelation(Jane, Bob)} = \text{“NonTreatingDoctor”} \)’.

**Definition 2** (Purpose, \( P \)). A Purpose is the user’s intention in accessing information resources or software services. Purpose is a domain-specific concept, and its value can be obtained in relation to the access request:

\[
P = \{p_1, p_2, p_3, \ldots, p_j\}
\]

where ‘\( P \)’ is the set of purposes and ‘\( p_1, p_2, p_3, \ldots, p_j \)’ are individual purposes.

Consider the same application scenario Scene 1, where the purpose of access is \( p_1 = \text{“EmergencyTreatment”} \).

In our situation model, a purpose ‘\( p_j \)’ (\( p_j \in P \)) can be identified based on the relevant states (i.e., the states of the relevant entities and the states of the relevant relationships between entities). The states are formed by using the currently available context information. This inferred purpose and the other relevant states based on the available context information (which is not used for the identification of purpose) are used to form a situation (see Examples 1 and 2).

**Example 1.** Consider Policy 1 of our application scenario: a user, who is a general practitioner, by playing an emergency doctor (ED) role can access a patient’s emergency medical records (EMR) in the hospital for an emergency...
treatment (ET) purpose, when the patient is in a critical condition. The following rule (4) is used to identify that the purpose is ‘ET’ (i.e., the purpose of a user playing the ‘ED’ role in accessing the medical records of a patient who is in a critical condition is ‘ET’),

\[ \text{Purpose}(p_1) \land \text{User}(u) \land \text{Resource}(res) \land \text{Owner}(o) \land \text{Role}(r) \land \text{hasRole}(u, r) \land \text{equal}(r, "ED") \land \text{isOwnedBy}(res, o) \land \text{healthStatus}(o,"critical") \Rightarrow \text{equal}(p_1, "ET"). \]

Example 2. Continuing from Example 1, the atomic situation concerned \((s_{a1} \in S_1)\) is that the user in the hospital wants to access a patient’s medical records for an emergency treatment purpose (‘ET’) (see rule (5)),

\[ s_{a1} = \text{User}(u) \land \text{Purpose}(p_1) \land \text{intendedPurpose}(u, p_1) \land \text{equal}(p_1, "ET") \land \text{hasLocation}(u, l) \land \text{equal}(l, "Hospital"). \]

3.1.2. Reasoning about situation

The process of inferring a new composite situation (complex situation) from one or more already defined atomic situations is referred to as reasoning about situation. One of the main advantages of our situation model is its reasoning capability. That is, once facts about the world are stated, other facts can be inferred using an inference engine through user-defined reasoning rules.

Definition 3 (Composite situation, \(S_c\)). Given a collection of atomic situations of the same purpose, a composite situation can be defined by performing logical composition (AND, OR or NOT) on one or more already defined atomic situations (i.e. each of the current atomic situations can be a composite situation or an atomic situation). For example,

\[ \begin{align*}
S_{c1} &= S_1 \land S_2 \\
S_{c2} &= S_3 \lor S_4 \\
S_{c3} &= \neg S_5
\end{align*} \]

where \(S_1, S_2, S_3, S_4\), and \(S_5\) are already defined atomic or composite situations \((S_i \in S_0\) or \(S_i \in S_c\)), and \(s_{c1}, s_{c2}, \) and \(s_{c3}\) are new composite situations.

In general,

\[ S_c = \{s_{c1}, s_{c2}, s_{c3}, \ldots, s_{c_k}\} \]

where ‘\(S_c\)’ is the set of composite situations and ‘\(s_{c1}’, ‘s_{c2}’, \ldots, ‘s_{c_k}\)’ are composite situations.

Example 3. Consider Policy 2 of our application scenario from Section 2: a user, by playing a registered nurse (RN) role, is granted the right to read/write a patient’s daily medical records (DMR) during her ward duty time (DT) and from the general ward (GW) where the patient is located, for a daily operational (DO) purpose. The daily operation (DO) purpose can be identified using the following rule (8). The rule specifies that a user by playing the ‘RN’ role can access a patient’s medical records for ‘DO’ purpose, when the patient’s health condition is normal:

\[ \begin{align*}
\text{Purpose}(p_2) \land \text{User}(u) \land \text{Owner}(o) \land \text{Resource}(res) \land \text{Role}(r) \land \text{hasRole}(u, r) \land \text{equal}(r, "RN") \land \text{hasLocation}(u, l) \land \text{equal}(l, "GW") \land \text{healthStatus}(o,"normal") \Rightarrow \text{equal}(p_2, "DO").
\end{align*} \]

Two atomic situations (rules (9) and (10)) regarding the mentioned policy are represented as follows:

\[ \begin{align*}
S_{a2} &= \text{User}(u) \land \text{Purpose}(p_2) \land \text{intendedPurpose}(u, p_2) \\
&\land \text{equal}(p_2, "DO") \land \text{Location}(l) \land \text{hasLocation}(u, l) \land \text{equal}(l, "GW") \land \text{hasRole}(u, "RN") \land \text{healthStatus}(o,"normal") \\
S_{a3} &= \text{User}(u) \land \text{Purpose}(p_2) \land \text{intendedPurpose}(u, p_2) \\
&\land \text{equal}(p_2, "DO") \land \text{Time}(t) \land \text{hasLocation}(u, l) \land \text{equal}(l, "GW") \land \text{hasRole}(u, "RN") \land \text{healthStatus}(o,"normal") \\
&\land \text{equal}(p_2, "DO") \land \text{equal}(t, "DT").
\end{align*} \]

An example policy associated with the situation ‘\(s_{a2}\)’ (rule (9)) can be read as, a user by playing a registered nurse (RN) role, who is located with a patient in the general ward (GW) of the hospital, can access the patient’s daily medical records (DMR) for a daily operational (DO) purpose, when the patient’s health condition is normal.

An example policy associated with the situation ‘\(s_{a3}\)’ (rule (10)) can be read as, a user by playing the ‘RN’ role can access the patient’s ‘DMR’ during her ward duty time (DT) for ‘DO’ purpose, when the patient’s health condition is normal.

A composite situation ‘\(s_{c1}\)’ \((s_{c1} \in S_c\) with these two atomic situations \((s_{a2} \land s_{a3})\) can be identified using the following logical conjunction:

\[ s_{c1} = s_{a2} \land s_{a3}. \]

An example policy associated with the situation ‘\(s_{c1}\)’ (rule (11)) can be read as, a user by playing the ‘RN’ role, who is co-located with a patient in the general ward (GW) of the hospital, can access the patient’s private medical records (PMR) during her ward ‘DT’ for ‘DO’ purpose, when the patient’s health condition is normal.

3.2. Software services

A service is a self-contained software entity and may be composed of other services (service composition). We consider the resource (e.g., patient medical record) in a service oriented manner, in order to provide fine-grained access control and grant the right access to the appropriate parts of a resource by the appropriate users. A service can be seen as a \((\text{res}, \text{op})\) pair with ‘res’ being a requested resource and ‘op’ being the action/operation on the resource. For example, the write operation on the emergency medical records is defined as \(\langle \text{EMR}, \text{write} \rangle\). In this way, the fine-grained access control to resources can be realized by managing the access to the service operations.
3.3. SAAC policy model for software service

A recent study [31] shows that the basic role-based access control (RBAC) model has become the most widely used access control model and the most attractive solution for providing security features.

Based on the formalization of the role-based access control (RBAC) model in [7], we present a formal definition of our policy model for situation-aware access control. Our policy model extends RBAC with relevant situations, which are defined in the previous section. Our goal in this research is to provide a way in which the role-permission assignment policies can be specified by incorporating dynamic situations as policy constraints.

**Definition 4 (SAAC policy model).** Our situation-aware policy model for access control can be formally described as a tuple (Formula (12)):

\[
M_{SAAC} = (R, S, Ser, SAACPolicy)
\]  

where R, S, Ser, and SAACPolicy represents Roles, Situations, Services, and Policies, respectively.

