Journal of Computer Security 21 (2013) 1–39
DOI 10.3233/JCS-2012-0454
IOS Press

Behavior-based access control for distributed healthcare systems

Mohammad H. Yarmand [a], Kamran Sartipi [b] and Douglas G. Down [a,*]

[a] *Department of Computing and Software, McMaster University, Hamilton, ON, Canada*
E-mails: {yarmanmh, downd}@mcmaster.ca
[b] *Faculty of Engineering and Applied Science, University of Ontario Institute of Technology, Oshawa, ON, Canada*
E-mail: kamran.sartipi@uoit.ca

Sensitivity of clinical data and strict rules regarding data sharing have caused privacy and security to be critical requirements for using patient profiles in distributed healthcare systems. The amalgamation of new information technology with traditional healthcare workflows for sharing patient profiles has made the whole system vulnerable to privacy and security breaches. Standardization organizations are developing specifications to satisfy the required privacy and security requirements. In this paper we present a novel access control model compliant with healthcare standards based on a framework designed for data and service interoperability in the healthcare domain. The proposed model for customizable access control captures the dynamic behavior of the user and determines access rights accordingly.

The model is generic and flexible in the sense that an access control engine dynamically receives security effective parameters from the subject user, and identifies the privilege level in accessing data using different specialized components within the engine. Standard data representation formats and ontologies are used to make the model compatible with different healthcare systems. The access control engine employs an approach to follow the user's behavior and navigates among engine components to provide the user's privilege to access a resource. A simulation environment is implemented to evaluate and test the proposed model.

Keywords: Access control, healthcare standards, user behavior, privacy specification, interoperability

1. Introduction

The cost of healthcare in developed countries is rising rapidly due to the population's expectation for a higher quality of health service including: broad accessibility, customizability, cost efficiency, and most importantly reliability and security. Also, integrating computer applications within the healthcare domain has caused health professionals to embrace quickly growing distributed information and communication technologies. The new proposals for national and international healthcare standardization meet most of these requirements.

[*] Corresponding author: Douglas G. Down, Department of Computing and Software, McMaster University, Hamilton, ON, Canada L8S 4K1. E-mail: downd@mcmaster.ca.

While solving the problem of interoperability among heterogeneous systems, these proposals introduce many security and privacy issues, as natural consequences of providing customizable services. Regarding confidentiality, integrity and availability requirements of patient data, a major concern is to avoid disclosure of these data to unqualified users. Access by unauthorized users to patient data may result in misdiagnosis, delays in treatment, or mistreatment. Other consequences may include financial problems such as denial of insurance coverage and loss of job opportunities [36].

Authentication and authorization methods at inter-/intra-organizational levels should be employed to provide the required security. In this context, several methods have been proposed namely role-based [20,45,54,63], team-based [21], attribute-based [16], content-based [22], scenario-based [43,48], situation-aware [61], context-aware [1,4,33,63], and context sensitive [34] access control methods. Only a few consider the problem in distributed systems [4,33,45,61]. Moreover, most access control methods deal only with static systems. However, dynamism and configurability are two requirements of models for distributed systems [33,45,61,62]. There are a small number of approaches that propose models adherent to healthcare standards [6,35,42].

In very large scale distributed systems, the integrity, flexibility, generality, and robustness of the model becomes critical. Due to the high complexity of these systems, any errors in the operation of the access control model can be difficult to detect. In order to apply a generic model to healthcare systems, the corresponding specific requirements should be considered. They define the architecture, clinical data model, and transportation protocol for integration.

Given the above issues, the specific problem which is targeted in this paper is: *Propose an access control model for distributed healthcare systems that is interoperable with various organizational data formats and security rules, so that the specific healthcare requirements and standards are satisfied.*

A novel access control model is designed which is dynamic, system independent, and configurable. The model uses interactions and data flows between different types of system entities (such as users, roles, and resources) to adjust access control decisions. These interactions are recorded and analyzed to extract the behavior of the entities. Healthcare standards obligations about security, architectural specifications, data modeling, and system integration are carefully studied to make the access control model compatible with these standards. Access control decisions are based on constraints defined for roles, teams, contexts, and delegation rules.

The contributions of this paper are as follows. (1) An access control model is provided that satisfies a comprehensive set of healthcare requirements and conforms to the healthcare standards dealing with access control. (2) The model is formally defined. In particular the concept of *user behavior* is introduced to configure the access control model based on user requirements. (3) To enhance interoperability, a common data model is proposed. (4) Implementation issues are discussed. The model is tested with simulated data. It should be highlighted that the contributions listed here are not unrelated. In particular the *user behavior* concept is designed in a way that refines healthcare standards requirements. A preliminary version of this work can be found in [60].

2. Healthcare standards

The healthcare industry has many organizations developing specifications and standards to support information exchange and system integration. These specifications are used to provide interoperability for a wide spectrum of healthcare applications. Here the security (particularly access control) specifications of these standards are outlined.

HL7. The Health Level 7 Security Technical Committee (HL7 STC) suggests using scenario driven access control based on dividing scenarios into work profiles and tasks. HL7 STC has released a list of healthcare scenarios that encompass security issues [28], and has introduced a hierarchy of healthcare roles and their access privileges [30]. These specifications should be used together to determine the access rights of a role for completing an HL7 scenario [29].

Canada Health Infoway. The Privacy and Security Architecture (PSA) group has not yet suggested an architecture to serve security requirements but it has offered two useful documents: *EHR Privacy and Security Requirements* [9] which discusses general security requirements in the healthcare domain and refers to data usage restrictions under privacy rules; and *EHRi Privacy and Security Conceptual Architecture* [10] which expresses the specifications of the communication and common services. PSA suggests the use of role and work group based and discretionary access control.

HIPAA. The Health Insurance Portability and Accountability Act (HIPAA) [24] provides a list of security and privacy suggestions and legal requirements. The access control requirements suggested by HIPAA [25] contain unique user identification, emergency access procedure, automatic log-off, and encryption and decryption.

IHE. Integrating the Healthcare Enterprise (IHE) [37] is an initiative designed to promote standard-based methods of data integration in healthcare. The technical security specification of IHE [39] includes authorization (role management, user/role certificate management, assertion rights, delegation rights, validity time), node authentication, information access, information integrity (document update and maintenance policy, document digital signature policy, folder policy), ethics, audit trail, and risk analysis.

Requirements. We have extracted a number of healthcare specific requirements from the standardization documents. In the following we list and briefly explain them.

- “emergency access”: in emergency or life-threatening conditions, bypass access must be available so that security restrictions do not prevent offering essential healthcare services. In an “emergency access” case, the user requests a permission he does not already have [8,26].
- “consent”: is when the patient delegates part of his authorities or limits access. Consent can be implicit (e.g., when a patient visits a physician) or explicit (e.g., when a patient allows others to make decisions on his behalf) [9,38].
- “workgroup access”: supports assigning the patient to a care group such as geographical, organizational, or circle-of-care work groups [9].
- “audit trail/monitoring”: all access requests together with supporting reasons should be logged and the record owner should be notified. Monitoring should be developed jointly with audit logging [9,12,38].
- “context awareness”: permissions of care givers are altered based on the context they are working in, represented as context constraints. Contextual properties include separation of duties, time-dependency, mutual exclusivity, cardinality, and location [26,31,38].

3. Access control model

In this section we define the basic concepts that are employed in the proposed model. In Section 3.1 we explain why a new access control approach is required. In Section 3.2, the *user behavior* concept, employed to visualize user activities, is informally defined. Different types of behaviors and their usage in making access control decisions are described. In Section 3.3, we formally define the basic concepts of the security domain and also the *user behavior*. Examples from the healthcare domain are used to further explain the formal definitions.

3.1. Motivating our approach

As will be discussed in Section 7, several access control models have been proposed for the healthcare domain. However very few of them consider the standards requirements discussed in Section 2. Therefore we intend to propose an access control method to target these requirements. More specifically this concept is used to (i) express access history related constraints (identified as a requirement in [19,23]) (ii) formalize audit trails; (iii) monitor access related activities (continuously logging system, regularly reviewing logs, reporting every access to a patient record, reporting security incidents are part of requirements described in [9,12,38]); (iv) detect patterns of misuse (stated as requirement 46 in [9]).

Sample rules that can be expressed with the new concept (in Section 3.2) are:

- A user u_i in a day, must take roles in the following order: r_k, r_l, r_h.

- A user u_i can not access a resource res_1 more than n_1 times between times t_1 and t_2.
- If a user u_i is in location loc_e, he can only join teams tm_x and tm_y.
- A user u_i cannot make access requests from locations loc_e, loc_f in less than t_m time units.

Another aspect of an access control model is its administration. To the best of our knowledge, there has been little effort toward improving access control models based on dynamic access requirements and characteristics of the target environment. In order to extract these requirements, it is necessary to visualize the activities of users of the system. The usage pattern can be analyzed to refine access privileges and provide suggestions for system configuration (in the form of filtering attributes domains, explained in Section 4.2.6). For example, by monitoring denied access requests, it might be determined that the governing policy rule should be modified to allow a particular access.

3.2. Action and behavior

In order to capture the behavior of a user, we need to record a set of run time contexts, whenever the user interacts with the system. We call each interaction an *action* and represent it by a tuple, namely an *action tuple*. An *action* is any interaction which either requests a resource or changes the level of access privilege. The *action tuple* is composed of several *attributes*, as follows:

> Action = <*User, Role, User Location, Server Location, Time of Day, Team, Delegation, Requested Profile Status, Service Invocation Type, Requested Data Type, Login/Logout Event, Emergency*>

where *User* is the user identification; *Role* is the user security role; *User Location* is the current location of the user; *Server Location* is where the requested resource is located; *Team* refers to the team to which the user currently belongs; *Delegation* explains the access rights given or taken by the delegation rules or consents; *Requested Profile Status* refers to attributes of the *Resource Context* class (explained in Section 4.1), for the requested profile; *Requested Data Type* refers to the clinical data type that the user has requested; *Service Invocation Type* is the type of service requested; *Login/Logout Event* records login and logout events; *Emergency* declares an emergency situation. The value that each of these attributes takes is called the attribute value.

