PO-SAAC: A Purpose-Oriented Situation-Aware Access Control Framework for Software Services

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

Swinburne University of Technology, Victoria 3122, Australia
{akayes,jhan,acolman}@swin.edu.au

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. Existing approaches are highly domain-specific and they control access to services depending on the specific types of context information without considering the *purpose*. To achieve *situation-aware access control*, in this paper we consider *purpose-oriented situations* rather than conventional situations (e.g., user's state). We take *situation* to mean the states of the entities and the states of the relationships between entities that are relevant to the purpose of a resource access request. We propose a generic framework, *Purpose-Oriented Situation-Aware Access Control*, that supports access control to software services based on the relevant situations. We develop a software prototype to demonstrate the practical applicability of the framework. In addition, we demonstrate the effectiveness of our framework through a healthcare case study. Experimental results demonstrate the satisfactory performance of our framework.

Keywords: Situation-aware access control, Context information, Purpose, Situation reasoning, Access control policy.

1 Introduction

In open and dynamic environments, Situation-Aware Access Control (SAAC) applications need to capture and manipulate context information [1] to identify relevant situations and need to adapt their behaviors as the situation changes. In such environments, users demand access to appropriate software services in an anytime and anywhere fashion, as described by Weiser [2], with more flexibility and richer resources, and yet not to compromise the relevant privacy and security requirements of the stakeholders. A security policy (situation-aware access control policy) normally states that the particular services can be invoked based on (i) *the states of the relevant entities* and (ii) *the specific purpose*; which describes the reason for which organizational resources are used [3]. For example, an emergency doctor's request to invoke a healthcare service (access to the emergency patient's records when the patient is in a critical health condition) may be possible from the inside of the hospital but may not from the public bus.

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 58–74, 2014.

Also, such service access request can be granted for the emergency treatment purpose. 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 [4]. 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 the states of the relevant relationships between different entities (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.

The basic components to achieve situation-awareness have already been defined by Endsley [5], *"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 describe situation as the states of the specific kind of entities (e.g., [6],[7],[8]). However, other than the entity states, the states of the relevant relationships between entities are not considered in this situation-awareness research.

Some situation-aware access control approaches have been proposed in the access control literature (e.g., [9], [10]), 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 states, 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 the situation-aware access control are: the combination of the relevant *entity states* and the *relationship states*, and the *purpose or user's intention.*

The Contributions. In order to address the above-identified research issues and challenges, we present a novel framework PO-SAAC (*Purpose-Oriented Situation-Aware Access Control*, to provide situation-specific access to software services. The novel features of this framework are as follows:

(C1) *Purpose-Oriented Situation Model.* Our framework uses the *purpose-oriented situation* information to provide 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 the situation and policy models, we introduce an ontology-based platform for modeling and identifying purpose-oriented situations, 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.

(C4) *Prototype Implementation and Evaluation.* In order to demonstrate the practical applicability of our approach, we have presented a *software prototype* for the development of the situation-aware access control applications. We have carried out a healthcare case study, in order to demonstrate the effectiveness of our framework. To demonstrate the feasibility of our framework, we have conducted a number of experiments on a simulated healthcare environment. We have quantified the *performance overhead* of our framework for measuring the response time. Experimental results have demonstrated the satisfactory performance of our proposed framework.

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 the design of our PO-SAAC framework, a situation model to specify different situations and a policy model for specifying situation-specific access control policies. Section 4 presents an ontology-based development platform for our framework. Section 5 describes the prototype implementation along with the viability of the framework. Section 6 discusses related work. Finally, Section 7 concludes the paper and outlines future work.

2 Research Motivation and General Requirements

As an example of the type of situations that a situation-specific access control framework has to consider, in this section we outline a motivating scenario that illustrates the need for the incorporation of purpose-oriented situation information in access control policies. We then distil the general requirements for managing the access to software services in a situation-aware manner.

Motivating Scenario. To exemplify the complexity of achieving situation-awareness in access control systems, we reflect on the area of patient medical records management 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. Concerning this scene, one of the relevant situation-aware access control policy is shown in Table 1 (see *Policy #1*).

Scene #2: After getting emergency treatment, Bob is shifted from the emergency department to the general ward of the hospital and has been assigned a registered nurse Mary, who has regular follow-up visits to monitor his health condition. Mary needs to access Bob's daily medical records from the general ward with certain conditions (see the corresponding Policy #2 in Table 1).

