RelBOSS: A Relationship-Aware Access Control Framework for Software Services

A.S.M. Kayes, Jun Han, Alan Colman, and Md. Saiful Islam

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

Abstract. *Context-awareness* is an important aspect of the *dynamically changing environments* and the *relationship context information* brings new benefits to the access control systems. Existing relationship-aware access control approaches are highly domain-specific and consider the expression of access control policies in terms of the relationship context information. However, these approaches are unable to dynamically capture the granularity levels and strengths of the relevant relationship. To this end, in this paper we present a formal *Relationship-Aware Access Control (RAAC)* model for specifying the relevant *relationship context information* and the corresponding *access control policies*. Using the *RAAC* model, we introduce an ontology-based framework, *Relationship-Based access control Ontology for Software Services* (*RelBOSS*). One of the main novelties of the framework is that it dynamically captures the *relationship context information* (the *type/name*, *granularity levels* and *strengths* of the relevant relationship). Experiments with a software prototype confirm the feasibility of our framework.

Keywords: Relationship-aware service, Relationship context information, Relationship-aware access control, Access control policy.

1 Introduction

The rapid advancement of computing technologies [1][2] has led the world to a new paradigm of access control from static to dynamic context-aware environments [3]. Such a paradigm shift brings with it lots of opportunities and challenges. On the one hand, users demand access to software services in which the relationship information need to be explicitly/implicitly captured for the purpose of access control decision making. On the other hand, such access needs to be carefully controlled due to the additional challenges of capturing the granularity levels and strengths of that relationship. Consider a software service in a healthcare domain, a doctor requests a patient's electronic health records. This access request should be granted based on the relevant relationship between the doctor and the corresponding patient, e.g., only a treating doctor can access the patient's private health records. Such service access request can be realized by modeling the relationship with certain granularity levels (e.g., 'TreatingDoctor-Patient' or 'Doctor-Patient') and strengths ('strong' or 'weak') from the available context information. The above challenges require a new *relationship model* for capturing the *relationship context information* and a *policy model* for making *relationship-aware access control decisions.*

Existing context definitions describe the information which characterizes the states of the entities [4][5][6]. As proposed by Dey [5], *"context is any information that can be used to characterize the situation of an entity (an entity may be a person, place or object)"*. However, other than the information about entities, the information about relationship between relevant entities (relationship context information) is not considered in this context-awareness research.

The relationship-aware access control approaches (e.g., [1], [7], [8]) consider the expression of access control policies in terms of the relationship information. These approaches only consider the *person-centric* or interpersonal relationships to control access to resources or services. However, the different granularity levels and strengths of the relevant relationship are not considered in these works. Carminati et al [1] consider the relationships between users (e.g., *friend*, *friend-of-friend*) in the access control processes. Zhang et al [7] consider the access permissions as binary relations between subjects and objects, and ReBAC [8] considers the relationships between individual users (e.g., *professional association*) in the access control policies. Different from these approaches, our approach represents much richer forms of relationships among relevant entities with different granularity levels and strengths of that relationship, and expresses the access control policies in terms of this relationship context information.

The Contributions. In this paper we present a framework, *RelBOSS*, in order to provide relationship-aware access to information resources and software services. The novel features of this framework are as follows:

(C1) *Relationship Model.* Our framework uses the *relationship context information* to provide relationship-aware access to software services (authorization), where we present a *relationship model* to represent and reason about the relevant relationship with different granularity levels and strengths.

(C2) *Relationship-Aware Access Control Policy Model.* Our framework presents a *policy model* to specify relationship-aware access control policies. The policy model supports access control to the appropriate software services based on the relevant relationship context information.

(C3) *Ontology-Based Framework Implementation.* Based on the relationship and policy models, we introduce an *ontology-based framework* for identifying relationship context information, and enforcing relationship-aware access control policies. Our ontology-based framework represents the basic context entities and information using the ontology language OWL, extended with SWRL for identifying and reasoning about relevant relationship context information and the corresponding access control policies.

(C4) *Prototype Implementation and Evaluation.* We have presented a *software prototype* for the development of the relationship-aware access control applications and demonstrated the effectiveness of our framework through a healthcare *case study.* 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.

Paper Outline. The rest of the paper is organized as follows. Section 2 presents a healthcare application scenario to motivate our work. Next, we present a formal definition of our framework, a relationship model to specify relationship context information and a policy model for specifying relationship-aware access control policies in Section 3. Section 4 presents an ontology-based development platform for our framework. Next, we describe the prototype implementation along with the viability of the framework in Section 5. After that, we discuss the related work in Section 6. Finally, Section 7 concludes the paper.

2 Research Motivation and General Requirements

In this section, we first outline a motivating scenario that illustrates the need for the incorporation of relationship context information in the access control processes. We then distill the requirements for managing the access to software services in a relationship-aware manner.

Motivating Scenario. *The scenario begins with patient Bob who is in the emergency room due to a heart attack. Jane, a medical practitioner of the hospital, is required to treat Bob and needs to access Bob's electronic health records (e.g., emergency medical records, private health records) in an emergency health condition. However, in a normal health condition, only the treating practitioners should be able to access the patient's private health records, if he/she is present in the same location with patient.*

The relationship information describing the context entities relevant to access control in the above scenario includes: the *type/name* of the relationship between the service requester and the service owner (e.g., 'User-Owner' relationship), the different *granularity levels* of the relevant relationship (e.g., 'User-Owner' relationship, at granularity level 0 (i.e., 'Doctor-Patient' relationship), 'User-Owner' relationship, at granularity level 1 (i.e., 'TreatingDoctor-Patient' or 'EmergencyDoctor-Patient' relationship), the *strengths* of the relevant relationship (e.g., the *strengths* of the 'User-Owner' relationship is 'strong' or 'weak'). This relationship context information needs to be identified/inferred based on the currently available context information using user-defined rules.