1. **Roles (R):** R represents a set of roles. A role reflects user’s job function or job title within the organization. A user is a human-being who is a service requester and whose service access request is being controlled:

\[
R = \{r_1, r_2, r_3, \ldots, r_m\}
\]  

2. **Situation (S):** S represents a set of situations that are specified by using the situation model. S is used to express the relevant situations (atomic situations \(S_a\) and composite situations \(S_c\)) in order to describe the SAAC policies:

\[
S = \{s_1, s_2, s_3, \ldots, s_n\} = S_a \cup S_c
\]  

3. **Services (Ser):** Ser represents a set of services. In our policy model, a service is a well-defined and self-contained software entity with an invocable interface to provide certain capability to perform certain operations on resources:

\[
Ser = \{ser_1, ser_2, ser_3, \ldots, ser_n\} = \{res \mid res \in Res, op \in OP\}
\]  

where ‘Res’ is a set of component parts of resources and ‘OP’ is a set of operations on the resources:

\[
Res = \{res_1, res_2, res_3, \ldots, res_p\}
\]

\[
OP = \{op_1, op_2, op_3, \ldots, op_q\}
\]

4. **Policies (SAACPolicy):** SAACPolicy represents a set of situation-aware role-service assignment policies to provide situation specific access to software services. The SAAC policies specify the dynamic assignments of permissions (service access permissions) to users (through assigning roles) when a set of situations are satisfied:

\[
SAACPolicy = (sp_1, sp_2, sp_3, \ldots, sp_{m})
\]

\[
= \{(r_1, ser_1, s_1), (r_2, ser_2, s_2), \ldots, (r_m, ser_m, s_n)\} \subseteq R \times Ser \times S
\]  

Our policy model extends the concept of common role-permission assignments (RPA) in RBAC (\(RPA \subseteq R \times P\)) [7], by introducing the concept of purpose-oriented situation, leading to situation-aware role-service assignments (\(SAACPolicy \subseteq R \times Ser \times S\)).

**Example 4.** Based on our policy model (\(Role(r) \wedge Service\(\langle ser, sp \rangle \wedge Situation(s) \rightarrow (r, ser, s) \in SAACPolicy\)) the rule shown in Table 3 defines the policy described in Example 1, i.e., a User ‘u’ by playing the Role ‘r’ (emergency doctor (ED) role) can invoke the Service ‘ser’ (writeEMR() service), if a Situation ‘s’ (\(s_{a1}\) as defined in Example 2) is satisfied.

4. Ontology-based PO-SAAC framework

In this section, we introduce an ontology-based PO-SAAC framework by using semantic technologies to model relevant purpose-oriented situations and situation-specific access control policies. The principal goal of our framework is to provide a practical basis for realizing the situation-aware access control concepts and applications using ontology-based modelling languages (e.g., OWL and SWRL).

4.1. Design considerations for the PO-SAAC framework

Our PO-SAAC ontology is capable of representing core concepts and domain-specific concepts, supporting reasoning about situations according to ontology-based reasoning rules, and specifying access control policies. Domain-specific concepts are the specializations of core concepts for specific domains. This separation between the core concepts and the domain-specific concepts encourages the reuse of core concepts, and provides a flexible basis for specifying domain-specific concepts.
• **Core concepts** represent the general concepts and the relationships between these concepts that are specifically identified from the access control perspective.

• **Domain-specific situation concepts** represent the situation concepts and the relationships between these concepts for a specific domain that are defined based on the core concepts.

• **Reasoning rules** express the user-defined rules to infer the purpose and atomic situations based on the currently available context information, and to derive composite situations that are not explicitly present but can be inferred from available atomic situations.

• **Policy rules** specify the situation-aware access control policies for dynamic access control decision making.

In the literature, there are many languages that have been developed for specifying and reasoning about computer-processable semantics. The ontology-based modelling technique has been proven to be a suitable logical language for modelling dynamic contexts and situations (e.g., [32,33]). The ontology-based modelling approach to achieve situation-awareness (e.g., [25,26]) is not only beneficial from the representational viewpoint but also from the reasoning viewpoint; that is, once facts about the world have been stated in terms of the ontology, other facts can be inferred using an inference engine through inference rules.

To model the PO-SAAC ontology, in this paper, we adopt the OWL language [34] as an ontology language to represent the situations, which has been the most practical choice for most ontological applications because of its considered trade-off between expressiveness and computational complexity of reasoning [33]. In order to support the process of inferring new composite situations, we need to define a set of reasoning rules that are associated with the existing or already defined situations. However, some of the reasoning rules require mathematical computation, which is not supported by the OWL language. Towards this end, the expressivity of OWL can be extended by adding SWRL rules [35]. We express the user-defined reasoning rules using the SWRL which provide the ability to identify the purposes and to reason about new composite situations.

### 4.2. Situation ontology

A graphical representation of the core situation ontology is shown in Fig. 2. The ontology facilitates software engineers to analyze and specify purpose-oriented situation information of service invocation for access control in a situation-aware manner. The ontology is divided into two layers. The bottom layer (Layer 1) shows the core concepts/elements for defining the context information specific to access control, which are already discussed in our previous work [30]. The top layer (Layer 2) shows the core concepts for specifying the relevant situations (atomic and composite situations) by

![Fig. 2. The core situation ontology.](image-url)
using the relevant information from Layer 1. The ontology models the following core concepts.

### 4.2.1. Core situation concepts

The top layer has the following situation modelling concepts, which are organized into a Situation hierarchy, namely State, Purpose, AtomicSituation, and CompositeSituation classes.

A Situation consists of the relevant States and the Purpose of user’s access request. A Purpose is a user’s intention in accessing the resources or services; and it can be identified based on the currently available context information. A State can be composed of the relevant context information. Definition 5 specifies the Situation definition in OWL. It shows that the Situation class has an object property consistsOf, which is used to link the Situation and the union of Purpose and State classes.

A Situation can be either a simple situation (an AtomicSituation) or a complex situation (a CompositeSituation). A CompositeSituation can be composed of one or more already defined atomic or composite situations using logical operators (AND, OR and NOT). The AtomicSituation and CompositeSituation classes are the subclasses of the Situation class. A built-in property subClassOf is used to relate the Situation class and its subclasses (see Definition 6).

**Definition 5 (Situation definition in OWL).**

```owl
<owl:Class rdf:ID=”Situation”>
  <owl:ObjectProperty rdf:ID=”consistsOf”>
    <rdfs:domain rdf:resource=”#Situation”/>
    <rdfs:range>
      <owl:unionOf rdf:parseType=”Collection”>
        <owl:Class rdf:about=”#Purpose”/>
        <owl:Class rdf:about=”#State”/>
      </owl:unionOf>
    </rdfs:range>
  </owl:ObjectProperty>
</owl:Class>
```

**Definition 6 (Situation class and its two subclasses).**

```owl
<owl:Class rdf:ID=”AtomicSituation”>
  <rdfs:subClassOf rdf:resource=”#Situation”/>
</owl:Class>
<owl:Class rdf:ID=”CompositeSituation”>
  <rdfs:subClassOf rdf:resource=”#Situation”/>
</owl:Class>
```

To achieve fine-grained control over access to services, the different purposes at various granularity levels (purpose hierarchy) need to be identified. Towards this end, we use an object property hasSubPurpose to model the purpose hierarchy. Definition 7 shows that the class Purpose has an object property hasSubPurpose.

**Definition 7 (‘hasSubPurpose’ object property definition).**

```owl
<owl:ObjectProperty rdf:ID=”hasSubPurpose”>
  <rdfs:domain rdf:resource=”#Purpose”/>
  <rdfs:range rdf:resource=”#Purpose”/>
</owl:ObjectProperty>
```

An object property intendedPurpose is used to link User and Purpose classes (not shown in Fig. 2 for the figures clarity). Definition 8 shows that the class User has an object property intendedPurpose. The details of the different purposes at various granularity levels are presented in Section 4.2.2.