Concerning the above scenario and their related policies, we can see that a set of constraints include: the user role (e.g., emergency doctor, registered nurse), the

Table 1. Example Access Control Policies

No	Policy
Policy #1	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 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).
Policy #2	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 daily operational purpose.

relevant environmental information (e.g., the emergency room, the interpersonal relationship between doctor and patient), the 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); and these policies refer to need to be evaluated in conjunction with these relevant information.

General Requirements. To support the situation-aware access control in a computer application like the patients' medical record 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 elementary information are integrated into the access control processes, some important issues arise. These issues and their related requirements are as follows:

(R1) *Representation of purpose-oriented situations:* What access control-specific elementary information should be identified as part of building a purpose-oriented situation model specific to SAAC? Furthermore, how to represent and reason about the different types of situations?
(R2) *Specification of situation-aware access control policies:* How to define the access control policies based on the relevant situations to realize a flexible and dynamic access control scheme?
(R3) *Implementation framework:* How to realize the relevant situations and the corresponding situation-specific access control policies in an effective way, in oder to access/manage software services?

3 Purpose-Oriented Situation-Aware Access Control

In this section, we present our Purpose-Oriented Situation-Aware Access Control (PO-SAAC) framework for software services.

3.1 Purpose-Oriented Situation Model

A situation consists of the set of *elementary information* (the combination of the relevant states and the user's intention or purpose-oriented information). In our

purpose-oriented situation model, we define the simple situation (*atomic situation*) and the complex situation (*composite situation*) that are used in specifying situation-specific access control policies.

3.1.1 Representation of Situation

Different atomic situations can be defined based on the data/information from the organization (domain-specific).

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 different relevant 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.).

An atomic situation 'S_a' is the logical conjunction of 'P' and 'St'.

$$S_a \quad = \quad P \quad \wedge \quad St \tag{1}$$

where 'P' denotes the purpose or user's intention in accessing the service, e.g., considering our application scenario (*Scene #1*), *purpose = "EmergencyTreatment"*; and 'St' denotes the state of the relevant entity, e.g., $location(Jane)$ = *"Hospital"*, or the state of the relevant relationships between entities, e.g., $interpersonalRelationship(Jane, Bob)$ = *"NonTreatingDoctor"*.

A purpose 'P' can be identified based on the currently available contexts (i.e., the *states* of the relevant entity and the relationships between entities).

Example 1. Consider *Policy #1* related to 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 emergency treatment (ET) purpose, when the patient is in a critical condition. The following rule (2) is used to identify that the purpose is 'ET' (i.e., a user by playing the 'ED' role can access a patient's medical records for 'ET' purpose, when the patient's health condition is critical),

$$\begin{aligned} Purpose(p) \wedge User(u) \wedge Role(r) \wedge Owner(o) \wedge \ Resource(res) \\ \wedge \ isPlayedBy(r,u) \wedge \ equal(r, \text{“}ED\text{”}) \wedge \ isOwnedBy(res,o) \\ \wedge \ healthStatus(o, \text{“}Critical\text{”}) \rightarrow equal(p, \text{“}ET\text{”}). \end{aligned} \tag{2}$$

Example 2. Consider the policy mentioned in *Example 1*, in which the relevant elementary information is represented as an atomic situation ($s_{a1} \in S_a$),

$$\begin{aligned} s_{a1} \quad = User(u) \wedge Purpose(p) \wedge intendedPurpose(u,p) \wedge equal(p, \text{“}ET\text{”}) \\ \wedge \ Location(l) \wedge \ hasLocation(u,l) \wedge equal(l, \text{“}Hospital\text{”}). \end{aligned} \tag{3}$$

3.1.2 Reasoning about Situation

The process of inferring a new composite situation (complex situation) from the one or more already defined/existing atomic situations is refereed to *reasoning*

about situation. One of the main advantages of our framework to situation-awareness is its reasoning capability; that is, once facts about the world is stated, other facts can be inferred using an inference engine through the reasoning rules.

Definition 2 (Composite Situation, S_c). Given a collection of atomic situations, the composite situations can be defined by performing logical composition (AND, OR or NOT) on the same purpose-oriented atomic situations.