General Requirements. To support such relationship-aware access control in a computer application like an electronic health records management system, we need to consider the 3Ws: ***who*** (the appropriate users/roles) wants to access ***what*** (the appropriate software services), and ***when*** (the relationship context information). As different types of relationship context information are integrated into the access control processes, some important issues arise. These issues and their related requirements are as follows:

(R1) *Representation of relationship context information:* What access control-specific elementary information should be identified as part of building a relationship model specific to relationship-aware access control? Furthermore, how to represent and reason about the relevant relationship context information based on the currently available context information?

(R2) *Specification of relationship-aware access control policies:* How to define the access control policies based on the relevant relationship context information to realize a flexible and dynamic access control scheme?

(R3) *Implementation framework:* How to realize the relevant relationship context information and the corresponding relationship-aware access control policies in an effective way, in oder to access/manage software services?

3 Relationship-Aware Access Control

In this section, we present our conceptual Relationship-Aware Access Control (RAAC) framework for Software Services.

In our previous work [9], we define the *context information* in which the two parts relevant to making access control decisions are: *the information characterizing the entities* and *the information characterizing the relationships between different entities*. In this paper, we elaborate the second part of the proposed context definition. Our main focus is to capture/infer the fine-grained relationship (the type/name, granularity levels and strengths) between different person-centric entities (*interpersonal* relationship), in order to control access to services depending on the relevant relationship context information.

Definition 1 (Relationship Context Information). The *relationship context information* used in an access control decision is defined as the relevant relationship (type/name) with different granularity levels and strengths of that relationship. The relationship *type*, *granularity levels*, and *strengths* are domain-dependent concepts, and their values can be obtained (captured/inferred) based on the access request (e.g., from the sensed contexts, inferred contexts, etc.).

The relationship context information '*Rel*' can be formally defined as a tuple in the form of

$$\begin{aligned} Rel &= \{rel_1, rel_2, ..., rel_n\} \\ rel &= < rel.name, rel.gLevel, rel.strength > \end{aligned} \tag{1}$$

where:

- *rel.name* denotes a relationship type, e.g., *'user-owner'* relationship.
- *rel.gLevel* denotes the different granularity levels of the relationship, e.g., *'user-owner' relationship at granularity level 1, i.e., 'treatingDoctor-patient' or 'emergencyDoctor-patient' relationship.*
- *rel.strength* denotes the relationship strengths of the relationship ('strong' or 'weak' relationship), e.g., *the strengths of the relationship between doctor and patient is 'strong' relationship.*

How to capture/represent the *type* of the relevant relationship, and how to identify/infer the different *granularity levels* and *strengths* of that relationship are discussed in the following sub-sections.

3.1 Representation of Relationship

A *relationship type* used in an access control decision can be captured based on the currently available context information.

Example 1: Concerning our application scenario, a relevant access control policy is as follows: a user Jane, who is a medical practitioner, by playing an emergency doctor (ED) role, is allowed to access ('read' or 'write' operation) the patient's emergency medical records (EMR). The following rule (2) is used to identify that the relationship *type/name* is "user-owner".

$$\begin{aligned} User(u) \wedge Owner(o) \wedge Resource(res) \wedge Relationship(rel_1) \wedge requests(u, \\ res) \wedge equal(res, \text{“}EMR\text{”}) \wedge owns(res, o) \wedge hasRelationship(u, rel_1) \\ \wedge\ hasRelationship(o, rel_1) \wedge RelationshipName(rN) \\ \wedge\ has(rel_1, rN) \rightarrow\ rel.name(rN, \text{“}user - owner\text{”}). \end{aligned} \tag{2}$$

3.2 Reasoning about Relationship

The process of inferring the *relationship granularity levels* and *strengths* from the currently available context information is referred to *reasoning about relationship*. One of the main advantages of our framework to relationship-awareness is its reasoning capability; that is, once facts about the world have been stated, other facts can be inferred using an inference engine through the reasoning rules.

The *relationship granularity levels* and *strengths* can be inferred by performing logical composition on the relevant context information.

Example 2: Consider the policy mentioned in *Example 1*, the *granularity levels* of the relationship between user and owner can be inferred using the following rule (3) (i.e., a user by playing the 'ED' (emergency doctor) role can access a patient's medical records, when the relationship granularity is "emergency doctor-patient").

$$\begin{aligned} User(u) \wedge Role(r) \wedge plays(u, r) \wedge Relationship(rel_1) \\ \wedge\ equal(r, \text{“}ED\text{”}) \wedge RelationshipGranularity(rG) \\ \wedge\ has(rel_1, rG) \rightarrow\ rel.gLevel(rG, 1). \end{aligned} \tag{3}$$

The above rule (3) identified the *granularity levels* of the relationship between resource requester and resource owner, which is 1 (i.e., "emergencyDoctor-patient" relationship). If the user in a doctor role, then the *granularity levels* of the relationship is 0 (i.e., "doctor-patient" relationship).

Example 3: Consider the same policy mentioned in *Example 1*, the *strengths* of the relationship between user and owner can be inferred using the rule (4).

$$\begin{aligned} User(u) \wedge Owner(o) \wedge Relationship(rel_1) \wedge RelationshipStrength(rS) \\ \wedge\ userIdentity(u, uID) \wedge connectedPeopleIdentity(o, cpID) \\ \wedge\ has(rel_1, rS) \wedge equal(uID, cpID) \rightarrow\ rel.strength(rS, \text{“}strong\text{”}). \end{aligned} \tag{4}$$

The above rule (4) identified the *strengths* of the relationship between resource requester (doctor) and resource owner (patient), which is "strong" (i.e., the physician is assigned as a treating doctor of the patient). If the physician is not a treating doctor of the patient, then the relationship strength is "weak".

As a whole, the relevant relationship context information rel_1 between resource requester and resource owner (relationship *name* is 'user-owner') associated with the above-mentioned policy is represented as:

$$rel_1 = \; < \text{``}user - owner\text{''}, 1, \text{``}strong\text{''} > \tag{5}$$

3.3 The RAAC Policy Model for Software Services

Based on the formalization of the traditional role-based access control (RBAC) model in [10], we present a formal definition of our relationship-aware access control (RAAC) policy model. Our policy model for RAAC applications extends the RBAC with relevant relationship context information. Our goal in this research is to provide a way in which the role-permission assignment policies in RBAC [10] that can be specified by incorporating relationship context information as policy constraints.