**Definition 8 (‘intendedPurpose’ object property definition).**

```owl
<owl:ObjectProperty rdf:ID=”intendedPurpose”>
  <rdfs:domain rdf:resource=”#User”/>
  <rdfs:range rdf:resource=”#Purpose”/>
</owl:ObjectProperty>
```

The bottom layer of our situation ontology (see Fig. 2) has the following core concepts of the context entities and context information. The core entities Person, Object, and Place are organized into the ContextEntity hierarchy. The different entities of User and Owner are organized into the Person hierarchy. The Resource, Relationship and Role are organized into the Object hierarchy. The relationship between a Resource and its Owner is represented by an object property named isOwnedBy. A isPlayedBy object property is used to relate the Role and User classes for representing the fact that a user has role. A context characterizes the ContextEntity (e.g., the location of user) or the Relationship between different entities (e.g., the interpersonal relationship between user and owner). The contexts (ContextInfo) are represented by a number of context information types, namely RelationshipInfo, StatusInfo, TemporalInfo, and LocationInfo. To specify the different relationships between different entities (persons), an object property hasRelationship is used which links the Person and Relationship classes. This general and extensible context model specific to access control is proposed in our earlier paper [30].

### 4.2.2. Inferring purpose

The core situation ontology (shown in Fig. 2) serves as an entry-point for the domain ontologies. The domain-specific concepts extend the core ontology’s corresponding general concepts. It is important for the application developers, providing a way to include domain-specific concepts into the core ontology. Fig. 3 shows an excerpt of the representation of the Purpose ontology for the healthcare domain (e.g., treatment purpose, daily operation purpose, and research purpose) to exchange patients’ medical records.

The different purposes at various granularity levels of a user’s service access request are individually identifiable, so as to achieve fine-grained control over access to services. As such, the Purpose class contains an important
data type property (xsd:int type) named granularityLevel, which indicates the granularity level (see Fig. 3). By doing so, we can provide different levels of purpose granularity. For example, an Emergency Doctor can access a patient’s emergency medical records for the Treatment purpose, at granularityLevel 0 (highest level), which means she also can access for the more specific purposes of NormalTreatment and EmergencyTreatment at the lower granularity levels. On the other hand, a General Practitioner can access some of a patient’s medical records (e.g., daily medical records) for the NormalTreatment (or NT) purpose. However, she cannot access a patient’s medical records for EmergencyTreatment (or ET) purpose.

Various types of ontology-based inferences can be performed for the identification of different purposes. Our situation ontology is extended with a set of user-defined SWRL rules to infer purpose or user’s intention in accessing the services. A purpose is identified based on the available context information. An example SWRL rule shown in Table 4 identifies that a user’s intended Purpose is EmergencyTreatment (or ET), based on the relevant context information.

Another example SWRL rule shown in Table 5 identifies that a user’s intended Purpose is DailyOperation (or DO), based on the relevant context information.

We also consider the granularity levels of the healthcare Role and Resource (patient’s medical records) hierarchies, in order to facilitate fine-grained control for different types (roles) of users, so as to achieve fine-grained control over access to resource components at various granularity levels [30].

4.2.3. Modelling atomic situation

Consider an atomic situation that is associated with an access control policy for the general practitioner in an emergency doctor role (Policy 1 in our application scenario in Section 2). We have already formalized this atomic situation in Example 2 (rule (5)) and the associated purpose in Example 1 (rule (4)).

The atomic situation is based on the following constraints: the intendedPurpose of the requester or user (that is already identified in Table 4) and the locationAddress from which the user has initiated the request. An example policy associated with this atomic situation 'sa1' (Sa1 e Sa) specifies a general practitioner by playing an emergency doctor (ED) role can access a patient’s medical records from the hospital for an emergency treatment (ET) purpose. The specification of this atomic situation (named sa1_EDFromHospitalForET) in OWL is shown in Table 6.

Let us consider two atomic situations that are associated with an access control policy for the registered nurse (Policy 2 in our application scenario in Section 2). We have already formalized these two atomic situation in Example 3 (rules (9) and (10)).

The first atomic situation is based on the following constraints: the intendedPurpose of the user (that is already identified in Table 5), the locationAddress from which the user has initiated the request and the co-located relationship (co-located or not) between user and owner. An example policy associated with this atomic situation 'sa2' (Sa2 e Sa) specifies a registered nurse (RN) can access a patient’s daily medical records from the general ward (GW) for a daily operational (DO) purpose. The specification

Fig. 3. An excerpt of purpose ontology.

<table>
<thead>
<tr>
<th>Table 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A reasoning rule that captures the Purpose is ET.</td>
</tr>
</tbody>
</table>

| Purpose(?purpose) ∧ Role(?role) ∧ roleIdentity(?role, “EmergencyDoctor”) ∧ User(?user) ∧ hasRole(?user, ?role) ∧ Resource(?resource) ∧ Owner(?owner) ∧ isOwnedBy(?resource, ?owner) ∧ healthStatus(?owner, “Critical”) → intendedPurpose(?user, ?purpose) ∧ purposeName(?purpose, “ET”) |

<table>
<thead>
<tr>
<th>Table 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A reasoning rule that captures that the Purpose is DO.</td>
</tr>
</tbody>
</table>

| Purpose(?purpose) ∧ Role(?role) ∧ roleIdentity(?role, “RegisteredNurse”) ∧ User(?user) ∧ hasRole(?user, ?role) ∧ Resource(?resource) ∧ Owner(?owner) ∧ isOwnedBy(?resource, ?owner) ∧ healthStatus(?owner, “Normal”) → intendedPurpose(?user, ?purpose) ∧ purposeName(?purpose, “DO”) |
of this atomic situation (named \(s_{a2} \_ \text{RNFromGWForDO} \)) in OWL is shown in Table 7.

The second atomic situation is based on the following constraints: the intendedPurpose and requestTime of the user. An example policy associated with this atomic situation ‘\(s_{a3} \) (\(s_{a3} \in S \)) specifies a registered nurse can access a patient’s daily medical records during her ward duty time (DT) for a daily operational (DO) purpose. The specification of this atomic situation (named \(s_{a3} \_ \text{RNAtDTForDO} \)) is shown in Table 8.

Table 7
An atomic situation ‘RNFromGWForDO’.

Table 8
An atomic situation ‘RNAtDTForDO’.

A set of reasoning rules can be specified to capture implicit knowledge for reasoning. For example, the reasoning rule specified in Example 3 (see rule (11)) can be used to reason about a new composite situation \(s_{c1} = s_{a2} \cap s_{a3} \) (\(s_{c1} \subseteq S \)). The specification of these two atomic situations \((s_{a2} \text{ and } s_{a3}) \) in OWL is shown in Tables 7 and 8. An example policy associated with this composite situation \(s_{c1} \_ \text{RNFromGWAtDTForDO} \), which is also the reasoning rule, is shown in OWL/XML in Table 9.

4.3. Policy ontology

The identification of the relevant concepts to specify situation-aware access control policies is already discussed.
Section 3.3. In this section, we present a policy ontology for situation-aware access control (SAAC), supporting policy specification (integrating relevant dynamic information into the access control policies) and policy evaluation (determining access control decisions). The main goal of our policy ontology is to specify the access control policies by incorporating the purpose-oriented situation information. Our policy ontology is a hierarchical model that can be extended by a set of user-defined rules using application specific concepts/elements (e.g., healthcare domain concepts).