Example 3. Consider *Policy #2* related to our application scenario. The daily operation (DO) purpose can be identified using the following rule (4).

$$\begin{aligned} Purpose(p) \wedge User(u) \wedge Role(r) \wedge Owner(o) \wedge\ & Resource(res) \\ \wedge\ isPlayedBy(r,u) \wedge\ equal(r, \text{“}RN\text{”}) \wedge\ & isOwnedBy(res,o) \\ \wedge\ healthStatus(o, \text{“}Normal\text{”}) \rightarrow & equal(p, \text{“}DO\text{”}). \end{aligned} \tag{4}$$

Two atomic situations regarding the mentioned policy are represented as,

$$\begin{aligned} s_{a2} \quad = User(u) \wedge Purpose(p) \wedge intendedPurpose(u,p) \wedge equal(p, \text{“}DO\text{”}) \\ \wedge\ Location(l) \wedge\ hasLocation(u,l) \wedge equal(l, \text{“}GW\text{”}). \end{aligned} \tag{5}$$

$$\begin{aligned} s_{a3} \quad = User(u) \wedge Purpose(p) \wedge intendedPurpose(u,p) \wedge equal(p, \text{“}DO\text{”}) \\ \wedge\ Time(t) \wedge\ hasRequestTime(u,t) \wedge equal(t, \text{“}DT\text{”}). \end{aligned} \tag{6}$$

A policy associated with the situation 's_{a2}' 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 daily operation (DO) purpose, when the patient's health condition is normal.

An example policy associated with the situation 's_{a3}' 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} and s_{a3}) can be identified using the following logical conjunction, $s_{c1} = s_{a2} \wedge s_{a3}$.

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 *pair <res, op>* 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 *<EMR, write>* or *writeEMR()*. In this way, the fine-grained access control to resources can be realized by managing the access to the service operations.

3.3 The SAAC Policy Model for Software Service

Based on the formalization of the RBAC model in [11], we present a formal definition of our policy model. Our policy model for SAAC applications that

extends RBAC with relevant situations, which is 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 attributes (i.e., relevant situations) as policy constraints.

Definition 3 (SAAC Policy Model). Our policy model for access control can be formally described as a tuple, where R, S, Ser, and $SAACPolicy$ represents Roles, Situations, Services, and Policies, respectively (Formula (7)):

$$M_{SAAC} = (R, S, Ser, SAACPolicy) \tag{7}$$

1. ***Roles (R):*** A set of roles $R = \{r_1, ..., r_m\}$. A role reflects user's job function or job title within the organization. A user is a human-being (who is a service requester) whose service access request is being controlled.
2. ***Situation (S):*** A set of situations $S = \{s_1, ..., s_n\} = S_a \cup S_c$ specified by using the situation model. S is used to express the relevant situations (atomic, S_a or composite, S_c) in order to describe the SAAC policies.
3. ***Services (Ser):*** A set of services $Ser = \{ser_1, ..., ser_o\} = \{(res, op) | res \in Res, op \in OP\}$, where Res is a set of component parts of resources, $Res = \{res_1, ..., res_p\}$ and OP is a set of operations on the resources, $OP = \{op_1, ..., op_q\}$. 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.
4. ***Policies (SAACPolicy):*** A set of policies $SAACPolicy = \{sp_1, ..., sp_r\} \subseteq R \times S \times Ser$. Our model has situation-aware role-service assignment policies to provide situation specific access to software services.

Our policy model extends the concept of common *role-permission assignments* (RPA) in RBAC ($RPA \subseteq R \times P$) [11], by introducing the concept of purpose-oriented situation, called *situation-aware role-service assignments*.

Example 4: Based on our policy model (*Role(r)* $\wedge$ *Situation(s)* $\wedge$ *Service(ser)* $\rightarrow$ *(r, s, ser)* $\in$ *SAACPolicy*), the following rule (shown in Table 2) expresses the policy mentioned 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 denotes s_{a1} mentioned in Example 2) is satisfied.

Table 2. An Example Situation-Aware Access Control Policy

If
$SAACPolicy(sp_1) \wedge Role(r) \wedge equal(r, \text{“}ED\text{”}) \wedge hasRole(sp_1, r) \wedge$
$Service(ser) \wedge equal(ser, \text{“}writeEMR()\text{”}) \wedge hasService(sp_1, ser) \wedge$
$Situation(s) \wedge equal(s, \text{“}s_{a1}\text{”}) \wedge hasSituation(sp_1, s)$
Then
$canInvoke(u, ser)$

The identification of the relevant information to represent the *purpose-oriented situations* and specify the corresponding *SAAC policies* satisfies requirements R(1)

and R(2), which is discussed earlier. To meet requirement R(3), we in the next section propose an *ontology-based development platform*.