Definition 2 (RAAC Policy Model). Our relationship-aware access control policy model can be formally described as a tuple, where R, Rel, Ser, and $RAACPolicy$ represents Roles, Relationship Context Information, Services, and Policies, respectively (see Formula (6)):

$$M_{RAAC} = (R, Rel, Ser, RAACPolicy) \tag{6}$$

– ***Roles (R):*** R represents a set of roles that reflect user's job functions or job titles within the organization (e.g., healthcare domain). The users or service requesters can play the roles and they are human-beings, whose service access requests are being controlled.

$$R = \{r_1, r_2, ..., r_m\} \tag{7}$$

– ***Relationship (Rel):*** Rel represents the relationship context information in order to describe the relationship-aware access control policies.

$$Rel = \{rel_1, rel_2, ..., rel_n\} \tag{8}$$

– ***Services (Ser):*** In this paper, we consider the resource (e.g., patients' medical health records) in a service-oriented manner. A service request ser can be seen as a *pair <res, op>*, $ser = \{(res, op) \; |res \in Res, op \in OP\}$, where Res represents a set of resources, $Res = \{res_1, res_2, ..., res_q\}$ and OP represents a set of operations on the resources, $OP = \{op_1, op_2, ..., op_r\}$. For example, the write operation on the emergency medical records is defined as *<EMR, write>*. In this way, the fine-grained access control to the appropriate parts of a resource by the appropriate users can be realized.

$$Ser = \{ser_1, ser_2, ..., ser_o\} \tag{9}$$

- ***Policies (RAACPolicy):*** *RAACPolicy* represents the relationship-aware access control policies. Our RAAC policy model has relationship-aware role-service assignment policies to provide relationship-aware access to software services. In our RAAC policy, a service is considered as a software entity, in order to provide fine-grained access control and grant the right operation (e.g., particular mode of access, such as 'read' and/or 'write') to the appropriate resources (e.g., private health records, emergency medical records).

$$RAACPolicy = \{rp_1, rp_2, ..., rp_p\} \tag{10}$$

Our RAAC policy model extends the concept of role-permission assignments (*RPA*) in RBAC (*RPA* $\subseteq R \times P$) [10], by introducing the concept of relationship context information, called relationship-aware role-service assignments.

$$\begin{aligned} RAACPolicy = \{&(r_1, rel_1, ser_1), ((r_2, rel_2, ser_2),\\ &..., ((r_m, rel_n, ser_o)\} \subseteq R \times Rel \times Ser \end{aligned} \tag{11}$$

Table 1. An Example of Relationship-Aware Access Control Policy

If
$RAACPolicy(rp_1) \wedge User(u_1) \wedge hasUser(rp_1, u_1) \wedge Role(r_1) \wedge plays(u_1, r_1)$
$\wedge\ hasRole(rp_1, r_1) \wedge equal(r_1, \text{“}ED\text{”}) \wedge Service(ser_1) \wedge hasService(rp_1, ser_1)$
$\wedge\ equal(ser_1, \text{“}writeEMR()\text{”}) \wedge Relationship(rel_1) \wedge hasRelationship(rp_1, rel_1)$
$\wedge\ rel.name(rel_1, nM) \wedge equal(nM, \text{“}user - owner\text{”}) \wedge rel.gLevel(rel_1, gL)$
$\wedge\ equal(gL, 1) \wedge rel.strength(rel_1, sT) \wedge equal(sT, \text{“}strong\text{”})$
Then
$canInvoke(u_1, ser_1)$

Example 4: Based on our policy model (*Role(r_1)* $\wedge$ *Relationship(rel_1)* $\wedge$ *Service(ser_1)* $\rightarrow$ *(r_1, rel_1, ser_1)* $\in$ *RAACPolicy*), the above-mentioned rule (shown in Table 1) expresses the RAAC policy mentioned in Example 1, i.e., a User 'u_1' by playing the Role 'r_1' (emergency doctor role "*ED*") can invoke the Service 'ser_1' (< *EMR*, *write*> or "*writeEMR*()" service), if a relationship 'rel_1' (named "*user-owner*" (*rel.name*), with "*emergencyDoctor-patient*" granularity (*rel.gLevel* is 1) and "strong" strength (*rel.strength*)) is satisfied.

The identification of the relevant information to represent the *relationship context information* and specify the corresponding *RAAC 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 framework, RelBOSS.*

4 RelBOSS: The Ontology-Based RAAC Framework

Based on our proposed *RAAC Model* (relationship and policy models), we in this section present an ontology-based framework, *Relationship-Based access control Ontology for Software Services (RelBOSS).* The principal goal of our RelBOSS framework is to realize the concepts of relationship and policy models using a logic-based language. To achieve this goal, we have already identified relevant concepts in the previous section.