A *Behavior* is defined as a sequence of actions that can be manifested in the following forms: A record of a sequence of actions performed during a specified time interval, e.g., during the last five hours, a day, a month, etc. Each observation might include a portion of the attributes of the action tuple. For example, in one day different tasks performed by a person are recorded as action tuples and their collective effect is considered as behavior.

Whenever an *attribute* value of the *action tuple* of a user changes, a new tuple is recorded. A new tuple may be recorded even if the user has not requested access to a resource. For example when a user joins a team, his privileges change and therefore a new tuple should be recorded even if the user does not request access to a resource.

3.2.1. Behavior based access control

The concept of user behavior can be used in different ways to make access control decisions. Here we introduce three different approaches.

Single action represents a single action tuple. Given a single action tuple we choose one of the action attributes as a key attribute and use it to constrain the domains of other attributes. If *Role* is the key attribute, the domain of other attributes would be limited based on *Role*. In order to determine how attribute domains are filtered according to the *Role* value, general clinical guidelines or hospital policies defined for that specific role can be used. If *User* is the key attribute, the domains of other attributes would be limited based on that specific user. This makes our model dynamic and flexible in the sense that attribute domains are loaded for the specific user. In order to determine how the domains are filtered according to a specific user, the history of action tuples recorded for that user is analyzed to extract associated domain values.

Daily behavior consists of a sequence of action tuples recorded in one day for a given user. Some access control processes require more than a single tuple to be able to make an access control decision. Examples are: log-in/out pattern; location proximity of consecutive requests; requested profile category and instance diversity; access request frequency; sequence of service invocation; policy rules explicitly defined over time such as access restrictions of a user on particular days.

Having defined the *user behavior*, we define *expected behavior* as follows: the behavior that users are expected to follow which are entered as rules by the system security administrator. The *user behavior* concept is sufficiently rich to allow the enforcement of different kinds of rules to express the *expected behavior*. Some of the possible rules are:

- *Ordering*: requiring that values occur in a certain sequence; limiting the number of values appearing between specific items of a sequence.
- *Timing*: limiting the occurrence frequency of a certain value; asserting time interval constraints between occurrences of certain values in a sequence; defining absolute time constraints such as enforcing part of a sequence to occur before a certain time.
- *Association*: enforcing that the occurrence of val_1 for att_1 should result in val_2 for att_2 (e.g. limiting value scope based on location).
- *Combination* of the above items: examples are statements such as detecting a particular order for att_1 should result in val_2 for att_2; occurrence of a particular order for att_1 before a particular time should result in a particular sequence for att_2.

A third concept, based on *user behavior*, called *common behavior* is defined as: the behavior of a user analyzed and extracted from the user behavior history. The system is capable of analyzing behavior histories to extract the different behavior criteria defined in the *expected behavior*. One of the approaches of this analysis is extracting the sequence of attribute values that an attribute takes most of the time, in the action tuples of a user (this approach is specified in Algorithm 2 of Section 4). Ideally, when users act according to their *expected behaviors*, the distance between *common* and *expected behaviors* should be minimal. Also *common behavior* can be used to adjust *expected behavior* rules, in cases where the distance is significant. In order to come up with the *expected behaviors* the administrator can use organization specific guidelines, based on the analysis of organization workflows. For example in the nursing domain, the Canadian Nurses Association (CNA) [13] provides guidelines for Canadian nurses. They released a document called *Advanced Nursing Practice – A National Framework* [15] which describes competencies and regulations for nurses.

The last topic we discuss in this subsection is handling access requests in emergency situations. "Emergency access" offers broader privileges to care givers by extending their existing privileges to access protected health information when timely access is needed to prevent harm or risk to life. Although no standard procedure exists on how to handle these requests, suggestions have been made by healthcare standards initiatives (see for example [8,26]). During these situations, emphasis should be placed on providing needed information instead of trying to enforce very tight security constraints.

When an emergency access request is made (assuming the user's current privilege does not include the requested permission), the access is granted unless the patient's consent directives state otherwise. The user is warned about being closely monitored and that he is subject to legal penalties in case he is committing an access violation. At the same time a notification is sent to the security officer to closely monitor the user's activities. In this sense emergency access works closely with the audit trail.

3.3. Formal definitions

In this section we first introduce the high level specification of our model using an algorithm and corresponding primitive types, relations, and examples from the healthcare domain.

Algorithm 1 accepts an access request as input and employs the relations between sessions, roles, teams, and delegations together with different constraints on actions and behaviors to find a matching access control policy to make the access control decision. The consistency among these relations and constraints is ensured by the order in which the algorithm treats them. For example if consent directives deny an access while an emergency situation is reported, consent directives are followed and the access is denied (see Lines 1–9).

Algorithm 1. Access Control Mechanism ($<u_i, p_g, b(u_i), a> \in AR$)

```
1:  if ($\exists <u_j, p_g, u_i, t,$ false$> \in D.$ current time $\leqslant t$) then
2:      Deny Access
3:      goto exit
4:  end if
5:  if ($a.e$ is true) then
6:      Grant Access
7:      Notify Security Administrator
8:      goto exit
9:  end if
10: $ses := SessionUser^{-1}(u_i)$
11: $RArray := SessionRole(ses) \cup SessionTeamRole(ses)$
12: for ($r \in RArray$) do
13:     $PArray := PArray \cup \{p \mid <p, r> \in PR\}$
14: end for
15: if $p_g \in PArray$ then
16:     if (($\exists\, acp_1 = <u_i, p_f, c,$ true$> \in$ ACP. $p_g \subseteq p_f \wedge$
        $c$ is true by checking $AC \wedge BC \wedge LC$) $\wedge$
        ($\nexists\, acp_2 = <u_i, p'_f, c',$ false$> \in$ ACP. $p_g \subseteq p'_f \wedge c'$ is true $\wedge$
        $(acp_2, acp_1) \in PP$)) then
17:         Grant Access
18:         goto exit
19:     end if
20: end if
21: if ($\exists <u_j, p_g, u_i, t,$ true$> \in D.$ current time $\leqslant t$
      $\wedge \exists <u_j, p_g, c'> \in$ ACP. $c'$ is true) then
22:     Grant Access
23: else
24:     Deny Access
25: end if
26: exit: Update $b(u_i)$ and Record Access Decision
```

Table 1

Variables of Algorithm 1

Variables	Explanation	Variables	Explanation
u_i, u_j	User ($\in U$)	*RArray*	Active (team) roles in session *ses*
p_g, p_f, p'_g, p'_f	Requested permission ($\in P$)	*a.e.*	Emergency situation declaration ($\in E$)
$b(u_i)$	Behavior of user u_i ($\in B$)	*PArray*	Array of active permissions gained from applying *PR* to active roles
a	Action tuple ($\in A$)		
ses	Session ($\in S$)	c, c'	Conjunction of constraints

Table 1 defines the variables used in Algorithm 1. Lines 1–4 of the algorithm state that if p_g includes accessing a resource which has been blocked for user u_i by the resource's owner, the access request is denied. In the healthcare setting, patient consent directives can be enforced by these lines. Lines 5–9 address an emergency access. In lines 10–20 first a session *ses* dedicated to user u_i is retrieved and the

Table 2

List of primitive types and relations

Notation	Name	Notation	Name
A	The action tuple relation	*P*	The Permission relation
AC	Action constraint expression	*PP*	Policy priority
ACP	Access control policy relation	*PR*	Permission-Role assignment
AP	The access pattern of a resource	*R*	The set of roles
AR	Access request relation	*RC*	Resource context relation
B	A mapping of user-action sequence	*Res*	The set of resources
BC	Behavior constraint expression	*RT*	Role-Team assignment
CntCon	Counting constraint	*S*	The set of sessions
ComBeh	Common behavior of a user	*SessionRole*	A mapping of session-roles
Constraint	$AC \wedge BC \wedge LC$	*SessionTeamRole*	A mapping of session-team-roles
D	The set of delegations	*SessionUser*	A mapping of session-user
DaiBeh	Daily behavior of a user	*T*	Time instances
DT	The set of a domain data types	*Tm*	The set of teams
L	The set of locations	*U*	The set of users
LC	Logical constraint expression	*UR*	User-Role assignment
OrdCon	Ordering constraint	*URT*	User-Role-Team assignment

active roles are assigned to the *RArray* variable. Then *PermissionRoleAssignment* is applied on *RArray* to load the active permissions in the *PArray* variable. Afterwards if the requested permission p_g belongs to *PArray*, the list of *AccessControlPolicies* is browsed to check if an access control policy exists that satisfies the access request and the action, behavior, and logical constraints. If such an access control policy is found, access is granted if no higher priority access control policy revokes the requested privilege. In lines 21–25 we check if permission p_g is given to user u_i via a delegation relation. Finally, line 26 updates $b(u_i)$ and records the access decision made.

We now discuss the details of Algorithm 1. A number of primitive types are introduced to define the collection of *sets* required to provide a formal definition of the model. These primitive types are as follows: users *U*, roles *R*, resources *Res*, sessions *S*, teams *Tm*, login/logouts *Log*, data types *DT*, emergency situations $E = \{true, false\}$, and purposes of use *Pr*.

We now use these primitive types to express entities and relations for the proposed access control model. These relations are finally employed to state an access request, an access control policy, and the access control method. Table 2 provides a summary of the notations used.

Session

SessionUser: $S \mapsto U$, is a function mapping each session to a single user. The user is constant for a session lifetime.