4.3.1. Core policy concepts

We model our policy ontology based on the 3Ws: \textit{who} (user/role) wants to access \textit{what} (resource/service) and \textit{when} (relevant situation). A graphical representation of the policy ontology is shown in Fig. 4, which shows the core concepts/elements for specifying the SAAC policies. Our policy ontology, as depicted in Fig. 4, has the following concepts which are organized into a hierarchy \textit{SAACPolicy}, including the \textit{Role}, \textit{Situation}, \textit{Service}, and \textit{AccessDecision} classes. The policy ontology uses the concepts \textit{User}, \textit{Role} and \textit{Resource} from the situation ontology introduced in the previous section (see Section 4.2). The reused classes are shown in unshaded ellipses (see Fig. 4).

The following code in OWL (see Definition 9) shows the class \textit{SAACPolicy} has an object property \textit{hasService}, which is used to link the classes \textit{SAACPolicy} and \textit{Service}.

\begin{definition}[\textit{hasService} object property definition].
<owl:Class rdf:ID="\textit{SAACPolicy}"/>
<owl:ObjectProperty rdf:ID="\textit{hasService}"
<rdfs:domain rdf:resource="#\textit{SAACPolicy}"/>
<rdfs:range rdf:resource="#\textit{Service}"/>
</owl:ObjectProperty>
\end{definition}

Similarly, we define three other object properties \textit{hasRole}, \textit{hasSituation} and \textit{hasDecision} (see Definitions 10–12). The property \textit{hasRole} is used to link the classes \textit{SAACPolicy} and \textit{Role}, the property \textit{hasDecision} links the classes \textit{SAACPolicy} and \textit{AccessDecision}, and the property \textit{hasSituation} links the classes \textit{SAACPolicy} and \textit{Situation}.

\begin{definition}[\textit{hasRole} object property definition].
<owl:ObjectProperty rdf:ID="\textit{hasRole}"
<rdfs:domain rdf:resource="#\textit{SAACPolicy}"/>
<rdfs:range rdf:resource="#\textit{Role}"/>
</owl:ObjectProperty>
\end{definition}

\begin{definition}[\textit{hasDecision} object property definition].
<owl:ObjectProperty rdf:ID="\textit{hasDecision}"
<rdfs:domain rdf:resource="#\textit{SAACPolicy}"/>
<rdfs:range rdf:resource="#\textit{AccessDecision}"/>
</owl:ObjectProperty>
\end{definition}

\begin{definition}[\textit{hasSituation} object property definition].
<owl:ObjectProperty rdf:ID="\textit{hasSituation}"
<rdfs:domain rdf:resource="#\textit{SAACPolicy}"/>
<rdfs:range rdf:resource="#\textit{Situation}"/>
</owl:ObjectProperty>
\end{definition}

An object property \textit{canInvoke} is used to link between \textit{User} and \textit{Service} classes. The following code in OWL shows the definition of \textit{canInvoke} property (see Definition 13).
The services are the set of operations on the resources, i.e., a Service consumes Resource to perform Operation. An object property consumes is used to link Service and Resource classes (see Definition 14) and the property toPerform links the classes Service and Operation (see Definition 15).

**Definition 13** ('canInvoke' object property definition).

```owl
<owl:ObjectProperty rdf:ID='"canInvoke"'>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range rdf:resource="#Service"/>
</owl:ObjectProperty>
```

**Definition 14** ('consumes' object property definition).

```owl
<owl:Class rdf:ID='"Service"'>
  <owl:ObjectProperty rdf:ID='"consumes"'>
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="#Resource"/>
  </owl:ObjectProperty>
</owl:Class>
```

**Definition 15** ('toPerform' object property definition).

```owl
<owl:Class rdf:ID='"Operation"'>
  <owl:ObjectProperty rdf:ID='"toPerform"'>
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="#Operation"/>
  </owl:ObjectProperty>
</owl:Class>
```

### 4.3.2. Policy specification

**Definition 16** (SAAC policy specification). A SAAC Policy captures the who/what/when dimensions which can be read as follows: a SAACPolicy specifies that a user (who is playing a role) has AccessDecision (“Granted” or “Denied”) to Service if a Situation is satisfied.

**Example 5.** Let us consider an access control policy for the emergency doctors (Policy1 in the application scenario of Section 2): a user by playing the emergency doctor (ED) role can access (Read/Write operation) a patient’s emergency medical records (EMR) in the hospital for an emergency treatment (ET) purpose, when the patient is in a critical health condition.

In this policy, the access decision is based on the following policy constraints: who the user is (user’s role), what resource is being requested (resource’s identity), and when the user sends the request (the location of the user, the health status of the patient, and the purpose or user intention in accessing the resource). The SAAC policy for the emergency doctors in OWL is shown in Table 10. The core policy concepts are specified in Line 1 to 6, the role specification (emergency doctor) is shown in Line 7 to 9, the service specification (emergency medical records on read/write operation) is shown in Line 10 to 21, and the access decision (granted decision) is specified in Line 22 to 24. An atomic situation $s_{a1}$ (emergency doctors in the hospital for an emergency treatment purpose when the patient’s health condition is critical) is specified in Line 25 to 27. The specification of this atomic situation ($s_{a1}$EDFromHospitalForET) is discussed in the previous section (the ‘purpose’ and ‘situation’ regarding this policy are specified in Examples 1 and 2).

The above policy (Table 10) in a more readable form (pseudo-code) is shown in Table 11.

**Example 6.** Consider the policy presented in Section 2 (Policy2): a user by playing the registered nurse (RN) role can access a patient’s daily medical records (DMR) during her ward duty time (DT) from the general ward (GW) of the hospital where the patient is located, for a daily operational (DO) purpose and when the patient health condition is normal.
In this policy, the access decision is based on the following policy constraints: who the user is (user’s role), what resource is being requested (resource’s identity), and when the user sends the request (the co-location relationship between user and resource owner, the health status of the patient, the locations of the user and patient, the request time of the user, and the purpose or user intention in accessing the resource). The SAAC policy for the registered nurses in OWL is shown in Table 12. The policy states that the registered nurses can access the patient’s daily medical records during her ward shift time for a daily operational purpose if they both are co-located. The specification of the composite situation associated with this policy, named $s_{RN\text{FromGWAtDTForDO}}$ (registered nurses during ward duty time from the general ward of the hospital for a daily operational purpose when the patient’s health condition is normal), is shown in Table 9 in Section 4.2.4. This policy in a more readable form (pseudo-code) is shown in Table 13.
oriented situation information. For example, in the above-mentioned example (see Example 6), Mary can access a patient Bob’s daily medical records by playing the registered nurse role and the relevant conditions are satisfied. We can observe that if Mary is located in the general ward during her ward duty time, she is authorized to access the private medical records of patient Bob, who is hosted in that ward in his normal health condition. Conversely, when the purpose-oriented situation changes, she is not allowed to access such records. That is, when Mary leaves the ward or before the beginning of her ward duty time or after the end of her duty time, or for a purpose other than daily operation (e.g., treatment purpose), she is not allowed to access the medical records of patient Bob.

### 4.3.3. Policy evaluation

The SAAC policies specified in the policy ontology should be enforced to ensure the appropriate use of software services or information resources.

During the evaluation phase, an access query in the query language SQWRL [36] is used to check the user’s service access request and determine the access permission on a requested service. The access query is formulated based on the access control policies, by capturing the policy constraints and situations that are currently in effect.

**Definition 17** (Access request, AR). An access request is defined as a tuple (pass, requested service), where pass is the user’s access pass (identity, password), and requested service is a (resource, operation) pair.

Algorithm 1 presents the SAACDecision algorithm for evaluating situation-aware access control policies and making dynamic access control decisions, based on the Access Request (AR), Situation Ontology (SO) and Policy Ontology (PO).