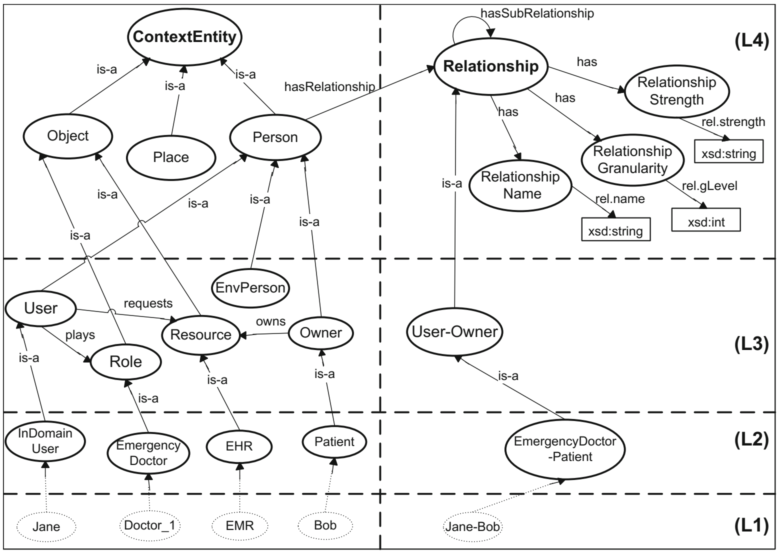


Fig. 1. Core ***Relationship*** Ontology

4.1 Design Considerations

In the present age, ontology-based modeling technique has been proven as a suitable logical language for modeling dynamic context information (e.g., relationship context information) [4], because of its representational and reasoning viewpoints. We adopt the OWL language [11] as an ontology language to model RelBOSS. The OWL language has been chosen for most ontological applications because of its considered trade-off between computational complexity of reasoning and expressiveness [4]. In order to support the process of inferring implicit knowledge (the *granularity levels* of the relationship, the relationship *strengths*), we define a set of user-defined reasoning rules that are associated with the basic knowledge (currently available context information). Towards this end, the expressivity of OWL is extended by adding SWRL [12] rules to RelBOSS.

4.2 Relationship Ontology

4.2.1 Representation of Relationship - Core Concepts

Figure 1 shows the conceptual view of our *Relationship* ontology (definition of upper concepts/entities and partial definition of domain-specific concepts). The *Relationship* ontology can be divided into four layers. The top layer *(L4)* shows the core relationship-aware concepts for specifying the relationship context information. The middle layer *(L3)* shows the constructs for defining access control-specific core concepts. The middle layer *(L2)* shows the domain-specific concepts that can be modeled using the core concepts. The bottom layer *(L1)* shows the instances based on the healthcare application used in this paper. The constructs in the ontology are summarized as follows.

The layer *(L4)* is structured around a set of core entities, each describing the concepts including *ContextEntity* (for defining access control-specific core classes/entities: *Person*, *Place* and *Object* types) and *Relationship* (for defining the relationship context information). Each entity is associated with its attributes (represented in *data type* property), relations with other entities (represented in *object* property), and its subclasses (represented in *is-a*, i.e., *is a subclass of* property). An object property *hasRelationship* is used to link the *Person* and the *Relationship* classes. The *Relationship* class is connected with three other classes: *RelationshipName*, *RelationshipGranularity* and *RelationshipStrength.* The *RelationshipName* class contains a data type property named *rel.name* (*xsd:string* type), which denotes the type of the relationship. The *RelationshipGranularity* class contains a data type property named *rel.gLevel* (*xsd:int* type), which denotes the different granularity levels of the relationship existing between the entities participating in a certain situation. The *Relationship-Strength* class contains a data type property named *rel.strength* (*xsd:string* type), which denotes the strengths of the relationship. The *Relationship* class has an object property *hasSubRelationship* to model the relationship hierarchy, so as to achieve the users' service access request at different granularity levels (detail in "Representation of Relationship - Domain-Specific Concepts" Subsection).

The layer *(L3)* defines the access control-specific core entities. The *User*, *Role*, *Resource*, *EnvPerson* and *Owner* classes are the subclasses of the core classes. A built-in property of the OWL, named *is-a*, is used to link between *ContextEntity* class and its subclasses. The relationship between user and owner is represented as *User-Owner* class, which is a subclass of the core class *Relationship.* We also define some of the data type properties: *userIdentity* is the attribute of the class *User*, to capture user identity; *roleIdentity* is the attribute of the class *Role*, to capture role identity; *resourceIdentity* is the attribute of the class *Resource*, to capture requested resource identity; and *connectedPeopleIdentity* is the attribute of the class *Owner*, to capture connected people identity. For the sake of simplicity, Figure 1 does not capture/present these data type properties. Overall, the layers 3 and 4 models the core/general concepts. A general context model specific to access control is proposed in our earlier work [9]. Our relationship ontology extends the existing context model, in order to derive fine-grained relationship (the type/name, granularity levels and strengths of the relationship).

4.2.2 Representation of Relationship - Domain-Specific Concepts

The core Relationship 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.

The layer *(L2)* in Figure 1 shows a partial definition of domain-specific entities/concepts for the healthcare domain. For example, the core classes *Role* and *Resource* of the healthcare domain are instantiated into subclasses *Emergency-Doctor* and electronic health records (*EHR*) respectively. The layer *(L1)* shows some example instances based on the healthcare scenario used in this paper. For example, the instances *Jane* and *Bob* are represented as *InDomainUser* and *Patient*, the emergency medical records (*EMR*) is the instance of *EHR*, etc.

The different relationships at various granularity levels of a user's service access request are individually identifiable, so as to achieve fine-grained relationship-aware control over access to services. Towards this end, we consider a detailed ontology with different levels of relationships for the healthcare application. We express the existence of different relationships at fine-grained levels. Figure 2 shows an excerpt of the representation of the *Relationship* ontology for the healthcare domain. In Figure 2, we only model the interpersonal relationships between user and owner (*User-Owner* class). As such, the *Relationship* class contains an important property (*xsd:int* type) named *rel.gLevel*, which indicates the relationship granularity levels. For instance, a medical practitioner can access a patient's electronic health records with the "*Doctor-Patient*" relationship (*rel.gLevel* = 0). However, he/she can access the emergency medical records, if he/she is an emergency doctor of the patient (*rel.gLevel* = 1, i.e., "EmergencyDoctor-Patient" relationship).