SessionRole: $S \mapsto R$, is a function mapping each session to an active role. The set of active permissions for a session can be inferred by using the composition of the *UR* and *PR* relations.

SessionTeamRole: $S \mapsto RT$, is a function mapping each session to an active team-role pair.

The above relations can be defined in such a way as to allow more than a single role per session. However this would violate healthcare domain requirements and can also cause policy conflict [38]. Hierarchal design for roles can be used to overcome this restriction (of one role per session). Here we assume the user can have a dedicated role and a role from its participating team.

Assignments

Permission: $P \subseteq OP \times Res \times Pr$, where *OP* defines the operations that can be performed on resources (e.g. $\{$*read, append, delete, update*$\}$) for the specified purpose. For example $P = \{p_1, p_2\}$ where $p_1 = \langle$*update, JaneAccount,AccountUpdate*$\rangle$, $p_2 = \langle$*update, JaneAccount, NewDietOrder*$\rangle$.

Adding 'purpose of use' to the definition of the **Permission** relation has a number of justifications in the healthcare domain [29,32,38]. Permissions are assigned to certain scenarios that describe use purpose. Further patient consent might specify different privileges for different access purposes of a given requester. In case of inter-organization access control, one way to resolve policy conflicts is to map use purposes among organizations.

UserRoleAssignment: $UR \subseteq U \times R$, defines the roles that a user can have. For example $UR = \{\langle Jane, user\rangle, \langle Jane, nurse\rangle, \langle Nero, patient\rangle, \langle Nancy, patient\rangle\}$.

PermissionRoleAssignment: $PR \subseteq P \times R$, defines the permissions a role can have. For example $PR = \{\langle p_1, nurse\rangle, \langle p_2, user\rangle\}$.

RoleTeamAssignment: $RT \subseteq R \times Tm$, defines the roles that can participate in a team. For example $RT = \{rt_1\}$ where $rt_1 = \langle nurse, diabeticNursingTeam\rangle$. The workgroup requirement stated earlier can be formalized with this relation.

UserRoleTeamAssignment: $URT \subseteq U \times RT$, defines the set of users that participate in a team and their roles in the team. For example $URT = \{\langle Jane, rt_1\rangle\}$.

Delegation: $D \subseteq U \times P \times U \times T \times GR$, where *GR* has a boolean value, defines a function that a delegator uses to grant/revoke a permission to a delegatee for a specified time period. For example Jane allows Nero to update Jane's profile in the next hour $D = \{\langle Jane, p_1, Nero,\ 2010.12.16\text{-}02{:}51,\ true\rangle\}$ given the delegation happens at 2010.12.16-01:51. Patient consent directives can be expressed with this relation. To grant/revoke an access (e.g. to a family member or a physician) the value of *GR* is set to *true/false*. A permanent/temporary delegation sets the value of T to *infinity/sometime in the future*. A specific application of this relation in the healthcare domain is when a care giver asks a colleague for a second opinion on a medical case [63].

Action and behavior

Action: $A \subseteq U \times R \times L \times T \times \mathit{Tm} \times D \times \mathit{RC} \times \mathit{Res} \times \mathit{OP} \times \mathit{DT} \times \mathit{Log} \times E$, is the action tuple defined in Section 3.2. For example $a_1 = \langle$*Jane, nurse,* l_1, t_1, *diabeticNursingTeam, nil,* rc_1,*NancyProfile, append*, $\{$*diabeticInfo*$\}$, *nil*,*false*$\rangle$.

Behavior: $B : U \mapsto 2^A$, is a function mapping a user to a sequence of actions that composes the behavior of the given user u. For example B(*Jane*) $= [a_1, a_2, a_3]$.

DailyBehavior: $\mathit{DaiBeh} : U \times \mathit{ATT} \mapsto 2^{att_{val}}$, is a function returning the sequence of attribute values (att_{val}) of the attribute $att \in ATT$ in the behavior records of user u, where *ATT* is the set of action tuple attributes. This function describes the *daily behavior* (see Section 3.2). For example *DaiBeh*(*Jane, role*) $=$ $\{$*user, nurse, nurse, researcher*$\}$.

CommonBehavior: $\mathit{ComBeh} : U \times \mathit{ATT} \mapsto 2^{att_{val}}$, is a function returning the sequence of attribute values (att_{val}) of the attribute $att \in ATT$ by analyzing the behavior records of user u. This function describes the *common behavior* (see Section 3.2).

Access request and policy

AccessControlPolicy: $\mathit{ACP} \subseteq \mathit{Subject} \times P \times \mathit{Constraint} \times \mathit{GR}$, where *Subject* is a user or a role. *ACP* is a relation which defines the constraint that should be satisfied so that a permission is granted/revoked to a subject. For example $acp_1 = \langle \mathit{Jane}, p_1, \mathit{constraint}_1, \mathit{true} \rangle$.

AccessRequest: $\mathit{AR} \subseteq U \times \mathit{reqPerm} \times B(u) \times A$, where *reqPerm* is the requested permission. For example $ar_1 = \langle \mathit{Jane}, \langle \mathit{read, NeroProfile} \rangle, B(\mathit{Jane}), a_1 \rangle$.

PolicyPriority: $\mathit{PP} \subseteq \mathit{ACP} \times \mathit{ACP}$, is a relation defining priority among access control policies. $(acp_1, acp_2) \in PP$ assuming $acp_1 \cdot subject = acp_2 \cdot subject \wedge acp_1 \cdot GR \neq acp_2 \cdot GR$, $acp_1 \cdot constraint$ and $acp_2 \cdot constraint$ are satisfiable at the same time, and $acp_1 \cdot p \cup acp_2 \cdot p \neq \emptyset$, means acp_1 has higher priority than acp_2.

Constraints

ActionCondition $:= \lambda\ att\ att' \cdot att\ op\ att'$ (λ is lambda notation) where *att* refers to the values of an attribute of the action tuple, *op* is a logical operator in the set

$\{>, \geqslant, <, \leqslant, \neq, =\}$, and att' is a value of the same type as *att*. The Boolean value resulted from applying operation *op* to values *att* and att' is returned as the output.

ActionConstraint: *AC*, is a predicate consisting of disjunction and/or conjunction of *ActionCondition* expressions. For example $ac_1 =$ (team $=$ *diabeticNursingTeam*) $\wedge$ (data type $\neq$ *confidential*). Simple context related constraints can be stated here.
BehaviorConstraint: *BC*, is defined as the result of checking different kinds of *expected behavior* (i.e. *OrdCon, AssCon, TimCon, ComCon* and *ConCon* described in the following). If a given *DaiBeh* satisfies all of the constraints defined by these rules, *BC* is *true*; otherwise *BC* is *false*. We formally define *OrdCon* and *CntCon* and move the rest to Appendix A.

OrderConstraint: $OrdCon : U \times ATT \mapsto 2^{att_{val}} \times Mandatory \times MinLength$, is a function returning the sequence of attribute values (att_{val}) (of a minimum length N and mandatory/optional occurrence of attribute values is the sequence based on *Mandatory*) of the attribute $att \in ATT$ in the behavior records of user u, where $Mandatory = \{true, false\}$ and $N \in \mathbb{N}$. For example *OrdCon*(*Jane*, *requested profile*) $=$ [⟨*JaneAccount, true, 3*⟩, ⟨*SaraProfile, true, 3*⟩, ⟨*NeroProfile, false, 3*⟩, ⟨*NancyProfile, true, 3*⟩].

CountConstraint: $CntCon : U \times ATT \mapsto N \times T \times T$, is a function expressing the valid number of times $n \in N$ that a user u can access $att \in ATT$ in the time interval $[t_1, t_2]$ where $t_1, t_2 \in T$. This relation targets counting constraints in the healthcare domain (see [19,23,57] for examples).

LogicalConstraint: *LC* is a logical expression that can not be expressed with either *AC* or *BC*. Constraints that use resource contexts are defined here. For example $lc_1 =$ *isMemberOfDiabetesDepartment*.

Constraint $:= LC \wedge AC \wedge BC$. For example $constraint_1 = lc_1 \wedge ac_1 \wedge bc_1$.

Context

Location: L, is a set of locations. For example $l_1 =$ *diabeticNursingStation*.

Time: T, represents time instances in the following format year.month.day-hour:minute). For example $t_1 =$ *2010.12.25-09:05*.

AccessPattern: $AP \subseteq 2^{L \times T}$, is a subset of location-time pairs (which is employed to record the location and time stamp of the users who access a resource). For example $ap_1 = \langle l_1, t_1 \rangle$.

ResourceContext: $RC \subseteq Res \times AP \times 2^U \times DT$, is a collection of attributes and contexts recorded for resources. For example $rc_1 = \langle$*JaneAccount*, ap_1, *Jane, reminder*$\rangle$.

4. Access control model architecture

In this section an architecture is introduced that supports the features of the proposed access control model. A common data model is created to provide interoperability for collecting the information that the architecture requires. The architecture specifies how to apply the definitions offered in Section 3.

4.1. Security effective parameters

In order to keep the model as general as possible, different security parameters should be both identified and captured in different environments. They are used as parameters for the privacy and security rules. These parameters are modeled in a class diagram shown in Fig. 1.

In order to establish interoperability and reusability, the relations between our class diagram and standard clinical data are defined. The interoperation between policy rules of two legacy systems can be facilitated through mapping the elements of their policy model to the model presented in Fig. 1. Our class diagram is connected to the HL7 Reference Information Model (RIM), an object model that is a representation of clinical data. A few classes of HL7 RIM have been used. The *service type* class (top right) represents a list of services that a user invokes; this list is mapped to Infoway storyboards and transactions of standard healthcare scenarios [11]. The *data type* class (top right) expresses the type of clinical data and is specified by the hierarchial object definitions from healthcare standards [32] (currently 100 types).