4 Ontology-Based PO-SAAC Framework

We have introduced an ontology-based PO-SAAC framework to model relevant purpose-oriented situations and situation-specific access control policies. The principal goal of our framework is to formalize the situation-aware access control concepts using a logic-based language. To achieve this goal, we have already identified relevant concepts in the previous section.

In the literature, there are many languages that have been developed for specifying computer-processable semantics. In the present age, ontology-based modeling technique has been proven as a suitable logical language for modeling dynamic contexts/situations (e.g., [12], [13]). The ontology-based modeling approach to achieve situation-awareness (e.g., [7], [8]) is not only beneficial from the representational viewpoint but also beneficial from the reasoning viewpoint; that is, once facts about the world is stated in terms of the ontology, other facts can be inferred using the inference engine through the inference rules.

To model the PO-SAAC ontology, in this paper, we adopt the OWL language 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 computational complexity of reasoning and expressiveness [13].

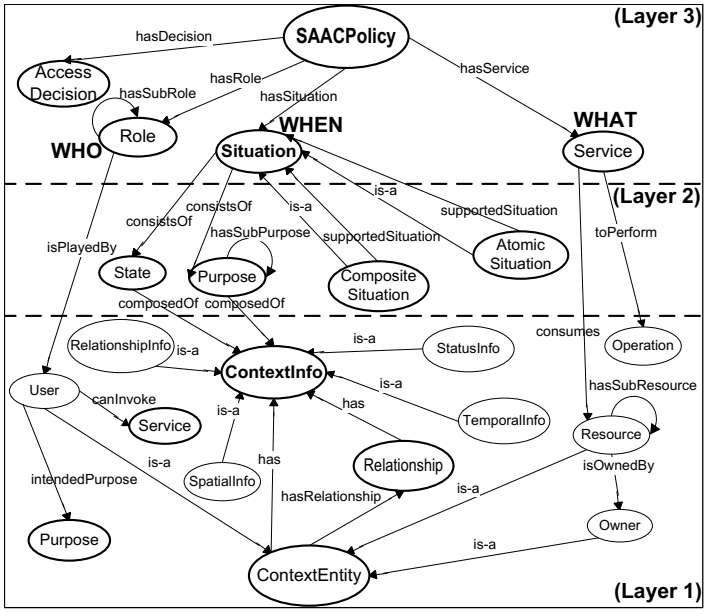


Fig. 1. The PO-SAAC Core Ontology

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. In addition, several 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 to an ontology. 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.

Core Concepts. A graphical representation of the PO-SAAC ontology is shown in Figure 1. We model our ontology based on the 3Ws: ***who*** (user/role) wants to access ***what*** (service) and ***when*** (relevant states and purpose). 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 three layers. The top layer (*Layer 3*) shows the core concepts/elements for specifying the SAAC policies. The middle layer (*Layer 2*) shows the situation modeling concepts. The bottom layer (*Layer 1*) shows the basic concepts for defining the context information and services (resources). The ontology models the following core concepts.

The top layer has the following core concepts, which are organized into *SAACPolicy* hierarchy, namely *Role*, *Situation*, *Service*, and *AccessDecision* classes. A 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.

The middle layer has the situation modeling concepts. 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 services; and it can be identified based on the currently available context information. A *State* can be composed of the relevant context information. A *Situation* can be either an *AtomicSituation* binding a simple situation or a *CompositeSituation* composed by one or more atomic situations using logical operators. (How a new composite situation is specified/reasoned based on the atomic situations by using the ontology-based reasoning rule, is discussed in "Reasoning About Situations" Subsection.) A *Service* consumes a set of software *Resources* to perform certain *Operations*. A *Role* is linked to the *User* class by an object property *isPlayedBy* for representing the fact that a role is played by a user. The *Purpose* class has an object property *hasSubPurpose* to model the purpose hierarchy, so as to achieve the users' service access request at different granularity levels (detail in "Domain-Specific Concepts" Subsection).