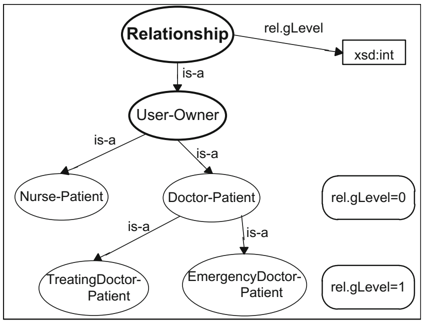


Fig. 2. An Excerpt of Relationship Ontology for Healthcare

4.3 Reasoning About Relationship

A more flexible reasoning mechanism is user-defined reasoning. Through the creation of SWRL user-defined reasoning rules, the relationship context information can be deduced from the basic context information. A set of reasoning rules are specified for implicit knowledge reasoning, which reasons about the implicit knowledge (the *name* or *type* of the relationship, and the *granularity levels* and *strengths* of that relationship) convoyed by the specification.

4.3.1 Example Rules

We specify a set of SWRL reasoning rules, to derive the name of the relationship between User and Owner. For example, Rule #1 in Table 2 is used to derive the name (*rel.name*) of the relationship (which is "*User-Owner*"). The rule states that ***if*** a User requests a Resource which is owned by a Patient (Owner), ***then*** the relationship name/type is "*User-Owner*".

Another set of SWRL reasoning rules are also specified in Table 2, to derive the granularity levels (*rel.gLevel*) of the relevant relationship. For example, ***if*** the relationship between user and owner is "*EmergencyDoctor-Patient*", ***then*** the granularity level is 1 (see Rule #2 in Table 2, where a user plays the Emergency Doctor (ED) role). ***If*** the user in a Doctor role, ***then*** the granularity level is 0 ("*Doctor-Patient*").

We further specify a set of reasoning rules to derive the *strengths* of the relationship, based on the basic context information. An example rule shown in Table 2 (see Rule #3) identifies the relationship strengths (*rel.strength*) is "*Strong*". Note that this rule has used the basic information, the user's profile information

Table 2. User-defined SWRL Reasoning Rules

Rule	Relationship	Reasoning Rules
1	Relationship Name	**User**(?u) ∧ **Resource**(?res) ∧ requests(?u, ?res) ∧ swrlb:equal(?res, "EMR") ∧ **Owner**(?o) ∧ Resource(?res) ∧ owns(?o, ?res) ∧ Relationship(?rel$_1$) ∧ hasRelationship(?u, ?rel$_1$) ∧ hasRelationship(?o, ?rel$_1$) ∧ **RelationshipName**(?rN) ∧ has(?rel$_1$, rN) → **rel.name**(?rN, "User-Owner")
2	Granularity Level	**User**(?u) ∧ Role(?r) ∧ plays(?u, ?r) ∧ swrlb:equal(?r, "ED") ∧ Relationship(?rel$_1$) ∧ **RelationshipGranularity**(?rG) ∧ has(?rel$_1$, rG) → **rel.gLevel**(?rG, 1)
3	strengths	**User**(?u) ∧ **Owner**(?o) ∧ Relationship(?rel$_1$) ∧ **RelationshipStrength**(?rS) ∧ userIdentity(?u, ?uID) ∧ connectedPeopleIdentity(?o, ?cpID) ∧ swrlb:equal(?uID, ?cpID) ∧ has(?rel$_1$, rS) → **rel.strength**(?rS, "Strong")

(*userIdentity*) and the patient's profile information (*connectedPeopleIdentity*). This basic information can be obtained through a program (e.g., in Java) from the data sources (e.g., RDBMS). For the motivating scenario, this rule determines that Jane and Bob has a relationship with "*Strong*" strength, because Jane is a treating doctor of patient Bob. ***If*** Jane is not a treating doctor of patient Bob, ***then*** the relationship strengths is "*Weak*".

4.4 The RAAC Policy Ontology

In the following, we present the two main parts of our RAAC policy ontology: *policy specification* (integrating relationship context information into the access control policies) and *policy enforcement* (determining access control decisions).

4.4.1 Policy Specification

Figure 3 shows the top-level conceptual view of our *RAAC* policy ontology.

We model our policy ontology based on the 3Ws: ***who*** (user/role) wants to access ***what*** (service) and ***when*** (relationship context information). A *RAAC Policy* is a collection of entities and it includes *User*, *Role*, *Relationship*, *Service*, and *AccessDecision* classes. The policy model allows the specification of RAAC policies for software services, where the authorized users (an authorized user is a user with a role instance) can invoke appropriate services in terms of the relevant relationship context information. It uses the *Relationship* concept from the relationship ontology (presented in Figure 1), to capture the relationship context information. The RAAC policy ontology also uses the concepts, *User*, *Role*, and *Resource* from the relationship ontology in order to capture the relevant context information at runtime. The reused classes are shown in shaded ellipses (see Figure 3).

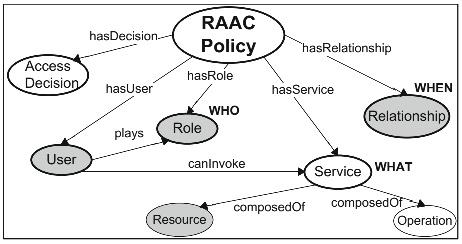


Fig. 3. The RAAC Policy Ontology

We characterize a *Service* as a composition of *Operations* on *Resources*, through which information resources or software services are invoked by the users or service requesters. An object property, named *composedOf*, is used to link between *Service* and *Resource* classes, and *Service* and *Operation* classes (see Figure 3). Table 3

Table 3. ***Service Definition*** and An Example Service ***Service_EMR_write***

```
<owl:ObjectProperty rdf:ID="composedOf">
    <rdfs:domain rdf:resource="#Service"/>
     <rdfs:range>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Resource"/>
                <owl:Class rdf:about="#Operation"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:range>
</owl:ObjectProperty>
<Service rdf:ID="Service_EMR_write">
    <composedOf rdf:resource="#Resource_EMR"/>
    <composedOf rdf:resource="#Operation_write"/>
</Service>
...
```

shows a service definition in OWL, a *Service* is composed of the collection/union of *Resource* and *Operation* (see the top part of the Table 3). An example of a service, named *Service_EMR_write* is shown in Table 3 also, which is composed of the *write* operation on a resource (e.g., a patient's emergency medical records (EMR)) (see the bottom part of the Table 3).