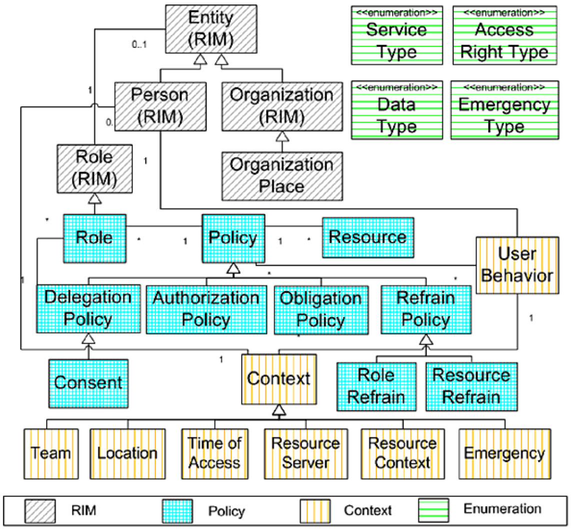


Fig. 1. Class diagram of the security effective parameters. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)

Moreover, resources are structured according to the HL7 Clinical Document Architecture (CDA) [27] to increase access granularity (e.g. observations, procedures, diagnostic images, financial transactions). The *access right type* class defines different operations specified by [32] (currently 26 types).

There are four categories of classes: (i) HL7 classes; (ii) context hierarchy classes; (iii) core security classes, and (iv) enumeration classes. We extend the policy classification offered by the Ponder project [17] to represent different policies. The remainder of this section explains the major classes.

The *policy* class regulates the conditions a user should meet, in order to access a resource. The policies are divided into four different categories. 'Authorization' policies define the actions that different roles are allowed to perform on resources. 'Delegation' policies determine temporal states under which users can delegate their access rights to other users. 'Refrain' policies revoke permission even if permission is given by other policies and are categorized as *role refrain* and *resource refrain* based on the target of their restriction. 'Obligation' policies specify the action (such as administrative actions) that must be performed when certain events occur.

Context Aware Systems (CAS) offer a number of benefits such as authorizing users based on their context, adjusting security levels automatically, and sharing services [3,40,41,49]. Healthcare security standards use contexts to express different constraints [26,31,38]. Context aware models define logical constraints over context and restrict the set of possible context configurations. These constraints are placed in the *policy* class to maintain model integrity. A major portion of the class diagram has been allocated to represent security related contexts derived from the *context* class. In the formal definitions in Section 3.3, these contexts are represented as action tuple attributes (*location* and *time* are conventionally shown as **context** in the formal definitions).

The *user behavior* class uses the *context* class and audit trails for resources and users to extract information to model the behavior of a user. The *resource context* class considers contexts over resources such as the access pattern made to a resource, the type of data the resource contains, and users who have previously accessed the resource.

4.2. Proposed architecture

An architecture is designed to use the concepts introduced so far and deliver the desired access control functionality. The architecture is shown in Fig. 2. The figure is divided into different layers that provide a high level categorization of different blocks (each box is called a block) of the model. The blocks of a layer perform a common major task.

An administrator configures the system dynamically by entering data through the *input* layer. This task tailors the generic model to a specific environment and provides interoperability and configurability. The model receives an access request and captures the context of the requesting user, checks for different policy rules, applies

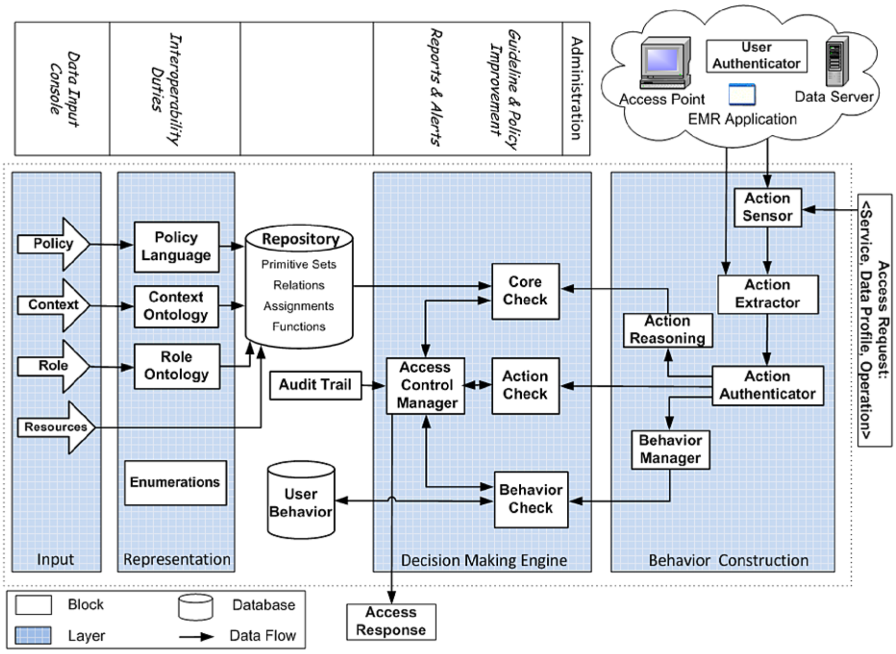


Fig. 2. The architecture of the proposed access control model. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)

the results of behavior based constraints and returns the final access decision. In this section all blocks of the model are introduced and their responsibilities and specifications are explained.

4.2.1. *Input*

The inputs are the same as the security effective parameters explained in Section 4.1.

4.2.2. *Representation*

In order to make the system interoperable and usable in different environments, we have to map input parameters (from the *input* layer) to a common standard format. In this way when a workflow spans multiple organizations with different security architectures, no change to internal security architectures is required. In the healthcare domain, HL7 RIM provides a hierarchy for clinical and security roles which we adopt as our standard ontology [29,30]. The Policy Languages box supports a common policy language (Rei) to facilitate interconnection.

4.2.3. *Repository*

The repository resides between the *input* and *decision making engine* layers as an interface for the engine. The repository is used to maintain model generality by making the engine independent of any particular data format. The input data are stored in the repository. The dynamic attributes of system entities such as user and resource contexts are also stored here.

4.2.4. Decision making engine

This layer uses access requests and the data gathered from other layers to make the access control decision. We now give the detailed specification of the major blocks of the *decision making engine* layer.

A policy specification language is chosen that is capable of expressing the desired policies. Access requests are entered as inputs to this block and a rule based process is followed to yield the final access control decision. In Section 6.2 the policy specification language and its supporting packages are introduced.

Access control manager. This block is responsible for determining the access decision based on the results gained from the other blocks of this layer. It first sends requests to the *action check* block and if valid sends them to the *core check* block to make decisions. This block would call the *behavior check* block if behavior related constraints are defined. If all of the defined constraints are satisfied, access is granted; otherwise access is denied. This block interacts with the *Administration* layer for monitoring and improvement purposes. While the tasks for making an access control decision are performed online (i.e. per each access request), the administrative tasks are performed offline and through the user *behavior repository*.

Action check. The *single action* part of the behavior based access control (defined in Sections 3.2 and 3.3) is specified here. This block accepts an action tuple as input and checks if the attribute values of the action tuple conform to the specified rules.

Core check. This block implements Algorithm 1. All of the policies defined in this algorithm such as constraining roles, resources, times, delegations, and teams are enforced in this block.

Behavior check. This block checks behavior related constraints by retrieving the *daily behavior* from the *behavior manager* block and comparing it with the *expected behavior* (defined in Section 3.3). An instance of expressing these rules using a policy specification language is shown in Appendix B.

Audit trail. Many security breaches are the results of individuals with legitimate access rights who view patient information that they should not be viewing as part of their duty. A suitable way to detect such breaches is to let the patient review the audit trails of their chart. The *audit trail* block establishes a historical record of user or system actions. Healthcare standards have specified the audit trail from different aspects [9,12,39]. They have suggested a list of auditable events including: node-authentication-failure, order-record-event, patient-care-assignment, patient-record-event, query-information, consent-directive-override, and user-authentication. They also express the minimum information to be logged including: user ID and role, user organization, patient ID, performed function, time stamp, supporting reason. This information can be extracted from the action tuple. In other words, it is possible to record action tuples as audit logs. This block is also responsible for making the records available to appropriate personnel.

4.2.5. Behavior construction

This layer is responsible for constructing basic data for the *decision making engine* layer, i.e. the *action* and *behavior* concepts. Different blocks are required to capture and represent these concepts. *Action sensor* senses any changes in the attributes of the action tuple. *Action extractor* composes the action tuple based on the data sensed by *action sensor*.

Action authenticator authenticates the context itself. Different methods such as statistical analysis, distributed reputation, and confidence value, are used for authenticating contexts [59]. Attribute values are checked for completeness and correctness. Erroneous values for critical attributes (such as user, role, team, requested profile and requested service) are rejected in order to maintain data consistency.

Behavior manager composes the behavior based on the input action tuple and the past history of user behavior and updates the *user behavior repository*. The *action reasoning* block uses context inference rules, input through the *input* layer, to infer the contexts that cannot be directly sensed. These data are passed to the *core check* block to apply relevant rules.

4.2.6. Administration

In order to improve the constraints defined by a security administrator, the system provides the capability of analyzing access related data to extract *common behaviors* of users (representing the current behaviors of users). The administrator can measure the distance between *common* and *expected behaviors* that he has defined. Ideally, if users tend to respect the access constraints defined for them, the gap between *common* and *expected behaviors* should be small. The administrator can use this distance as a measure to adjust the *expected behavior*, if necessary (i.e. adjusting the rules so that the users experience fewer unintended access denials while they respect their access constraints). In order to extract a *common behavior*, the following problem should be solved: *given a list of actions for a user, find the common sequences for each attribute (if such sequences exist) to construct the user behavior.*

Algorithm 2 is one possible solution to the stated problem. Giving the main ideas required, it can be used as a starting point for further refinement. This algorithm has two functionalities: finding the common attribute values and determining the order of a sequence of attribute values. Algorithm 2 can be run for sequences with different lengths and for different sets of attributes.