**Algorithm 1. SAACDecision – make situation-aware access control decision.**

**Input:** AR, SO, PO  
**Output:** decision

1: procedure IDENTIFYSituation
2: situation ← identifySituation (AR, SO)
3: end procedure
4: procedure SELECTPolicy
5: policy ← selectPolicy (AR, PO)
6: end procedure
7: procedure MAKEDecision
8: decision ← makeDecision (situation, policy)
9: end procedure

Our algorithm identifies which situations are applicable in a particular scenario, selects corresponding access control policies, and makes the situation-aware access control decision. It consists of the following three procedures: identifySituation, selectPolicy and makeDecision.

Firstly, the procedure identifySituation identifies the relevant situations in the situation ontology, SO, based on the user’s access request, AR (Lines 1–3). This is done by using our implemented Situation Manager. It first captures the low-level context information from the data sources, then derives the high-level context information, and finally infers the relevant situations. We have implemented a number of context providers, as part of the Situation Manager, which receive raw context data from the data sources and generate the low-level context information. We have also implemented the situation inference engine, as part of the Situation Manager, which uses the user-defined reasoning rules and derives the high-level context information from the other context information. The inference engine is also responsible to infer the purpose-oriented situations using the available context information and the situation inference rules from the situation ontology.

Secondly, the procedure selectPolicy identifies the relevant access control policies in the policy ontology, PO (Lines 4–6). This is done by using our implemented Policy Manager. We have implemented the policy decision point, as part of the Policy Manager, which processes the user’s access request and identifies the situation-aware access control policies according to that request.

Finally, the procedure makeDecision makes the situation-aware access control decision (Lines 7–9), based on the identified situations and policies from the previous two stages. Our implemented SAAC Decision Engine is responsible for the evaluation of the access request. Depending on whether the applicable situations and policies are satisfied, the SAAC Decision Engine determines whether the access request is “granted” or “denied”.

The implementation of the Situation Manager, Policy Manager, SAAC Decision Engine, and their sub-components and functionalities are discussed in the “Prototype Implementation” section (see Section 5.1). More specific examples can be found later in the “Case Studies” section (see Section 6).

Overall, the situation ontology captures the relevant situations and the policy ontology captures the applicable SAAC policies. Access queries are formulated to check the user’s request to access the service using both the policy constraints and the applicable situations. For the application example (Scene 1 in the motivating scenario presented

---

**Table 14**

An access query.

| SAACPolicy(policy) ∧ User(name) ∧ Role(role) ∧ Situation(situation) ∧ Service(service) ∧ AccessDecision(decision) → sqwrl:select(User, role, situation, service, decision) |

**Table 15**

Access query results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>ED</td>
<td>EDFromHospitalForET</td>
<td>EMR_Write</td>
<td>Granted</td>
</tr>
<tr>
<td>Jane</td>
<td>ED</td>
<td>EDFromHospitalForET</td>
<td>EMR_Read</td>
<td>Granted</td>
</tr>
</tbody>
</table>
in Section 2), the defined access query (see an access query in Table 14) is used to determine whether the access is Granted or Denied. The access query results in Table 15 show that Jane's service access request (“EMR_Read or EMR_Write”) is granted.

5. Prototype implementation of the framework

To alleviate the complexity of building situation-aware access control applications, we in this section present a prototype implementation of the PO-SAAC framework. We also present a situation-aware access control application in the healthcare domain, to illustrate the use and practical applicability of our framework.

5.1. Prototype architecture

We have developed our framework prototype in Java2 SE [37] using widely supported tools and it has been deployed on an Intel(R) Core(TM) Duo T2450 at 2.00 GHz processor computer with 1 GB of memory running Windows XP Professional OS. The main advantage of Java is its support for interoperability between heterogeneous platforms [38]. We have used the Protégé-OWL API [39] to implement the core and healthcare ontologies. A software architecture of our prototype framework is shown in Fig. 5. The prototype implementation has a set of Software Components, which support the software engineers to develop Situation-Aware Access Control (SAAC) applications using this framework.

Our prototype framework has the following main components, Situation Manager, Policy Manager, and SAAC Decision Engine. They form the situation middleware layer (see Fig. 5). This middleware layer captures raw context data from various data sources in the sensor layer (note that, we have used the context sources as the sensors). It also infers the relevant situations and supports the building of situation-aware access control applications, such as the patient medical record management (PMRM) system, in the application layer. The functional components related to the sensor and application layers are application-specific, which are outside our research scope. In this paper, our main focus is the SAAC Middleware Layer and its associated components.

5.1.1. Situation manager

The Situation Manager consists of the following components: Context Providers and Situation Inference Engine. We

![Fig. 5. The overall architecture of the PO-SAAC framework.](image-url)
have implemented a number of Context Providers in Java which extract low-level context information by processing the raw context data from the data sources (sensors). In order to execute user-defined rules for situation reasoning, we have implemented the Situation Inference Engine. The Situation Inference Engine infers the purpose-oriented situation information using the available context information from the Situation Ontology and user-defined OWL/SWRL inference rules. We have used both the Jess rule engine [40] and Java to implement the Situation Manager. The Situation Manager is also responsible for allowing application developers to add, edit and delete user-defined reasoning rules for situation inference.

We have used Protégé-OWL API [39] to implement the Situation Ontology for representing and capturing the general and domain-specific concepts, and the relationships among these concepts. We have used the Protégé-OWL API [39] to codify the user-defined OWL rules. We have also used the SWRL rule language [35] and integrated the SWRLTab plug-in to Protégé-OWL, in order to codify the user-defined SWRL rules. The SWRL rules can directly use OWL knowledge from the ontology. The SWRLTab editor in Protégé-OWL checks syntax and provides built-in functions for mathematical computations (e.g., add, subtract, power), comparisons (e.g., equal, not equal, greater than, less than or equal), etc. [42]. In the previous section (Section 4.2), we have already presented the Situation Ontology for representing and inferring the dynamic contextual information and purpose-oriented situation information.

5.1.2. Policy manager

We have implemented the Policy Decision Point as part of the Policy Manager. The Policy Decision Point is responsible to select applicable situation-aware access control policies. It is also used to allow the SAAC application developers to add, edit and delete situation-aware access control policies, based on the Policy Ontology.

Similar to the situation ontology, we have used the Protégé-OWL API [39] to implement the Policy Ontology for specifying the situation-aware access control policies. In Section 4.3, we have presented the Policy Ontology for situation-aware access control policies.

5.1.3. SAAC decision engine

The SAAC Decision Engine is responsible for the evaluation of access requests. We have implemented the Policy Enforcement Point as part of the SAAC Decision Engine. Upon receiving an access request from the user, the Policy Enforcement Point forwards it to the Policy Manager. The Policy Manager checks the user’s access request and the relevant situation-aware access control policies in the Policy Ontology. At the same time, the Situation Manager identifies the relevant situations in the Situation Ontology. Then, depending on whether or not the conditions of the applicable access control policies are satisfied by the identified situations, the Policy Enforcement Point determines whether the access request is “Granted” or “Denied”.

If the decision is granted, i.e., if access to the requested service is authorized, the Policy Enforcement Point returns the requested resource to the user, otherwise, a denied response is sent to the user. Specifically, in order to check who can access what resources at runtime, we have used the SQWRL [36] as a query language. To codify the access queries, we have integrated the SQWRLQueryTab [43] plug-in to the Protégé-OWL (see an access query in Table 14).

5.2. Developing a SAAC application for healthcare

Using the PO-SAAC prototype implementation, we have developed a situation-aware application from the healthcare domain to illustrate the operation of our purpose-oriented situation-aware access control approach.

The environment of our application is the Patient Medical Records Management (PMRM) system. It demonstrates the practical applicability of our framework. This PMRM application is an example of how to realize situation-aware access control decisions. The main goal of this application is to provide access control for the different medical records of patients based on the relevant situations. The application allows different users (e.g., doctors and nurses [29]) to invoke different operations to access specific patient medical records in a situation-aware manner.