Definition 3 (RAAC Policy Specification). A relationship-aware access control policy captures the ***who/what/when*** dimensions which can be read as follows: a *RAACPolicy* specifies that a *User* who is playing a *Role* has *Access-Decision* ("Granted" or "Denied") to *Service* if the relevant *Relationship context information* is satisfied.

Table 4. An Example RAAC Policy (Upper Level)

```
<RAACPolicy rdf:ID="rp1">
    <hasUser rdf:resource="#User_plays_ED"/>
    <hasRole rdf:resource="#Role_EmergencyDoctor_ED"/>
    <hasService rdf:resource="#Service_EMR_write"/>
    <hasRelationship rdf:resource="#Relationship_name_gLevel_strength"/>
    <hasDecision rdf:resource="#AccessDecision_Granted"/>
</RAACPolicy>
...
```

Table 4 shows the policy (which is mentioned in Example 4) 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 (e.g., "*writeEMR()*"), and ***when*** the user sends the request (*relationship context information*). The '*rel.name*', '*rel.gLevel*', and '*rel.strength*' regarding this *Relationship* are derived based on the SWRL reasoning rules (shown in Table 2).

4.4.2 Policy Enforcement/Evaluation

The specified RAAC policies are enforced to ensure the appropriate use of information resources or software services. Towards this end, we have implemented the *RAAC Policy Enforcement Point* and *RAAC Policy Decision Point* in Java, inspired by the XACML architecture [13]. The prototype architecture is presented in the next Section (see Figure 4). Using our developed prototype, the policy enforcement (relationship-aware access control decisions) is performed according to the following data flow sequences:

1. The users (service requesters) send the access requests to access the resources/services which get processed by our developed *RAAC Policy Enforcement Point*.
2. Once receiving the request for service access, the *RAAC Policy Enforcement Point* queries the applicable access control policies and the relevant relationship context information to the *RAAC Policy Decision Point*.
3. Based on the selected access control policies in the policy ontology, the *RAAC Policy Decision Point* makes the access control decision by runtime assessing the relationship context information from the relationship ontology.
4. The decision made by *RAAC Policy Decision Point* is passed to *RAAC Policy Enforcement Point*. If it is admissible, the access request is granted, otherwise, it is denied. Being granted to access the services (i.e., if the decision is "granted") means that the users are able to access the services. If the decision is "denied", a denied response is sent back to the users.

During the policy evaluation phase, an *access query* is used to check the user's access request (*who can access what services*). We use the query language SQWRL [14], which is based on OWL and SWRL. When a service access request comes, the *RAAC Policy Decision Point* identifies the applicable policies using the policy ontology and identifies the relationship context information using the relationship ontology. Then the formulated access query is used to check the user's request to access the services using both the policy constraints and the relevant relationship context information. For the application example in the motivating scenario, the defined access query (see an access query in Table 5) is used to determine whether the access is *Granted* or *Denied*. One of the entries in the access query results satisfies Jane's service access request ("*EMR_write*") is *Granted* (see Table 6).

Table 5. An Example Access Query (Simplified)

RAACPolicy(?policy) $\wedge$ User(**?user**) $\wedge$ Role(**?role**) $\wedge$ rel.name(**?rel_nM**) $\wedge$ rel.gLevel(**?rel_gL**) $\wedge$ rel.strength(**?rel_sT**) $\wedge$ Service(**?service**) $\wedge$ AccessDecision(**?decision**) $\rightarrow$ **sqwrl:select**(?user, ?role, ?rel_nM, ?rel_gL, ?rel_sT, ?service, ?decision)

Table 6. Access Query Result (Shown Only One Entry)

?user	**?role**	**?rel_nM**	**?rel_gL**	**?rel_sT**	**?service**	**?decision**
Jane	ED	User-Owner	1	Strong	EMR_write	Granted

5 Prototype Implementation and Evaluation

5.1 Prototype Implementation

Figure 4 shows a high-level conceptual view of our prototype framework in Java2 SE using widely supported tools.

We have used the Protégé-OWL API [15] to implement the relationship and policy ontologies (RelBOSS ontology KB). Once we expressed the set of polices that apply to a system as a knowledge base, run time reasoning can be performed for access control decision. During the 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 [16] for executing the SWRL rules. In particular, the query language SQWRL [14], which is based on OWL and SWRL, is adopted to process service access requests. Due to space limitation, the complete description of the prototype can not be included in this paper.

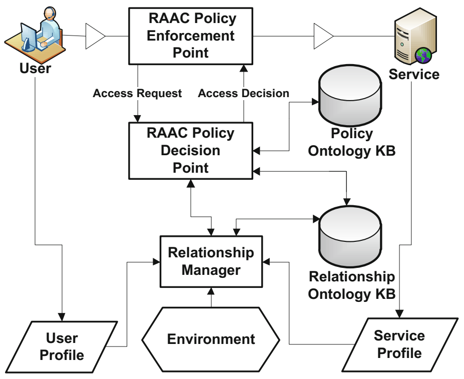


Fig. 4. Our Prototype Architecture

5.2 Developing a RAAC Application for Healthcare

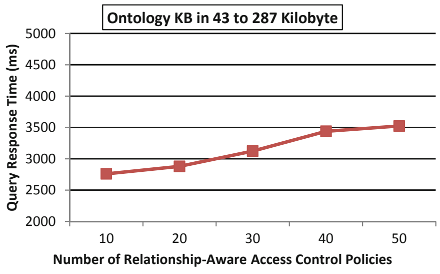
Using the prototype, we develop a relationship-aware access control application in the domain of electronic health records management system, and provide a healthcare case study (i.e., test-scenario based implementation). The main goal that we aim with this application is to access different medical records of patients based on the relevant relationship context information.