Figure 3 shows an example of applying Algorithm 2. Given values for an action attribute for five days, Algorithm 2 creates lists of activities for each day and calculates the *cumulative list*. Then assuming $N = 5$, the first five values are identified. Finally, behavior is represented, as a set of *combinations* of action attributes with different sizes (*length*) and occurrence frequencies (*occurrenceCount*).

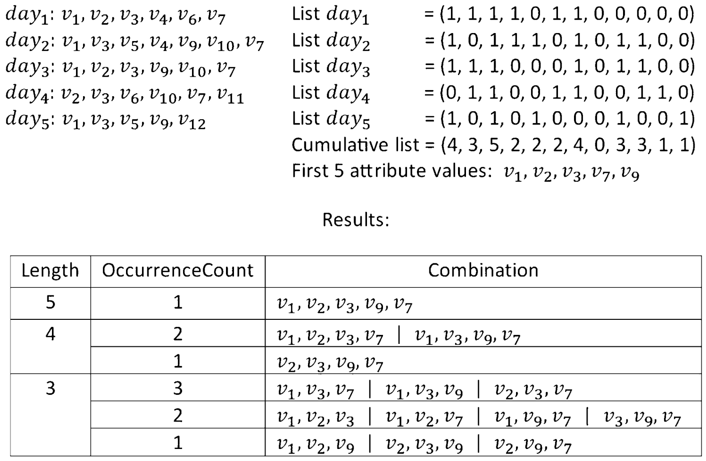
A second functionality proposed here (not specified completely) is to configure the action tuple based on the environment. The user behavior repository is reviewed to determine if certain attributes of the action tuple are not used. If so, the action tuple can be condensed. For example in an environment where no security teams

Algorithm 2. Extract behavior sequence for an attribute value from a set of action tuples

Input: (i) List of user action records; (ii) threshold

Output: *behavior*, as a set of *combinations* of action attributes with different sizes and occurrence frequencies greater than *threshold*

```
 1: List the values that appear in user action records for a given attribute
 2: for each $day$ do
 3:     Initialize an instance of the list (of step 1) to 0
 4:     Mark an attribute value as 1 if the user has used it in this $day$
 5: end for
 6: CumulativeList := Sum up the list of different days (used in steps 2–5)
 7: TopNList := attribute values with the greatest frequency of occurrence
 8: for length: $N$ to 2 do
 9:     for each combination with size length of the first N elements of TopNList do
10:         occurrenceCount = number of matches in all days with this combination
11:         if occurrenceCount $>$ threshold then
12:             behavior := behavior $\cup$ combination
13:         end if
14:     end for
15: end for
16: return behavior
```



day_1: $v_1, v_2, v_3, v_4, v_6, v_7$
day_2: $v_1, v_3, v_5, v_4, v_9, v_{10}, v_7$
day_3: $v_1, v_2, v_3, v_9, v_{10}, v_7$
day_4: $v_2, v_3, v_6, v_{10}, v_7, v_{11}$
day_5: $v_1, v_3, v_5, v_9, v_{12}$

List day_1 = (1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0)
List day_2 = (1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0)
List day_3 = (1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0)
List day_4 = (0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0)
List day_5 = (1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1)
Cumulative list = (4, 3, 5, 2, 2, 2, 4, 0, 3, 3, 1, 1)
First 5 attribute values: v_1, v_2, v_3, v_7, v_9

Results:

Length	OccurrenceCount	Combination
5	1	v_1, v_2, v_3, v_9, v_7
4	2	v_1, v_2, v_3, v_7 | v_1, v_3, v_9, v_7
	1	v_2, v_3, v_9, v_7
3	3	v_1, v_3, v_7 | v_1, v_3, v_9 | v_2, v_3, v_7
	2	v_1, v_2, v_3 | v_1, v_2, v_7 | v_1, v_9, v_7 | v_3, v_9, v_7
	1	v_1, v_2, v_9 | v_2, v_3, v_9 | v_2, v_9, v_7

Fig. 3. An example illustrating Algorithm 2.

exist, there is no need to keep the team attribute in the tuple. This helps to reduce unnecessary complexities in the model.

Ideally, system characteristics can be observed to extract a security configuration that better serves security requirements. Some of the system characteristics to consider are: interactions of instances of different roles, user assignment to different roles, type of requested data or service, resource request frequency, delegation frequency between roles, denied access requests and data flow at inter and intra organizational levels. The access control policy can be configured based on these characteristics to satisfy dynamic user access requirements such as extending user access rights by adding new permissions, modifying delegation rules between roles, and increasing resource accessibility.

5. Access control model evaluation

In this section we discuss the complexity of our framework. We also clarify for what purposes the behavior concept is and is not employed during the access control decision process.

In the following we examine the complexity of our framework from different perspectives, and suggest that the imposed complexity is reasonable.

- *Inherent complexities.* Our framework provides a number of facilities which allow security administrators to express different privacy policy rules. However, the framework does not impose any obligations on the administrators to use complicated rules, unless such rules are inherent to their system. Therefore the complexity of the rules depends on the nature of the underlying system.
- *Usability complexities.* A major factor in usability complexity is related to configuring the generic framework for a specific system. This issue is discussed in Section 6.2. The administrator should become familiar with the syntaxes chosen for modeling policies and contexts in the *input layer*. To address this concern, a graphical interface is prototyped and explained in Section 6.3.
- *Process complexities.* We define security rules and provide a corresponding engine to grant or deny access requests. This generic approach is common in the literature [11, 33, 55, 58, 59]. We further optionally provide a facility for behavior analysis.
- *Time complexity.* Algorithm 1, which covers major functionalities of the framework, has polynomial time complexity proportional to *max(number of permissions defined for the given user, number of access control policies(ACP)).* The running time of Algorithm 2 is O*(number of days in the user history * N!)* where *N* is the maximum length of sequence of attributes we are looking for. However *N* is fixed and small (typically less than 10 in practice). This makes the running time complexity polynomial in terms of the number of *days*.

Here we want to clarify more how we use the behavior concept in making an access control decision. In addition we explicitly describe the type of usage that we avoid. Assume we modify Algorithm 1 such that it automatically uses behavior analysis results in making the access control decision. Consider an extreme case where there are no *expected behavior* constraints (*BC* is not passed in line 16). In Algorithm 1 replace checking the expected behavior constraint (*BC*) with the following predicate: extract the *common behavior* of user u_i and compare it with $b(u_i)$, his daily behavior; if the distance of the two behaviors is more than a certain threshold

Algorithm 3. Measure the difference between two orders and notify if the distance is large

Input: (i) a *common order* and an *expected order*; (ii) threshold
Output: boolean, as distance measurement result

```
orderViolation = 0
for $\forall v_1, v_2 \centerdot v_1 \neq v_2 \wedge v_1, v_2 \in$ expected order do
    if occurrence order of $v_1$ and $v_2$ is different in common order then
        orderViolation = orderViolation + 1
    end if
end for
if orderViolation $>$ threshold then
    return false
else
    return true
end if
```

(see Algorithm 3 for one possible measurement method), the expression is evaluated to false (and therefore the access is denied); otherwise it is evaluated to true.

Such an algorithm causes different concerns and questions including:

- how to determine threshold variables (see Algorithm 3) to decide if a given behavior follows a *common behavior* or not. Is comparing *daily behavior* good enough or should we be considering behavior over a week? How far do we need to go back in time to create *common behavior*?
- the *common behavior* concept reflects the average and usual behavior of the user. *Common behavior* is not an exact representation of every instance of a user behavior. However the access control requires exact verifiable rules rather than statistical and threshold-based rules. Also it is not possible to control how restrictive a *common behavior* constraint can become. Could this impose further constraints on top of a user's workflow and regulation in order to access a resource?
- how to verify that a *common behavior* is actually a correct behavior from the access control point of view. There might be a case that *common behavior* represents a user's habit rather than the user's compliance to an authorization regulation. A scenario can be imagined where frequent invalid requests of a user eventually authorizes an invalid privilege. In such a case denying or granting access based on habit is not a correct decision.

These concerns could lead into access control vulnerabilities. To avoid these issues we have done the following. Firstly, we introduce the *expected behavior* concept. Dissimilar to *common behavior*, *expected behavior* defined by the administrator is an exact rule, not based on statistical data, and therefore does not suffer from the above issues. Secondly, we use the analysis results as suggestions and not directly including them in the decision making process. The analysis results once recognized and refined by human judgement, become free from the above shortcomings and are translated to exact rules to be used.

A large distance between two orders is the sign of either a user making frequent privilege violations (or a possible attack if no specific pattern can be determined) or an *expected behavior* rule that enforces invalid constraints. Therefore the administrator can either upgrade the *expected behavior* to cover a user's requirements or warn the user of frequent access request violations (or investigate the possibility of an attack). *Expected behavior* adjustment for the case of an *orderConstraint* is performed by reordering values that caused violations. Other types of *expected behaviors* are measured and adjusted in a similar manner. Algorithm 3 is one simplistic possibility for comparing ordering constraints among a *common* and an *expected behavior*.

In an environment like healthcare with a dedicated and trustful audience (compared to a public environment with potential attackers), it is not normal to find many access violations. Therefore suggestions are meant to reduce the number of violations. On the other hand, results can be used to discover new rules that can refine user access scope in case they are initially defined too broadly. The administrator can run this algorithm periodically to ensure current permissions fit the need-to-know principle discussed in the literature.

6. Simulated environment

In this section we provide an implementation for the architecture of our proposed access control model. This implementation aims to provide a prototype for different sections of the architecture. The current implementation is not intended to be used in real projects as it is not a complete instance of the proposed architecture.