The bottom layer has the following core concepts of the context entities and context information. The different relevant entities (*User*, *Resource*, *Owner*) are organized into *ContextEntity* hierarchy. The relationship between a *Resource* and its *Owner* is represented by an object property named *isOwnedBy*. 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 are represented by a number of context information types (*ContextInfo*), namely *RelationshipInfo*, *StatusInfo*, *TemporalInfo*, and *SpatialInfo*. To specify the different relationships between different entities, an object

property *hasRelationship* is used which links *ContextEntity* and *Relationship* classes. A general and extensible context model specific to access control is proposed in our earlier paper [14].

Domain-Specific Concepts. The PO-SAAC core ontology (shown in Figure 1) serves as an entry-point for the domain ontologies. The domain-specific concepts extend the core ontology's corresponding base concepts. It is important for the application developers, providing a way to include domain-specific concepts into the core ontology. Figure 2 shows an excerpt of the representation of the *Purpose* ontology for the healthcare domain (e.g. treatment purpose, research purpose) to exchange patients' medical records. A purpose is identified based on the currently available context information. To identify the purpose, we specify a set of user-defined SWRL rules. An example rule shown in Table 3 identifies the *Purpose* is $DailyOperation(DO)$, based on the current contexts.

Table 3. An Example Rule that Captures the *Purpose* is *DO*

Purpose(?purpose) ∧ Role(?role) ∧ roleID(?role, "RegisteredNurse") ∧ User(?user) ∧ isPlayedBy(?role, ?user) ∧ Resource(?resource) ∧ Owner(?owner) ∧ isOwnedBy(?resource, ?owner) ∧ healthStatus(?owner, "Normal") → **intendedPurpose**(?user, "DO")

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 *granLevel*, which indicates the granularity level. 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 *granLevel* 0 (highest level), which means she also can access for all the other sub-purposes (at the lower granularity levels). A General Practitioner can access a patient's some medical records (e.g., daily medical records) for the *NormalTreatment* purpose. However, she can not access a patient's emergency medical records for *EmergencyTreatment* purpose.

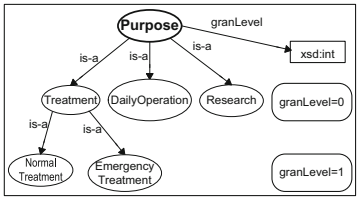


Fig. 2. An Excerpt of Purpose Ontology

We also consider the granularity levels of the healthcare *Role* and *Resource* (patient's medical records) hierarchies, in order to facilitate different 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 [14].

Reasoning about Situations. Various types of ontology-based inferences can be performed for the situation identification and reasoning, service composition and policy evaluation, including implicit knowledge reasoning as well as consistency checking. A set of reasoning rules are specified for implicit knowledge

reasoning, which reasons about the implicit knowledge conveyed by the specification. For example, the rule specified in *Example 3* is written in OWL/XML that is used to reason about a new composite situation ($s_{c1} = s_{a2} \cap s_{a3}, \quad s_{c1} \subseteq S_c$). The specification of these two atomic situations (s_{a2} and s_{a3}) are discussed in *Example 3*. An example policy associated with this composite situation s_{c1} specifies a registered nurse (RN) can access a patient's daily medical records from the general ward (GW) during her ward duty time (DT) for daily operation (DO) purpose. The specification of this composite situation is shown in Table 4.

Table 4. An Example Composite Situation *RNFromGWAtDTForDO*

```
<CompositeSituation rdf:ID="s_c1_RNFromGWAtDTForDO">
    <supportedSituation rdf:resource="#s_a2_RNFromGWForDO"/>
    <supportedSituation rdf:resource="#s_a3_RNAtDTForDO"/>
</CompositeSituation>
<Situation>
    <owl:intersectionOf rdf:parseType="Collection">
        <Situation rdf:about="#s_a2_RNFromGWForDO"/>
        <Situation rdf:about="#s_a3_RNAtDTForDO"/>
    </owl:intersectionOf>
    <rdfs:subClassOf>
        <Situation rdf:about="#CompositeSituation"/>
    </rdfs:subClassOf>
</Situation>
```

5 Prototype Implementation and Evaluation

We have developed our prototype in Java2 SE using widely supported tools and it has been deployed on a Intel(R) Core(TM) Duo T2450 @ 2.00 GHz processor computer with 1GB of memory running Windows XP Professional OS. We have used the Protégé-OWL API to implement the core and healthcare ontologies. During the SAAC policy evaluation phase, an access query is used to process the user's service access request. We have used the SWRL rules to evaluate the policies. We have used the Jess Rule Engine for executing the SWRL rules. In particular, the query language SQWRL, which is based on OWL and SWRL, is adopted to process service access requests.