Case Study. Consider our application scenario, where Jane, by playing an emergency doctor (ED) role, wants to access the requested service *writeEMR()* (i.e., the write access to the emergency medical records (EMR) of patient Bob). Our *Relationship Ontology* captures the relevant relationship context information based on the currently available information and relationship specification rules. Our *Policy Ontology* captures the relevant RAAC policy. Based on this information, the framework returns the RAAC decision, i.e., Jane's service access request is ***Granted*** (see an access query in Table 5 and result in Table 6), because the ontology captures relevant relationship context information, and satisfies a RAAC policy which is stored in the policy base (ontology KB).

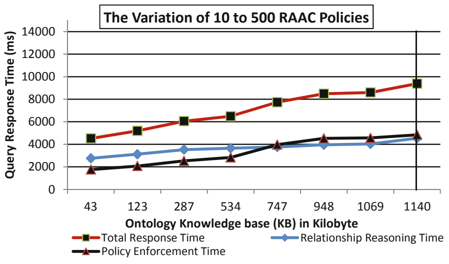
5.3 Experimentation

With the goal of evaluating the runtime performance of the various stages of our prototype framework, we have conducted a range 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. We have measured the *end-to-end query response time* (T) of the different aspects of *access request processing*, i.e., the time taken from the arrival of the user's service access request to the end of its execution (the sum of *(i) relationship context reasoning time* and *(ii) RAAC policy enforcement time*). The first measure indicates how long it took to infer the relationship context



(a) Policies versus Response time



(b) KB Size versus Response time

Fig. 5. Query Response Time Over Different Numbers/Size of Policies/KB

information (by capturing the currently available context information). The second measure indicates how long it took to process a user's service access request (by incorporating the inferred relationship context into the access control process and making relationship-aware access control decision). We separate the time delay for loading the relationship ontology and policy ontology from the access request processing time. The ontology loading occurs when the system runs the first time. In general, the most important parameter here is *the access request processing time*, which is of more interest or significance than the ontology loading time.

Experimental Results and Analysis. In the first test, we have evaluated the query response time when the number of access control policies increased. First, we have selected 10 policies with respect to 10 different health professional roles: *General Practitioner*, *Emergency Doctor*, *Registered Nurse*, etc [17]. We have varied the number of policies up to 50 with 50 health professional roles. Each of these variations is executed 10 times. The average value of the 10 execution runs is used for the analysis (see Figure 5(a)). The test results show that the query response time (T) increases when the number of policies increases. It varies between 2.7 and 3.5 seconds for the variation of 10 to 50 relationship-aware access control (RAAC) policies. In Figure 5(a), we can see that the query response time seems to be linear. Overall, the system performance is acceptable.

In the second test, we have again evaluated the response time (relationship reasoning time and policy enforcement time) over various size of the ontology knowledge base (KB). We have varied the number of policies up to 500 with respect to 138 different health professionals [17] (i.e., 138 roles). To measure the query response time, we have run the experiment 10 times. The results in Figure 5(b) present the average response time over various size of ontology KB. The computational overhead increases at a linear rate (note that the scales are not linear in the horizontal and vertical axises). At the point of the KB being 1140 kilobytes (where we have specified 500 policies for the 138 health professional roles), it takes approximately 9 seconds to process the access request.

In the last test, we have focused on measuring the ontology loading time (the relationship ontology and policy ontology) of our prototype with the increasing size of KB. The test results in Figure 6 show that the ontology loading time in increasing the size of the KB varies from 9 to 21 seconds approximately. Similar to previous test, in Figure 6, the scales are not linear in the horizontal and vertical axises. This results do not have impact on the end-to-end query response time (T) as ontology loading is only done at the start of the system.

In general, the query response time increases with the increasing number of access control policies (over various size of the ontology KB). Although, this seems a little expensive, at the point where we have modeled 500 relationship-aware access control policies for the 138 health professional roles (see Figure 5(b)), the performance can be improved by using more powerful machine.

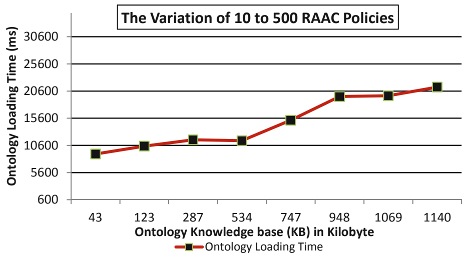


Fig. 6. KB Size versus Ontology Loading Time

6 Related Work and Discussion

Considering research in the field of relationship information, in this section we briefly highlight several existing research works. Some works integrate the *person-centric* relationship information and other works incorporate the *location-centric* relationship information in the access control processes.

Friend of a Friend (FOAF) [18] is one of the most used semantic web ontologies, which describes persons, using a set of properties that provide information about a person such as *name*, *gender*, *email*, and *relationship* with other persons. While FOAF considers only the generic relationships between two people, corresponding to the ones modeled by the *foaf:knows* property, more specialized interpersonal relationships describing the interconnection between two people at a fine-grained level (e.g., the relationship between the medical practitioner and patient is a “EmergencyDoctor-Patient” with “strong” strength) is an important aspect for relationship-aware access control applications. Some other research works have attempted to adopt and extend the FOAF ontology for deriving social relationships based upon the user’s daily communication with other people [19][20]. They extend the *foaf:knows* property with sub-properties such as *NeighborOf* and *ColleagueOf*. Same as basic FOAF model, these approaches do not allow the specification of different relationship context information such as the different granularity levels and strengths associated with a relationship. On the other hand, Toninelli et al [21] adopt the FOAF ontology, representing the types of resources and the relationships between them, in order to develop their Yarta middleware. They only consider the type of the relationships (e.g., *isMember(Agent,Group)*). A major difference of our relationship model with respect to these FOAF and FOAF-extended models is that, we not only consider the *person-centric* relationship (relationship type) but also consider the different *granularity levels* and *strengths* of that interpersonal relationship based on the currently available context information.