The standard three-tier architecture is used for this prototype. It includes the *data access* layer which acts as an interface for the database. The *business layer* contains the core logic. Finally, the *presentation layer* offers a user interface. The following technologies and tools are used: Java, Eclipse Enterprise Edition, and MySql database. The database is created with the MySql Database Management System (DBMS).

6.1. *Ontology specification*

The Role hierarchy and context classes of the class diagram are represented by the Web Ontology Language (OWL). OWL is also used to map between specific system hierarchies and the hierarchy ontology offered by the model. The *owl:equivalent* feature is employed to define equivalency between attributes of hierarchies. Also, contexts and their relations are modeled using the *owl:objectProperty* and *owl:dataTypeProperty* features.

6.2. *Policy specification languages*

Several policy specification languages can be used. Amongst these, Ponder [51] and Rei [53] are two languages that support the requirements of our model. Here we explain how to use the Rei language to specify policies. Rei is a language based on OWL that allows policies to be specified as constraints over allowable and obligated actions on resources in the environment. Regarding the difficulties of using available tools, we implemented an engine for interpreting and applying the Rei specifications, explained in Section 6.4.

The policy specification is divided into separate files, each describing a different portion.

Ontology file

Different entities, contexts, and the general concepts (classes) of Fig. 1 are introduced in an ontology file. As an example, some of the entries of the ontology file (in the form *entity*(*attribute*)) are: Person(name, affiliation, isCareGiver) and Patient (locatedIn, associatedCareGiver).

Instance file

This file contains class definitions that are subclasses of the classes introduced in the ontology file and have constraints over entries of the instance file. In the following example an action called *NursDiabAction* is shown that represents those actions that a nurse of the Diabetes Department performs.

```
<owl:Class rdf:ID="NursDiabAction">
  <rdfs:subClassOf rdf:resource="DiabAction"/>
  <rdfs:subClassOf rdf:resource="MajorProfChg"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="actor"/>
      <owl:allValuesFrom rdf:resource="Nurse"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Other entries of the instance file are direct instances of defined classes. The following example shows how to introduce a patient to the system:

```
<hosp:Patient rdf:ID="Nero">
  <hosp:associatedCareGiver rdf:resource="Jane"/>
  <hosp:affiliation rdf:resource="DiabetesDept"/>
  <hosp:PatientProfile rdf:resource="NeroEHR"/>
  <hosp:locatedIn rdf:resource="room223"/>
</hosp:Patient>
```

Policy file

Policy files contain constraints, different types of policies and meta policies, and the collection of active policies. Constraints are logical expressions restricting the attribute values of objects. The following example describes the constraint of someone being a member of the Diabetes Department.

```
<constraint:SimpleConstraint rdf:ID="IsMemberOfDiab">
  <constraint:subject rdf:resource="var1"/>
  <constraint:predicate rdf:resource="affiliation"/>
  <constraint:object rdf:resource="DiabetesDept"/>
  <policy:desc>
    All members of Diabetes Department
  </policy:desc>
</constraint:SimpleConstraint >
```

Policies are described using *permission/prohibition*, *delegation/revocation*, and *obligation* tags. Different policies (role, team, and context) are modeled based on the constraints that must be checked prior to giving permission to a user. Policies can be specific or generic. The former assigns access rights to individuals while the latter describes the general conditions under which an access right is granted or denied. An example of giving specific permission to Jane, a nurse of the Diabetes Department, to perform an action defined by nurses in the Diabetes Department is:

```
<deontic:Permission rdf:ID="Perm_Jane">
  <deontic:actor rdf:resource="&inst;Jane"/>
  <deontic:action rdf:resource= "&inst;ANursDiabAction"/>
</deontic:Permission>
```

Different *expected behaviors* can be expressed with Rei in a similar manner. Interested readers are encouraged to check Appendix B for these specifications.

6.3. Administrator interface