5.1 Prototype Architecture

A high-level architecture of our prototype framework is shown in Figure 3. We have implemented a set of *Software Components*, which can support the software engineers to develop Situation-Aware Access Control (SAAC) applications using this framework.

Currently a simple Java class *SAACDecisionEngine* is used to check the user's request to access the services and makes situation-specific access control decisions. We have implemented *PolicyEnforcementPoint* as part of the *SAACDecisionEngine*. Once the *SAACDecisionEngine* receives the request

in accessing software services, it queries the *PolicyManager* class for the relevant policies. The *PolicyDecisionPoint*, and *PolicyAdministrationPoint* are implemented as parts of the class *PolicyManager* that are used to allow the

Table 5. An Example SAAC Policy (simplified)

```
<SAACPolicy rdf:ID="sp1">
        <hasDecision rdf:resource="#AccessDecision_Granted"/>
        <hasRole rdf:resource="#EmergencyDoctor_ED"/>
        <hasService rdf:resource="#Service_EMR_Write"/>
        <hasSituation rdf:resource="#sa1_EDFromHospitalForET"/>
</SAACPolicy>
```

engineers to add, edit, and delete access control policies. We have developed a number of *ContextProviders* (which capture low-level context information) and the *ContextReasoner* (which infers high-level information) as parts of the *SituationManager*. The *SituationManager* is used to identify relevant purposes and situations. The ontology knowledge bases are stored in the form of the OWL concepts, SWRL rules and SQWRL queries (PO-SAAC Ontology).

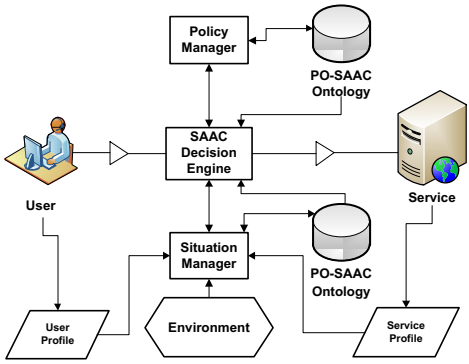


Fig. 3. Our Prototype Architecture

5.2 Developing a SAAC Application for Healthcare

A situation-aware application in the healthcare domain is built to illustrate the operation of our situation-aware access control. The environment of our application is the patients' medical records management (PMRM). The main goal that we aim with this application is to access different medical records of patients based on the relevant situations. We simulate the Java programs and the relational databases as 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_Pofile table (containing patient_id, patient_name and connected_people_id), patient Health_Profile (which provides patient_id, heart_rate and body_temperature), etc.

Policy Specification. Table 5 shows the *Policy #1* written in OWL that is related to our application scenario. In this policy, the access decision ("*Granted*"

decision) is based on the following policy constraints: ***who*** the user is (user's *role*, e.g., "*ED*"), ***what*** service being requested (*service*, e.g., "*writeEMR*()"), and ***when*** the user sends the request (*purpose-oriented situation*). The '*purpose*' and '*situation*' regarding this policy are specified in Examples 1 and 2.

Case Study. To demonstrate the effectiveness of our framework, we present a case study. Consider our application scenario (*Scene #1*), where Jane, by playing an emergency doctor role, wants to access the requested service *writeEMR*() (i.e., the write access to the emergency medical records of patient Bob). The bottom layer of our PO-SAAC ontology captures the relevant context information (interpersonal relationship, location, health status, etc.). The middle layer of the ontology captures the relevant situation based on the captured information and situation specification rules. For the PMRM application we have specified different situations. Some of these 'situations' and their associated 'context information' and 'purpose' using situation reasoning rules are shown in Table 6.

Table 6. Definition of Different Situations

Situation	**Situation Definition (high-level description)**
An emergency doctor from the hospital for emergency treatment (EDFromHospitalForET)	User_Role(ED) ∧ Location_ED(Hospital)∧ Purpose(ET)
A general practitioner from the hospital for normal treatment (GPFromHospitalForNT)	User_Role(GP) ∧ Location_GP(Hospital)∧ Purpose(NT)
A registered nurse from the general ward for daily operation (RNFromGWForDO)	User_Role(RN) ∧ Location_RN(GeneralWard) ∧ Purpose(DO)
A guest researcher from the hospital for research (GRFromHospitalForR)	User_Role(GR) ∧ Location_GR(Hospital)∧ Purpose(R)

Our ontology (top layer) also captures the relevant SAAC policy. Based on this information, the ontology returns the SAAC decision, i.e., Jane's service access request is ***Granted*** (see an access query in Table 7 and result in Table 8), because the ontology captures relevant situation, and satisfies a SAAC policy which is stored in the policy base (ontology knowledge base).