We use Java programs and relational databases to simulate the different context sources. For example, our prototype application has context sources: SystemTime (which provides current_time), Location (which provides location_address), User_Pass relational table (containing usr_id and password), User_Role table (containing usr_id and usr_role_id), Patient_Profile table (containing patient_id, patient_name and connected_people_id), patient Health_Profile (which provides patient_id, heart_rate and body_temperature), etc. Further details of this PMRM application can be found in the case studies of the next section (Section 6).

Overall, our framework prototype provides efficient infrastructure support for building situation-aware access control applications in dynamic and context-aware environments (see Section 7 below for its performance evaluation).

6. Case studies

For access requests, our PO-SAAC framework provides the following modelling features for dynamic situation-aware access control decision making.

- **Situation**: our framework models purpose-oriented situations using its situation model.
  - Modelling and identifying the states of the relevant context entities and the states of the relevant relationships between entities from the currently available contextual information.
  - Modelling and inferring the purpose or user’s intention in accessing the services.
  - Reasoning about situations using the inferred purpose and other relevant states.
- **Policy**: our framework models the situation-aware access control policies using its policy model.
6.1. Test case 1: Healthcare domain

Consider the application scenario from Section 2 (Scene 1), where Jane, by playing an emergency doctor role, wants to access the services writeEMR() and readEMR() the emergency medical records of patient Bob. The service AccessRequests are submitted to the SAAC Decision Engine for evaluation. Jane’s service AccessRequests are shown as follows:

\[
\begin{align*}
\text{<pass = (Jane, \text{****}), service = (EMR(Bob), Write)>}
\text{<pass = (Jane, \text{****}), service = (EMR(Bob), Read)>}
\end{align*}
\]

The SAAC Decision Engine forwards the requests to the Situation Manager, which checks the relevant situation information in the situation ontology (introduced in Section 4.2). It also requests the applicable access control policies in the policy ontology (introduced in Section 4.3) through the Policy Manager.

The bottom layer of our situation ontology (see Fig. 2) captures the relevant context information (interpersonal relationship, location, health status, etc.). The top layer of the ontology captures the relevant situations based on the captured context information and situation specification rules.

By using the core situation concepts, for the patient’s medical records management (PMRM) application (see Section 2), we specify different atomic situations based on the data/information from the healthcare domain. Some of these ‘situations’ and their associated ‘context information’ and ‘purpose’ using situation reasoning rules are shown in Table 16.

In order to identify the relevant situations, our situation ontology first identifies the user’s intention or purpose of resource/service access based on the relevant context information. Table 4 in Section 4.2 identified that the purpose of Jane’s requests are for Emergency Treatment (ET), based on the relevant contextual information (role identity, health status, etc.).

The ontology identifies the relevant states (the states of the relevant entities and the states of the relevant relationships between entities) using currently applicable context information from the bottom layer of our situation ontology. These states and the inferred purpose (above) are used to infer the situation defined in OWL in Table 6 (see the ‘Situation Ontology’ in Section 4.2).

Our policy ontology specifies the applicable situation-aware access control policies. A relevant policy in OWL is shown in Table 10 (see the ‘Policy Ontology’ in Section 4.3).

The applicable situation information (identified in the situation ontology) and the applicable policy (specified in the policy) captured at the time of access request are provided to the SAAC Decision Engine as part of the request processing. Based on this information, the SAAC Decision Engine determines whether the request is ‘Granted’ or ‘Denied’ for the submitted access request, and returns the decision to the user.

For the above-mentioned application scenario, Jane’s service access request is Granted (see an access query in Table 14 and the query results in Table 15), because the situation ontology captures relevant situation that satisfies a SAAC policy.

When the situation changes (i.e., Jane leaves the hospital or for a purpose other than emergency treatment), she is not allowed to access the emergency medical records of patient Bob.

6.2. Test case 2: University domain

In this section, we demonstrate the use of our PO-SAAC approach through another application from the university domain. The application allows different university users (e.g., professor, lecturer, student) to access appropriate student evaluation records (e.g., discussion reports, assignments) in a situation-aware manner. Though this second case study deals with issues of similar nature to the first case study, the main purpose of including the second case study from the university domain is to demonstrate the general applicability of our approach to a different domain.

<table>
<thead>
<tr>
<th>Situation name</th>
<th>Situation definition (high-level description)</th>
</tr>
</thead>
<tbody>
<tr>
<td>An emergency doctor from the hospital for an emergency treatment (EDFromHospitalForET)</td>
<td>User_Role(ED) ∧ Location_ED(Hospital) ∧ Purpose(ET)</td>
</tr>
<tr>
<td>A general practitioner (who is treating physician of a patient) from the emergency room for an emergency treatment (GPFromERForET)</td>
<td>User_Role(GP) ∧ Location_GP(EmergencyRoom) ∧ Interpersonal_Relationship(TreatingDoctor) ∧ Purpose(ET)</td>
</tr>
<tr>
<td>A general practitioner from the hospital for normal treatment (GPFromHospitalForNT)</td>
<td>User_Role(GP) ∧ Location_GP(Hospital) ∧ Purpose(NOT)</td>
</tr>
<tr>
<td>A registered nurse from the general ward (where the patient is present) for a daily operation (RNFromGWForDO)</td>
<td>User_Role(RN) ∧ Location_RN(GeneralWard) ∧ Colocated_Relationship(Yes) ∧ Purpose(DO)</td>
</tr>
<tr>
<td>A registered nurse at ward duty time for a daily operation (RNAtDTForDO)</td>
<td>User_Role(RN) ∧ RequestTime_RN(DutyTime) ∧ Purpose(DO)</td>
</tr>
<tr>
<td>A guest researcher from the hospital for research (GRFromHospitalForR)</td>
<td>User_Role(GR) ∧ Location_GR(Hospital) ∧ Purpose(R)</td>
</tr>
</tbody>
</table>

Table 16
Definition of different situations.
Consider a scenario, where a professor Richard is allowed to access the service writeDP(Tom) (i.e., the write access to the weekly discussion report (DR) of his PhD student Tom) for research purpose. Normally, all professors are allowed to access all the reports of his group’s students in any location, however, a student only can see his report in the university.

Richard’s service AccessRequest is shown as follows:

\[
\text{<pass} = (\text{Richard, *****, service} = (\text{DP(Tom), Write})>\]

A relevant ‘situation’ and its associated ‘context information’ and ‘purpose’ using situation reasoning rules are given as follows. Table 17 identifies the purpose is Research, based on the relevant context information (role identity, etc.).

The situation definition in Table 18 identifies the relevant situation based on the purpose and available context information (e.g., interpersonal relationship).

The relevant policy (pseudo-code) is shown in Table 19.

Based on the relevant situation information (identified in the situation ontology) and the applicable policy (specified in the policy ontology), our framework returns the situation-aware access control decision, i.e., Richard’s service access request is Granted. Table 20 shows an access query that is used to determine whether the access is Granted or Denied. The access query result in Table 21 shows that Richard’s service access request ("DP_Write") is granted.

In the above-mentioned application scenario, only the professor Richard can access his PhD student Tom’s discussion report. Other professors are not allowed to access the discussion report of Tom and Richard. Because, in this case, the context/situation changes (i.e., the interpersonal relationship is not a supervisor or for a purpose other than research).
In general, when a new access request comes or the situation changes, the SAAC Decision Engine sends automated request to the Situation Manager for the relevant situations and to the Policy Manager for the applicable policies. Then it (re-)evaluates the situation-aware access control decisions according to the new information.

Overall, the above two case study scenarios have explored the use of our PO-SAAC framework for the development of the situation-aware access control applications. It yields different access control decisions in a situation-aware manner, and shows general applicability of our PO-SAAC framework.

7. Performance evaluation

In this section, we measure the query response time of our framework implementation in providing resource/service access permissions to users.