Some recent access control research works (e.g., [1], [2], [7], [8]) have integrated the *person-centric* relationship information into policy specifications, each of them having different origins, pursuing different goals and often, by nature, being highly domain-specific. Carminati et al [1] have proposed an access control approach, which allows the specification of access control policies for online resources in order to model various aspects of online social networks. They propose access control policies that depend on the relationship between users (e.g., *friend*, *friend-of-friend*), and relationships between user and resource. Zhang et al [7] have proposed an approach named relation-based access control (RelBAC), in which the access permissions are formalized as binary relations between subjects and objects. They consider subject, object and permission as the compulsory access control components. Fong et al [8] have presented a relationship-based access control approach named ReBAC. They consider the relationships between individual users (e.g., *professional association*) in the access control policies. Squicciarini et al [2] have proposed a privacy protection mechanism for social networks named PriMa, which specifies access control policies for users profile information. PriMa considers the relationship between resource owner and requester (e.g., *common friend*) in the access control processes.

These relationship-aware approaches consider the expression of access control policies in terms of the *person-centric* or interpersonal relationships to control access to information resources or software services. However, these approaches do not address and model the *different granularity levels* and *strengths* of the interpersonal relationship. Furthermore, these existing approaches are domain-specific and they are hard to be implemented for various access control applications in today's dynamic context-aware environments. Different from these approaches, we, therefore, introduce in this paper a *Relationship Ontology*, to dynamically identify much richer forms of relationships among relevant entities with different granularity levels and strengths of that relationship, and a *RAAC Policy Ontology*, to provide relationship-aware access to information resources based on the inferred relationship context information.

Several other access control research works (e.g., [22], [23]) have incorporated the *location-centric* relationship information into policy specifications. Ardagna et al [22] have proposed an access control approach for location-based services. They integrate three types of location-based conditions into their policy model, position-based, movement-based and interaction-based conditions. Tarameshloo and Fong [23] have proposed an access control approach for Geo-social computing systems, which allows users to declare their current locations and uses these declared locations to make access control decisions. They propose access control policies that depend on the location declarations/claims by users, e.g., *nearby* users can allow access. These access control approaches only consider the *location-centric* relationships for making access control decisions. However, these approaches do not address and model the *person-centric* relationships.

7 Conclusion and Future Work

Dynamic and context-aware environments require new access control frameworks, changing the focus of access control paradigm from identity/role-based to relationship-based. Existing relationship-aware access control (RAAC) approaches integrate the relationship dimension into policy specifications. However, these approaches do not provide adequate supports for identifying the different granularity levels and strengths of the relationship. One of the main contributions of this paper is the formal *RAAC Model* for specifying the relevant relationship information with different granularity levels and strengths, and the corresponding relationship-aware access control policies. Another contribution of this paper is an ontology-based framework, *RelBOSS*, in order to realize RAAC model using OWL and SWRL. The RelBOSS framework defines and enforces the relationship-aware access control policies by incorporating the relevant relationship context information. We have demonstrated the practical applicability of our framework through the implementation of a software prototype in the healthcare domain and presented a case study. The case study shows that our framework is able to capture relevant relationship context information at runtime and invoke software services in a relationship-aware manner. In addition, the experimental results show the feasibility of our framework.

Our framework can be extended to model and capture other types of relationships (e.g., a fine-grained “location-centric” relationship can be identified by using the spatial relations between different entities, i.e., *nearby* or *co-located* persons). We plan to apply our framework/system to the real world and further examine performance and optimisation issues 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.

References

1. Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M., Thuraisingham, B.: A semantic web based framework for social network access control. In: SACMAT, pp. 177–186 (2009)
2. Squicciarini, A., Paci, F., Sundareswaran, S.: Prima: an effective privacy protection mechanism for social networks. In: ASIACCS, pp. 320–323 (2010)
3. Weiser, M.: Some computer science issues in ubiquitous computing. Commun. ACM 36(7), 75–84 (1993)
4. 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)
5. Dey, A.K.: Understanding and using context. Personal Ubiquitous Computing 5(1), 4–7 (2001)
6. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using owl. In: PerCom Workshops, pp. 18–22 (2004)
7. Zhang, R., Giunchiglia, F., Crispo, B., Song, L.: Relation-based access control: An access control model for context-aware computing environment. Wirel. Pers. Commun. 55(1), 5–17 (2010)
8. Fong, P.W., Siahaan, I.: Relationship-based access control policies and their policy languages. In: SACMAT, pp. 51–60 (2011)
9. 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)

10. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29, 38–47 (1996)
11. OWL (February 2014), `http://www.w3.org/2007/owl/`
12. SWRL (February 2014), `http://www.w3.org/submission/swrl/`
13. Moses, T.: Extensible access control markup language (xacml). In: OASIS Standard (2005), `https://www.oasis-open.org/committees/xacml/`
14. O'Connor, M.J., Das, A.K.: Sqwrl: A query language for owl. In: OWLED (2009)
15. Protégé-OWL (February 2014), `http://protege.stanford.edu/`
16. JESS: Jess rule engine (February 2014), `http://herzberg.ca.sandia.gov/`
17. ASCO: Australian standard classification of occupations, health professionals (February 2014), `http://www.abs.gov.au/`
18. FOAF: Friend of a friend, `http://xmlns.com/foaf/spec/20100809.html` (February 2014)
19. RELATIONSHIP: A vocabulary for describing relationships between people (February 2014), `http://vocab.org/relationship/`
20. Devlic, A., Reichle, R., Wagner, M., Pinheiro, M.K., Vanrompay, Y., Berbers, Y., Valla, M.: Context inference of users' social relationships and distributed policy management. In: PerCom Workshops, pp. 1–8 (2009)
21. Toninelli, A., Pathak, A., Issarny, V.: Yarta: A middleware for managing mobile social ecosystems. In: GPC, pp. 209–220 (2011)
22. Ardagna, C.A., Cremonini, M., di Vimercati, S.D.C., Samarati, P.: Access control in location-based services. In: Privacy in Location-Based Applications, pp. 106–126 (2009)
23. Tarameshloo, E., Fong, P.W.L.: Access control models for geo-social computing systems. In: SACMAT, pp. 115–126 (2014)