Table 7. An Example Access Query (Simplified)

SAACPolicy(?policy) ∧ User(**?user**) ∧ Role(**?role**) ∧ Situation(**?situation**) ∧ Service(**?service**) ∧ AccessDecision(**?decision**) → **sqwrl:select**(?user, ?role, ?situation, ?service, ?decision)

Table 8. Access Query Result (Shown Only One Entry)

?user	**?role**	**?situation**	**?service**	**?decision**
Jane	ED	EDFromHospitalForET	EMR_Write	Granted

5.3 Performance Evaluation

We evaluate the runtime system performance using our prototype system, where we adopt our PO-SAAC approach to identify and reason about the relevant purpose-oriented situation.

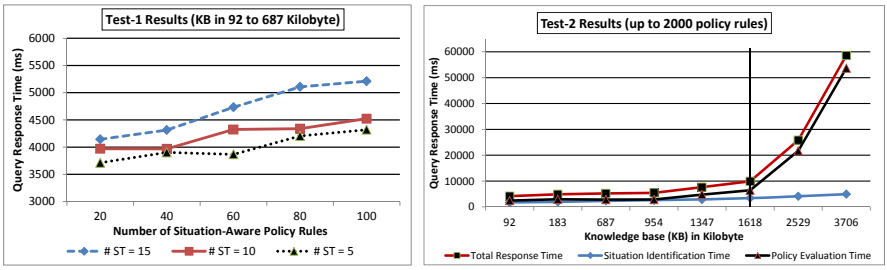
Experimental Setting. With the goal of evaluating the runtime performance of the various stages of our prototype framework, we have conducted two sets of experiments on a Windows XP Professional operating system running on Intel(R) Core(TM) Duo T2450 @ 2.00 GHz processor with 1GB of memory.

Measurement. The main purpose of this experimental investigation is to quantify the performance overhead of our approach. Our main measures included *situation identification time* and *policy evaluation time*. The first measure indicates how long it took to identify/infer a relevant situation (by capturing the currently available context information and identifying the purpose using this information). The second measure indicates how long it took to determine a user's access permission on a requested service (by incorporating the identified situation into the access control process and making situation-aware access control decision). We calculate the average *end-to-end response time* (T_{RT}), time from the arrival of the user's service access request (query) to the end of its execution, which equals to the time for identifying relevant situation and time for evaluating relevant policy.

Results and Analysis. We have examined the performance of PO-SAAC. The main finding was that the time for *making situation-aware access control decision* (based on the *situation identification time* and *policy evaluation time*) is acceptable, as they impose just a small, acceptable overhead.

The ***first test*** focuses on measuring the response time of our prototype in the light of 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 emergency treatment purpose is 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 (see test results in Figure 4(a)). The test results 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 seconds* for 15 types of situation information and for the variation of 20 to 100 policy rules. We can see that the average response time seems to be linear. Overall, the performance is acceptable.

In the ***second test***, we have again evaluated the total response time (situation identification and policy evaluation time) 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 [15] (i.e., 138 roles). To build the ontology KB 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 (see test results in Figure 4(b)). As the size



(a) Number of Policies V Response time (b) Size of KB V Response time

Fig. 4. Average Response Time Over Different Number/Size of Policies/KB

of the ontology KB increases beyond 1618 kilobyte, the computational overhead increases dramatically. This is due to the large number of policy rules (larger KB size), fully utilizing the memory capacity of the computer. At the point of the KB being 1618 kilobyte (500 policy rules for the 138 health professional roles), it takes *approximately 10 seconds* to process the request. We can see that changes to the number of access control policies do not have much impact on the response time, when the ontology knowledge base (KB) size is small. Overall, the runtime performance is acceptable for a reasonable sized KB.