We have developed a prototype for administrators in order to reduce the complexity of employing the model and facilitate the interactions with the model defined in the *Administration* layer. This prototype provides a Graphical User Interface (GUI) as an alternative for the syntactical representation introduced in Section 6.2.

The prototype has a menu for each of the sets, relations, and constraints of the formalisms defined in Section 3.3. It also provides a monitoring menu, consisting of retrieving common behavior and viewing audit trails.

Figures 4–6 provide a number of snapshots of the prototype. Figure 4 shows items of the “primitive” menu for entering primitive data type values. Figure 5 presents the access control policy creation page. A list of available values for each of the items of this relation is shown on the right. Once a valid value is entered for all items, the relation can be created. Figure 6 illustrates the audit trail page for a given user.

Figure 7 shows how to check for an existing or potential *associationConstraint* relation by passing a user id, an action attribute value, and a time period as inputs. In this example, the filtering attribute is team with value of *diabeticDeptNursingTeam*. Figure 8 shows the results of comparing a common and an expected behavior. The common behavior is extracted using the user attribute of the relevant *associationOrder*. In this example the *associationOrder* filters user location with *diabeticNusringStation*.

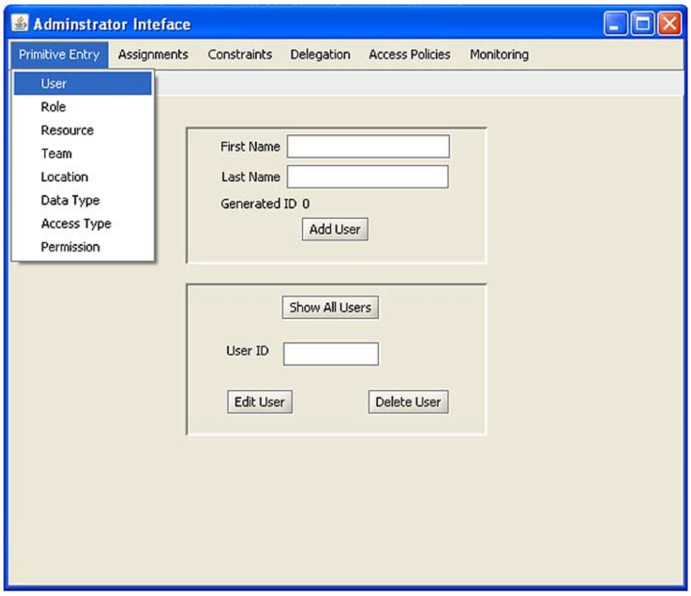
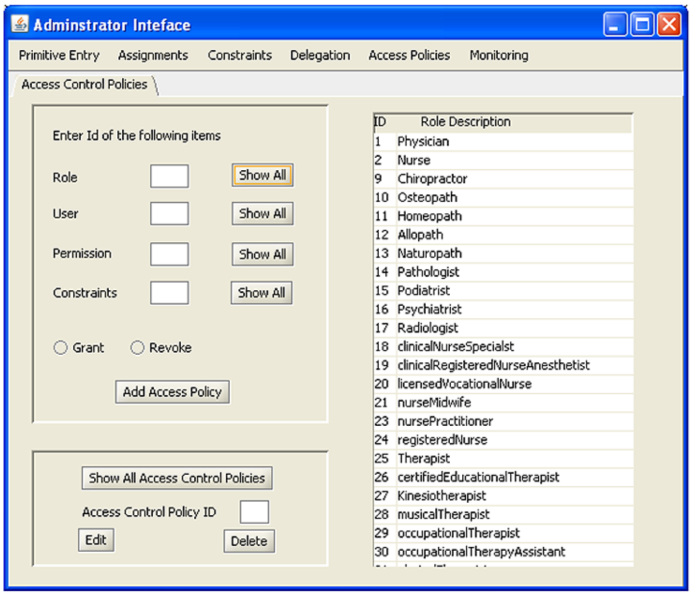
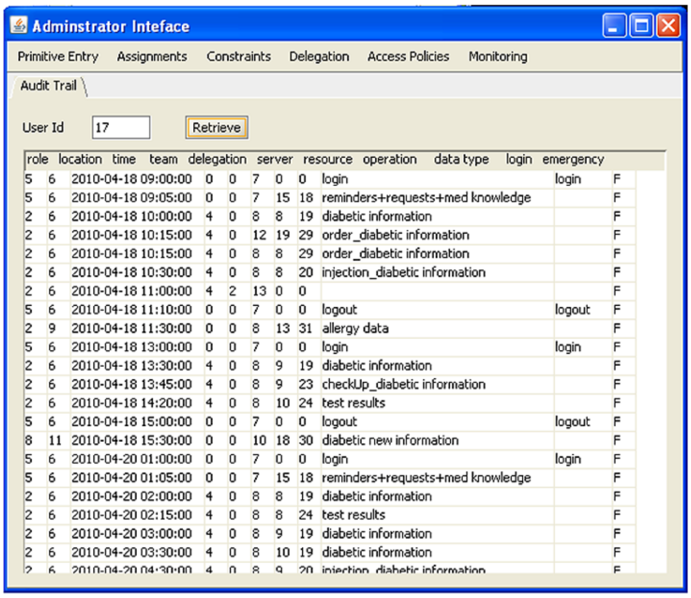


Fig. 4. Administrator's user page. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)



ID	Role Description
1	Physician
2	Nurse
9	Chiropractor
10	Osteopath
11	Homeopath
12	Allopath
13	Naturopath
14	Pathologist
15	Podiatrist
16	Psychiatrist
17	Radiologist
18	clinicalNurseSpecialst
19	clinicalRegisteredNurseAnesthetist
20	licensedVocationalNurse
21	nurseMidwife
23	nursePractitioner
24	registeredNurse
25	Therapist
26	certifiedEducationalTherapist
27	Kinesiotherapist
28	musicalTherapist
29	occupationalTherapist
30	occupationalTherapyAssistant

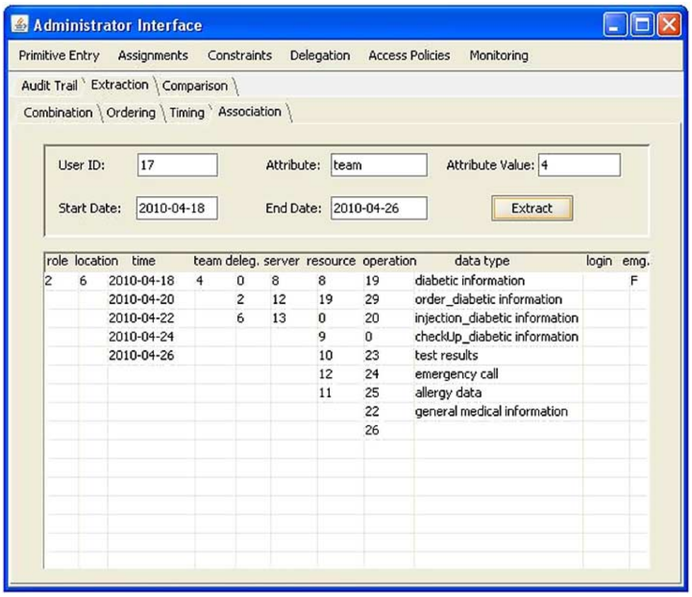
Fig. 5. Administrator's access control policy page. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)



User Id 17

role	location	time	team	delegation	server	resource	operation	data type	login	emergency
5	6	2010-04-18 09:00:00	0	0	7	0	0	login	login	F
5	6	2010-04-18 09:05:00	0	0	7	15	18	reminders+requests+med knowledge		F
2	6	2010-04-18 10:00:00	4	0	8	8	19	diabetic information		F
2	6	2010-04-18 10:15:00	4	0	12	19	29	order_diabetic information		F
2	6	2010-04-18 10:15:00	4	0	8	8	29	order_diabetic information		F
2	6	2010-04-18 10:30:00	4	0	8	8	20	injection_diabetic information		F
2	6	2010-04-18 11:00:00	4	2	13	0	0			F
5	6	2010-04-18 11:10:00	0	0	7	0	0	logout	logout	F
2	9	2010-04-18 11:30:00	0	0	8	13	31	allergy data		F
5	6	2010-04-18 13:00:00	0	0	7	0	0	login	login	F
2	6	2010-04-18 13:30:00	4	0	8	9	19	diabetic information		F
2	6	2010-04-18 13:45:00	4	0	8	9	23	checkUp_diabetic information		F
2	6	2010-04-18 14:20:00	4	0	8	10	24	test results		F
5	6	2010-04-18 15:00:00	0	0	7	0	0	logout	logout	F
8	11	2010-04-18 15:30:00	0	0	10	18	30	diabetic new information		F
5	6	2010-04-20 01:00:00	0	0	7	0	0	login	login	F
5	6	2010-04-20 01:05:00	0	0	7	15	18	reminders+requests+med knowledge		F
2	6	2010-04-20 02:00:00	4	0	8	8	19	diabetic information		F
2	6	2010-04-20 02:15:00	4	0	8	8	24	test results		F
2	6	2010-04-20 03:00:00	4	0	8	9	19	diabetic information		F
2	6	2010-04-20 03:30:00	4	0	8	10	19	diabetic information		F
2	6	2010-04-20 04:30:00	4	0	8	9	20	injection_diabetic information		F

Fig. 6. Audit trail page of the administrator interface for a typical user (User Id 17). (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)



User ID: 17

Attribute: team

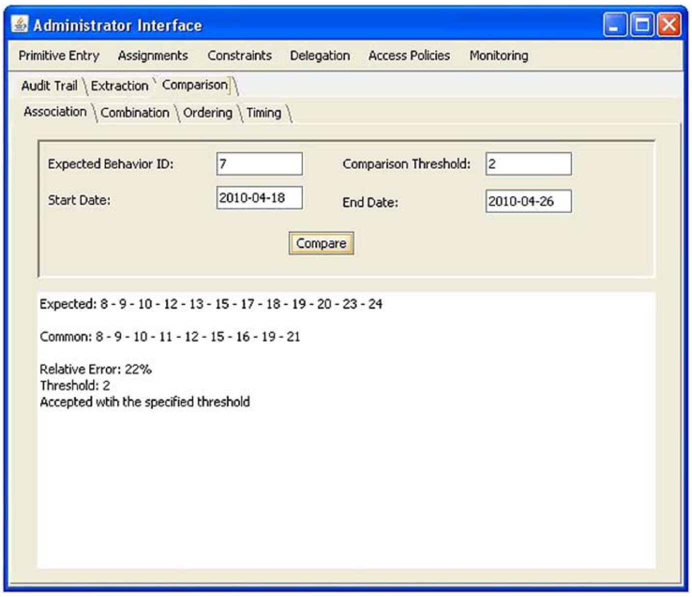
Attribute Value: 4

Start Date: 2010-04-18

End Date: 2010-04-26

role	location	time	team	deleg.	server	resource	operation	data type	login	emg.
2	6	2010-04-18	4	0	8	8	19	diabetic information		F
		2010-04-20		2	12	19	29	order_diabetic information		
		2010-04-22		6	13	0	20	injection_diabetic information		
		2010-04-24				9	0	checkUp_diabetic information		
		2010-04-26				10	23	test results		
						12	24	emergency call		
						11	25	allergy data		
							22	general medical information		
							26			

Fig. 7. *AssociationConstraint* extraction page for a typical user (User Id 17). (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)



Expected Behavior ID: 7

Comparison Threshold: 2

Start Date: 2010-04-18

End Date: 2010-04-26