7.1. Experimental setting

With the goal of evaluating the runtime performance of the various stages of our prototype framework, we have conducted three sets of experiments on two different machines:

1. Windows XP professional operating system running on Intel(R) Core(TM) Duo T2450 at 2.00 GHz processor with 1 GB of memory (experimental setting 1).
2. Windows 7 operating system running on Intel(R) Core(TM) 2 Duo E8400 at 3.00 GHz processor with 2 GB of memory (experimental setting 2).

7.2. Measurement

The main purpose of this experimental investigation is to quantify the performance overhead of our approach and framework. Our main measures include:

- Situation identification time – It is the time taken to identify/infer a relevant situation (by capturing the currently available context information and identifying the purpose using this information).
- Policy evaluation time – It is the time taken to determine a user’s access permission on a requested service (by incorporating the identified situation into the access control process and making access control decision).

We calculate the average end-to-end response time, time from the arrival of the user’s service access request (query) to the end of its execution, which equals to the sum of the time for identifying relevant situation and the time for evaluating relevant policy.

7.3. Experimental results and analysis

We have conducted three sets of experiments to examine the performance of PO-SAAC.

7.3.1. Test 1

This first set of experiments were carried out on the first machine setup mentioned in Section 7.1 and measured the response time of our prototype implementation against increasing number of policies.

First, we have selected 20 policy rules in which 5 situation types (ST) act as the policy constraints (e.g., the situation-aware policy rule for emergency doctor for an emergency treatment purpose as shown in Table 5). We have varied the number of policies up to 100 with 15 different types of situation variations. Each of these variations is executed 10 times for each of following cases: 5 ST (situation types), 10 ST, and 15 ST. For each setting, the average value of the 10 execution runs is used for the analysis.

The test results in Fig. 6 show that the average response time increases when the number of situation types and policies increases. For example, it varies between 4.1 and 5.2 s for different number of situation types and for the variation of 20–100 policy rules. We can see that the average response time seems to be linear.

7.3.2. Test 2

In the second test, we have again evaluated the total response time on our first machine setup over various size of the knowledge base.

We have varied the number of policies up to 500 with respect to 138 different types of health professionals [29] (i.e., 138 roles). To build the ontology knowledge base of increasing sizes, we have specified 2000 policies. In order to measure the response time, we have run each experiment 10 times and the average value of the 10 execution runs is used for the analysis.

The test results in Fig. 7 show that the average response time increases when the number of situation types and policies increases. For example, it varies between 4.1 and 5.2 s for different number of situation types and for the variation of 20–100 policy rules. We can see that the average response time seems to be linear.
runtime performance is acceptable for a reasonable sized knowledge base.

7.3.3. Test 3

We have conducted another experiment on a second machine setup mentioned in Section 7.1 and again evaluated the query response time, to examine the impact of the machine capacity.

Similar to test 2, we have varied the number of policies up to 2000 with respect to 138 different health professionals. The average value of the 10 execution runs is used for the analysis.

The test results in Fig. 8 show the runtime performance on the different machines. The query response time seems to be linear with small ontology KB (1618 kilobytes). When the ontology KB beyond 1618 kilobytes, the response time behaves non-linear, due to the limitation of the ontology reasoner. Overall, we can see that the computational overhead decreases using a more powerful machine, indicating the impact of the computing resources (CPU and memory) on the performance.

8. Related work

In this section, we examine the existing access control research works that are related to our research. We classify these works into six groups: (i) context-aware role-based access control approaches, (ii) organization-based access control approaches, (iii) situation-aware access control approaches, (iv) attribute-based access control approaches, (v) purpose-based access control approaches, and (vi) situation modelling approaches.

8.1. Context-aware role-based access control

The Role-Based Access Control (RBAC) concept emerged in the early 90s for managing and enforcing security in large-scale domains. The traditional RBAC approach evaluates access permissions depending on the user’s roles in an organization and it ensures that only an authorized user is given access to a certain resource or service [7].

Several access control works (e.g., [11–14,44,45]) have extended the basic RBAC approach and incorporated location and time as context information in the RBAC access control policies. These approaches do not consider the purpose-oriented situation information as we do in our approach.

Recently, some other role-based access control approaches have also adopted and extended the basic RBAC approach (e.g., [15–18]). They consider some further contextual information other than the temporal and spatial dimensions, such as the user and resource dimensions. These approaches are still limited in identifying the situation information.

In general, the existing RBAC and context-aware RBAC approaches do not consider situations or provide functionalities to incorporate situation information in the access control process. In this paper, we have adopted and extended the basic RBAC concepts, and incorporated purpose-oriented situation information in the access control policies for dynamic access control decision making.

8.2. Organization-based access control

The above-mentioned RBAC approaches address role-based access control decision making in the context of a single organization. The organization-based access control approaches have been designed to allow access control management policies to be applied in a multi-organizational setting (e.g., [46–48]).

Kalam et al. [46] and Cuppens et al. [47] have proposed the basic Organization-Based Access Control (OrBAC) approach. OrBAC tries to overcome the limitations of traditional RBAC approach by considering the concept of organization together with the concept of context and it helps to model the contextual security policies of the organizations.

Preda et al. [48] have presented an improvement over the basic OrBAC approach for dynamically deploying security policies and proposed the context-aware access control policies in terms of resource access and availability, and also in terms of system vulnerabilities and treats.

The core and extended OrBAC approaches provide different types of contexts such as temporal, spatial, provisional, prerequisite and user-declared information as policy constraints. However, they are still limited in considering the context information characterizing the relationships between entities, context inference and situation information, and the use of this information in access control...
policies. In contrast, our approach considers this information and its use in formulating access control policies.

8.3. Situation-aware access control

The above-mentioned access control approaches directly incorporate contextual information in the access control policies for access control decision making. On the other hand, situation-aware access control approaches control resource/service access on the basis of situation information (i.e., specific combinations of contextual information).

Kim and Lim [27] have proposed the Situation-Aware Role-Based Access Control (SA-RBAC) approach, which extends the basic RBAC approach [7] and dynamically grants roles (or permissions) to users based on the situation information of the user. The situations in the SA-RBAC approach combine the required credentials of users, and the context information such as location, time, and system resources relevant to the user’s access request.

Yau and others [49] have defined situation as a set of context attributes of users, systems and environments over a period of time affecting future system behavior. Later, Yau and Liu have presented a Situation-Aware Access Control (SA-AC) based privacy-preserving service matchmaking approach [28]. The SA-AC approach incorporates situation-aware constraints into the RBAC approach, such that the states of service providers, requesters and environments can be taken into account when making the access control decisions.

Garcia-Morchon and Wehrle [50] have proposed a two-layer modular context-aware access control approach for medical sensor networks comprising a data layer and an engine layer. The data layer comprises all the information (including context information such as location and time) that is required for access control decisions. The engine layer manages the access control decisions in critical, emergency, and normal situations using this context information and access control policies. This access control approach adopts the basic RBAC approach [7] and incorporates the situation information into access control process.

These situation-aware access control approaches only consider the states of the relevant entities as policy constraints. In dynamic and context-aware environments, however, the states of the relevant relationships between entities are also important consideration in access control decision making. In our PO-SAAC approach, a situation not only involves the states of the specific types of context entities but also the states of the relevant relationships between entities. Moreover, our approach also considers the purpose or user’s intention in accessing the services in modelling situations.

8.4. Attribute-based access control

A number of further research works (e.g., [19–22]) have adopted and extended the basic Attribute-Based Access Control (ABAC) approach to provide access control to software services in a context-aware manner.

A context-aware access control (CAAC) approach for ubiquitous computing environments has been proposed by Corradi et al. [19], where service access permissions are given to users based on the context information: user location, user activities, user device, time, resource availability and resource status. Hulsebosch et al. [20] have proposed a context-sensitive access control (CSAC) framework based on the user’s location and access history. Toninelli et al. [21] have proposed a semantic CAAC approach which provides resource access permission on the context aspects of resource availability, user identities, location and time. These approaches also have limitations in considering a limited set of contexts.