6 Related Work and Discussion

In [9], Kim and Lim propose the Situation-Aware Role-Based Access Control (SA-RBAC) model, which extends the basic RBAC model [11] and dynamically grants roles (or permissions) to users based on the situation information of the user. The SA-RBAC model is used to deal with the situation information by considering the combination of the required credentials of users, and the context information such as location, time, and system resources relevant to the user's access request. In [16], Yau and others have defined the 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 [10]. SA-AC model incorporates situation-aware constraints into RBAC model, such that the states of service providers, requesters and environments, which can affect the access control decisions. These approaches only consider the states of the relevant entities as the policy constraints. In open and dynamic environments, however, the states of the relevant relationships between different 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 different relevant entities. Moreover, in our approach, the purpose or user's intention in accessing the services is also considered for modeling situation.

Previous works on ontology-based context/situation-awareness also provide valuable insights for modeling a fine-grained ontology-based SAAC framework.

The CONtext ONtology (CONON) [6], situation theory ontology (STO) [7] and situation-awareness (SAW) ontology [8] research describe 'situation' as the states of the specific kind of entities (e.g., attributes of users or other relevant entities). However, this research are highly domain-specific and they do not consider several important concepts which are important consideration for situation modeling in today's dynamic environments: the states of the relevant relationships between entities, and the purpose or user's intention in accessing the services.

Byun and Li [3] have proposed a privacy preserving access control model for relational databases where 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. In [17], the authors 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 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 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.

7 Conclusion and Future Work

In this paper, we have presented a new Purpose-Oriented Situation-Aware Access Control 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. Another contribution of this paper is an *ontology-based development platform*, in order to formalize PO-SAAC model using OWL and SWRL. The practical applicability of our framework is demonstrated through the implementation of a software prototype. In addition, we have developed a SAAC application in the healthcare domain and presented a case study. The case study shows that our framework captures relevant situations at runtime and invokes software services in a situation-aware manner. The experimental results have shown that our framework has satisfactory performance. Future work focus on the scalability of our framework in the mobile platform will be an important issue to be addressed.

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

References

1. Dey, A.K.: Understanding and using context. Personal Ubiquitous Computing 5(1), 4–7 (2001)
2. Weiser, M.: Some computer science issues in ubiquitous computing. Commun. ACM 36(7), 75–84 (1993)
3. Byun, J.-W., Li, N.: Purpose based access control for privacy protection in relational database systems. The VLDB Journal 17(4), 603–619 (2008)
4. Dimitropoulos, L.L.: Privacy and security solutions for interoperable health information exchange: nationwide summary. AHRQ Publication (2007)
5. Endsley, M.R.: Design and evaluation for situation awareness enhancement. In: Proceedings of the Human Factors Society 32nd Annual Meeting, Santa Monica, CA, USA, pp. 97–101 (1988)
6. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using owl. In: Proceedings of the Second PerCom Workshops, pp. 18–22 (2004)
7. Kokar, M.M., Endsley, M.R.: Situation awareness and cognitive modeling. IEEE Intelligent Systems 27(3), 91–96 (2012)
8. Yau, S.S., Huang, D.: Development of situation-aware applications in services and cloud computing environments. International Journal of Software and Informatics 7(1), 21–39 (2013)
9. Kim, Y.G., Lim, J.: Dynamic activation of role on rbac for ubiquitous applications. In: Proceedings of the 2007 International Conference on Convergence Information Technology, pp. 1148–1153 (2007)
10. Yau, S.S., Liu, J.: A situation-aware access control based privacy-preserving service matchmaking approach for service-oriented architecture. In: ICWS, pp. 1056–1063 (2007)
11. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29, 38–47 (1996)
12. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing 6, 161–180 (2010)
13. Riboni, D., Bettini, C.: Owl 2 modeling and reasoning with complex human activities. Pervasive and Mobile Computing 7, 379–395 (2011)
14. Kayes, A.S.M., Han, J., Colman, A.: An ontology-based approach to context-aware access control for software services. In: Lin, X., Manolopoulos, Y., Srivastava, D., Huang, G. (eds.) WISE 2013, Part I. LNCS, vol. 8180, pp. 410–420. Springer, Heidelberg (2013)
15. ASCO: Health professionals (Jul 2013), `http://www.abs.gov.au/`
16. Yau, S.S., Karim, F., Wang, Y., Wang, B., Gupta, S.K.S.: Reconfigurable context-sensitive middleware for pervasive computing. IEEE Pervasive Computing 1(3), 33–40 (2002)
17. Sun, L., Wang, H., Soar, J., Rong, C.: Purpose based access control for privacy protection in e-healthcare services. JSW 7(11), 2443–2449 (2012)