A recent attribute-based 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 [22]. Its consideration of context is still limited to the user, device and environment dimensions.

These basic and extended ABAC approaches do not have the capability of inferring purpose-oriented situations as we have in our approach.

8.5. Purpose-based access control

Purpose-based access control has also received considerable attention in the context-aware computing domain.

Byun and Li [2] have proposed a privacy preserving access control model for relational databases where the purpose information associated with a given data element specifies the intended use of the data element. Their access control policy normally states that the particular resources can be accessed only for the specific purpose; and a purpose describes the reason for data access and data collection.

Sun et al. [51] have presented a purpose-based access control model (usage access control and purpose extension) for medical information system, where ‘usage’ means usage of rights on digital objects, and ‘purpose’ dictates how access to data items should be controlled.

A major difference of our PO-SAAC approach with respect to these purpose-based access control approaches is that we not only consider the purpose information but also consider the different granularity levels of the purpose information. In addition, different from these approaches, our approach can dynamically identify the appropriate purpose or user’s intention in accessing the requested services based on the currently available context information.

8.6. Situation modelling approach

Several situation modelling approaches have been proposed in the literature for modelling information in situation-aware computing environments.

The context ontology [24], situation theory ontology [25] and situation-awareness ontology [26] describe “situation” as the states of the specific kind of context entities (e.g., attributes of users or other relevant entities). These approaches provide valuable insights into developing a fine-grained ontology-based situation-aware access control framework. However, these approaches are highly domain-specific and they do not consider several concepts that are important to situation modelling
8.7. Comparative assessment of our approach and the related access control approaches

In the previous sections, we have highlighted the strengths and limitations of the various access control approaches ranging from the traditional forms of access control approaches to application-specific approaches. In this section, we conduct a comparative assessment of our PO-SAAC approach and the related access control approaches. We first present the key features of our approach and then we analyze the existing access control approaches in terms of these features.

8.7.1. Comparative analysis

Our purpose-oriented situation-aware access control approach provides the following key features.

- **Purpose-oriented situation information**: Our PO-SAAC approach provides a Purpose-Oriented Situation Model (PO-SM) to identify relevant atomic situations and reason about composite situations. The situation model uses the available contextual information in order to identify the elementary information.
  - Identifying and inferring the states of the context entities (ES).
  - Identifying and inferring the states of the relationships between entities (RS).
  - Identifying and inferring the purpose or user’s intention in accessing the resources (UI).
  - Modelling the basic atomic situations, combining the ES, RS, and UI (AS).
  - Reasoning about composite situations based on one or more atomic situations (CS).

- **Dynamic access control decision making**: Our PO-SAAC approach provides a Situation-Aware Policy Model (SA-PM) for dynamic access control decision making.
  - Specification of purpose-oriented situation-aware access control policies for dynamic access control decision making, by incorporating the situation information in the access control policies.

We compare our approach with existing access control approaches based on the above-identified key features. In general, we distinguish two categories of access control approaches for this analysis. Firstly, the access control approaches that incorporate different types of contextual information into role-based access control. Secondly, the access control approaches that integrate situation information into access control processes.

Table 22 shows a comparative analysis of how the existing access control approaches and our approach support the identified key features. We use a √ if a related approach provides similar and/or comparable support for a certain aspect, a ▵ if a related approach provides partial support for a particular aspect, and a △ if a related approach provides no support for a particular aspect.

Several existing access control approaches have extended the basic role-based access control approach (RBAC) [7] by incorporating some specific types of contextual information into access control processes: temporal information (e.g., [11,12]), spatial information (e.g., [13]), and both time and location (e.g., [14]). A number of recent approaches (e.g., [15,16,18]) have also adopted the RBAC approach for context-specific access control. These approaches usually focus on the integration of several further types of contextual information other than location and time, such as user attributes. Different from these approaches, our PO-SAAC approach considers specific combinations of available contextual information for identifying the relevant situation information. Moreover, these existing context-aware RBAC approaches incorporate the
context information to specify the context-aware access control policies. Different from them, our approach considers the situation information for the specification of access control policies.

The existing situation-aware access control approaches describe situation as the specific combinations of certain contextual information (e.g., [26]). Kim and Lim propose the situation-aware RBAC (SA-RBAC) approach, which extends the basic RBAC approach and dynamically grants (resource access) permissions to users based on the situation information of the user [27]. Yau and Liu have presented a situation-aware access control (SA-AC) based privacy-preserving service matchmaking approach [28]. The SA-AC approach incorporates situation-aware constraints into RBAC approach that can affect the access control decisions, such that the states of service providers, requesters and environments. In order to specify situation, these approaches consider the states of the relevant entities. Different from these approaches, our PO-SAAC approach [1] also considers the states of the relevant relationships between entities for situation specification. None of these approaches consider the purpose or user’s intention in accessing resources and services, whereas our approach considers this.

8.7.2. Overall discussion

The existing context-aware access control approaches have only considered specific types of contextual information. On the other hand, the situation-aware access control approaches consider specific combinations of certain contextual information (e.g., the states of the relevant entities) as situations. However, they do not consider the states of the relevant relationships between context entities or the purpose or user’s intention in accessing resources and services. Furthermore, access control that considers dynamic situation information with reasoning capabilities is largely missing in the literature. In dynamic and context-aware environments, these concepts are also important considerations in access control decision making. Therefore, our purpose-oriented situation model not only identifies the relevant atomic situations, but also includes a reasoning technique to infer new composite situations from one or more atomic/composite situations using user-defined rules.

The existing policy models associated with access control approaches incorporate specific types of contextual information or combinations of contextual information (situations) in the access control processes. However, these policy models do not have adequate functionalities to incorporate purpose-oriented situation information into access control policies for dynamic access control decision making. Therefore, our situation-aware policy model for dynamic access control decision making incorporates purpose-oriented situation information into access control policies.

9. Conclusion and future work

In this paper, we have presented a new purpose-oriented situation-aware access control (PO-SAAC) framework for software services. One of the main contributions of this paper is the PO-SAAC model for specifying the purpose-oriented situations and the corresponding situation-specific access control policies. It introduces a formalization of the basic concepts of the situation and policy models for situation-aware access control. Another contribution of this paper is an ontology-based development platform, for realizing the PO-SAAC model with semantic technologies OWL and SWRL. It includes a situation ontology specific to access control for specifying the relevant situations (atomic situations), and a reasoning technique for inferring composite situations based on user-defined reasoning rules. The situation ontology can capture the relevant elementary information from the environments (the relevant states and the user’s intention or purpose) in order to identify situations. Using our PO-SAAC framework, users can have different access permissions to information resources and software services depending upon the purpose-oriented situations. By introducing the concept of purpose-oriented situations, the purpose-oriented control over access to information resources and software services has been achieved.

We have developed a software prototype of the PO-SAAC framework and a situation-aware access control application in the healthcare domain using this framework. We have also included an additional case study from the university domain. The case studies show that our framework captures relevant situations at runtime and is capable of managing software service invocations in a situation-aware manner. We have performed a range of experiments and the experimental results have shown that our framework provides satisfactory performance. In addition to the case studies and runtime performance evaluation, we have compared our framework with existing context/situation-aware access control frameworks. The comparative analysis has shown that our framework offers a number of new capabilities for situation-aware access control. Our PO-SAAC framework is able to provide effective support for software engineers in building situation-aware access control applications.

In our PO-SAAC approach, the mapping between users and roles is a many-to-many relationship, i.e., one user can be assigned many roles and many users can be assigned to one role. Consequently, such issues as role conflict may arise. As future work we will investigate context-aware user-role assignment and its related management issues. In addition, we will consider the consistency and completeness of the rule-based specification as future work. We will also focus on the scalability of our approach and framework in the mobile computing setting. In addition, we plan to apply our framework to the real world and further examine and optimize its performance as future work.

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.