```
Expected: 8 - 9 - 10 - 12 - 13 - 15 - 17 - 18 - 19 - 20 - 23 - 24

Common: 8 - 9 - 10 - 11 - 12 - 15 - 16 - 19 - 21

Relative Error: 22%
Threshold: 2
Accepted wtih the specified threshold
```

Fig. 8. *AssociationConstraint* comparison page. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0454.)

6.4. Implementation classes

The classes that are used to implement the simulation environment are shown in the class diagram of Fig. 9.

The **ACmodel** class initializes the model by entering the policy rules. Access requests are sent to the model through this class. These requests are objects of the **UserBehavior** class which are sent to the **ACmanager** class. The **DBmanager** class resides in the data layer and is the interface between the data layer and other classes. The **FileScannerParser** class is responsible for inserting the policy rules from the Rei policy specification files into appropriate tables of the database. The **CoreCheck**, **ActionCheck**, **BehaviorCheck**, and **ACmanager** classes perform the responsibilities of the *decision making engine* and *administration* layers. Finally the **AdministratorInterface** class represents the administrator interface described in Section 6.3.

6.5. Simulation result

Different ways of modeling and categorizing tasks of care givers exist. These models have differing degrees of commonality. For example the model in [47] describes nursing tasks as direct case (physical and psychological), direct cure, and adjunct activities, while the model in [52] expresses tasks as assessment, nursing diagnosis, planning, implementation, and evaluation; these models have both overlapping and

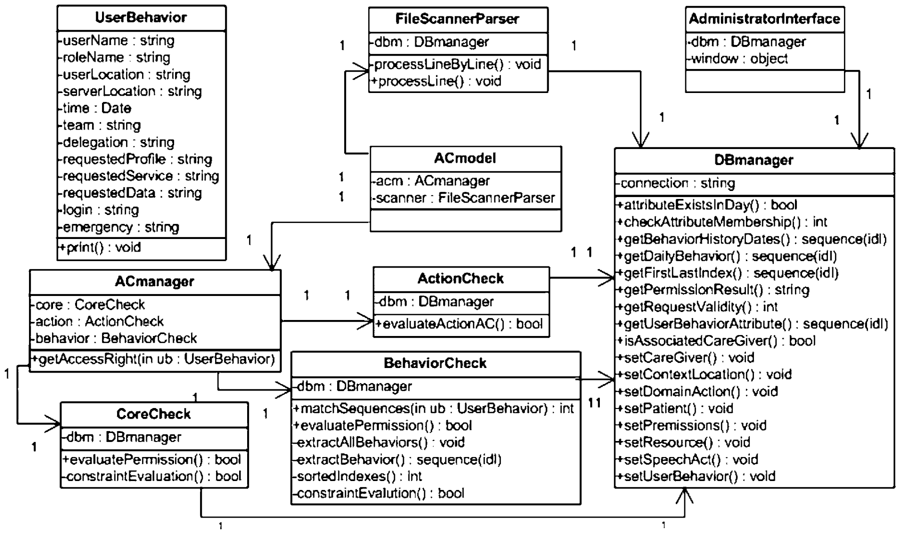

UserBehavior
-userName : string
-roleName : string
-userLocation : string
-serverLocation : string
-time : Date
-team : string
-delegation : string
-requestedProfile : string
-requestedService : string
-requestedData : string
-login : string
-emergency : string
+print() : void
FileScannerParser
-dbm : DBmanager
-processLineByLine() : void
+processLine() : void
AdministratorInterface
-dbm : DBmanager
-window : object
ACmodel
-acm : ACmanager
-scanner : FileScannerParser
DBmanager
-connection : string
+attributeExistsInDay() : bool
+checkAttributeMembership() : int
+getBehaviorHistoryDates() : sequence(idl)
+getDailyBehavior() : sequence(idl)
+getFirstLastIndex() : sequence(idl)
+getPermissionResult() : string
+getRequestValidity() : int
+getUserBehaviorAttribute() : sequence(idl)
+isAssociatedCareGiver() : bool
+setCareGiver() : void
+setContextLocation() : void
+setDomainAction() : void
+setPatient() : void
+setPremissions() : void
+setResource() : void
+setSpeechAct() : void
+setUserBehavior() : void
ACmanager
-core : CoreCheck
-action : ActionCheck
-behavior : BehaviorCheck
+getAccessRight(in ub : UserBehavior)
ActionCheck
-dbm : DBmanager
+evaluateActionAC() : bool
BehaviorCheck
-dbm : DBmanager
+matchSequences(in ub : UserBehavior) : int
+evaluatePermission() : bool
-extractAllBehaviors() : void
-extractBehavior() : sequence(idl)
-sortedIndexes() : int
-constraintEvalution() : bool
CoreCheck
-dbm : DBmanager
+evaluatePermission() : bool
-constraintEvaluation() : bool

Fig. 9. The class diagram of the implementation classes.

disjoint parts. The rules governing healthcare institutions also vary depending on the nature of the institutions and the decisions made by local managers of those institutions. After talking to a number of care givers in the McMaster University Medical Centre (MUMC) and consulting the literature ([44,47,50,52]), we extracted typical common activities and daily schedules. We focused on ward nurses to develop a case study covering different rules that require access control.

In this simulation, access requests for a number of care givers in a day are represented. We have chosen these requests such that different rules and constraints discussed in the paper are covered. We have avoided considering similar repetitive requests, as we believe they do not provide additional insight for understanding the model. In these requests, all personnel are affiliated with the *Diabetes Department*. Involved parties are nurses: *Jane, Julia, Josh*; students: *Flora*; unlicensed assistive personnel: *Daria*; patients: *Nero, Nancy, Natalie, Mike, Nash, Sara*. Locations are EMR server, Shared Health Record (SHR), Active Role Server, library computer, Diabetes Nursing Station (DNS). Table 3 represents the requests in plain English. Table 4 shows requests expressed by action tuples corresponding to Table 3.

The following rules are assumed (based on our consultation with care givers and the literature):

1. Care givers can only access patient profiles within their departments.
2. There must be a time interval of at least 5 minutes between access requests from the *Diabetes Department* and the library (due to their distance).
3. Nurses are not allowed to delegate certain procedures (such as assessments and nursing diagnosis) to unlicensed assistive personnel while they are allowed to delegate some others (such as taking vital signs and intake and output).

Table 3

Description of a portion of access requests made to the model in one day

#	Description
1	*Jane* logs into her account information and registers as the responsible nurse.
2	*Jane* reviews her account.
3	*Jane* joins the Diabetes Department team and tries to review *Sara*'s profile, a patient not assigned to *Jane*.
4	*Jane* updates supply change information of *Nancy*, a patient assigned to *Jane*.
5	As a nurse *Jane* delegates her diagnosis authority to *Daria*, a nurse in the same team.
6	*Jane* updates check up information of *Nancy*, a patient assigned to *Jane*.
7	*Jane* reviews *Natalia*'s profile, a patient assigned to *Jane*.
8	*Jane* updates drug information of *Natalia*, a patient assigned to *Jane*.
9	As a nurse *Jane* tries to discharge *Nancy*, a patient assigned to *Jane*.
10	As a member of Cardiac Department team *Jane* tries to update check up information of *Nancy*, a patient assigned to *Jane*.
11	*Jane* logs out of her account.
12	*Jane* commits some search request from the library.
13	*Julia* tries to update exercise information of *Nancy*, a patient not assigned to Julia. (Assume *Julia* has logged in and registered earlier in the day.)
14	*Julia* tries to update check up information of *Nash*, a patient assigned to *Julia*.
15	*Julia* updates injection information of *Nero*, a patient assigned to *Julia*.
16	*Julia* tries to perform a search from a resource library.
17	As a member of the surgery team *Josh* reviews *Nero*'s profile.
18	As a member of the surgery team *Josh* tries to access a *Nancy*'s profile in a different operation.
19	As a member of Diabetes Department team, *Josh* tries to review *Mike*'s profile, a patient assigned to *Josh*.
20	*Flora* as a student tries to update the diagnosis information of *Mikes*'s profile in absence of an attending physician.

4. Nurse *Julia* should visit patients in the following order: *Nero-Nash*.
5. A care giver must not participate in surgeries which are less than 3 hours apart.
6. The patients assigned to nurse *Julia* are: *Nero* and *Nash*.
7. Care givers should register as responsible for their shift, before they are allowed to do anything else.
8. There are certain procedures which require confirmation of a care giver with higher permission levels.
9. Students gain access through a delegation made by their attending physicians.

The accesses which are denied are given in Table 5. For each access the type of unsatisfied constraint and the rule violated are shown. All other accesses are granted.

Table 6 shows details of the action tuples and the results of each of the constraints for two of the above access controls. For each request, a portion of the results of the blocks of the *decision making engine* layer is shown. We will not represent the action

Table 4

A portion of access requests made to the model in one day (corresponding to Table 3), represented in the form of action tuples

#	Person	Role	Location of user	Location of server	Time of day	Team	Delegation	Requested profile	Requested service	Emg.
1	Jane	User	DNS	EMR server	2010.11.30-9:00	null	null	Jane's account	log in	F
2	Jane	User	DNS	EMR server	2010.11.30-9:05	null	null	Jane's account	review	F
3	Jane	Nurse	DNS	SHR	2010.11.30-10:00	diabetes nursing	null	Sara's profile	review	F
4	Jane	Nurse	DNS	SHR	2010.11.30-11:00	diabetes nursing	null	Nancy's profile	supply change	F
5	Jane	Nurse	DNS	active role server	2010.11.30-11:30	null	delegate diagnosis to Daria	active roles database	null	F
6	Jane	Nurse	DNS	SHR	2010.11.30-12:00	diabetes nursing	null	Nancy's profile	check up	F
7	Jane	Nurse	DNS	SHR	2010.11.30-13:30	diabetes nursing	null	Natalie's profile	review	F
8	Jane	Nurse	DNS	SHR	2010.11.30-15:00	diabetes nursing	null	Natalie's profile	drug X delivered	F
9	Jane	Nurse	DNS	SHR	2010.11.30-15:00	diabetes nursing	null	Natalie's profile	discharge	F
10	Jane	Nurse	DNS	SHR	2010.11.30-16:00	cardiac nursing	null	Nancy's profile	check up	F
11	Jane	User	DNS	EMR server	2010.11.30-16:15	null	null	Jane's account	log out	F
12	Jane	Researcher	library computer	library server	2010.11.30-16:30	null	null	library database	search	F
13	Julia	Nurse	DNS	SHR	2010.11.30-10:30	diabetes nursing	null	Nancy's profile	help exercise	F
14	Julia	Nurse	DNS	SHR	2010.11.30-10:45	diabetes nursing	null	Nash's profile	check up	F
15	Julia	Nurse	DNS	SHR	2010.11.30-10:30	diabetes nursing	null	Nero's profile	perform injection	F
16	Julia	Researcher	library computer	library server	2010.11.30-10:32	null	null	library database	search	F
17	Josh	Nurse	operating room	EMR server	2010.11.30-12:30	operating team	null	Nero's profile	review	F
18	Josh	Nurse	operating room	EMR server	2010.11.30-14:00	operating team	null	Nancy's profile	review	F
19	Josh	Nurse	DNS	SHR	2010.11.30-14:30	diabetes nursing	null	Mike's profile	review	F
20	Flora	Student	DNS	SHR	2010.11.30-15:00	students	null	Mike's profile	update diagnosis	F

Table 5

Denied access requests supported by corresponding violated constraints and rules

#	stnd.	Acn.	Deleg.	Order.	Assoc.	Time	Context	Logic.	Explanation
3	×	√	√	√	×	√	√	√	Breaking rule 1
5	√	√	×	√	√	√	√	√	Breaking rule 3
9	×	√	√	√	√	√	√	×	Breaking rule 8
10	√	×	√	√	√	√	√	√	Invalid team value
13	√	√	√	√	×	√	√	√	Breaking rule 6
14	√	√	√	×	√	√	√	√	Breaking rule 4
16	√	√	√	√	√	√	×	√	Breaking rule 2
18	√	√	√	√	√	×	√	√	Breaking rule 5
19	√	√	√	√	√	√	√	×	Breaking rule 7
20	√	√	×	√	√	√	√	√	Breaking rule 9

Table 6

Sample simulation results for two requests, including action tuples and results of checking different constraints

Action tuple *time:11:00:00.0, user:Jane, role:Nurse, user location: diabeticNursingStation, server location:sharedHealthRecord, team: DiabeticDeptNursingTeam, requested profile:NancyEHR, requested service: DiabReview, requested data:diabetic information, emergency:false*

Core: Access granted with generic inferred permission
Action: All arguments are valid!
Behavior: Expected behavior is followed!
Logical: Logical constraints are satisfied!
Delegation: No delegation applies!
FINAL DECISION: Access is GRANTED!

Action tuple *time:14:00:00.0, user:Josh, role:Nurse, user location:operatingRoom, server location:EMRserver, team:OperatingTeam, requested profile:NancyEHR, requested service:SignIn, requested data:SignIn, emergency:false*

Core: Access granted with generic inferred permission
Action: All arguments are valid!
Behavior: A time constraint is violated!
FINAL DECISION: Access is DENIED!

tuples and result details for other requests as they look similar to the Table 6 entries and are generated using Tables 4 and 5.

7. Related work

In this section, the literature is reviewed for specific access control models for the healthcare domain.

According to Ferreira et al. [20], who reviewed 59 articles on access control in the healthcare domain, 22 out of 40 selected articles used Role Based Access Control (RBAC). A few of these access control models are described here. Li et al. [46] extend RBAC for a laboratory information system. They define constraints on the *permission* relation. They extract the different roles and their associated job functions. The access privilege is specified up to accessing particular tables or data records within tables. Aspect Oriented Programming (AOP)[1] is used to implement this model. This allows a quite easy integration of the access control model into an existing system without changing the existing code.

Hung [35] provides an extended RBAC model that focuses on privacy. He investigates the privacy requirements identified by HIPAA (see Section 2). He determines the major privacy concerns as (1) the acquisition, storage, and processing of data, (2) consent for processing and disclosure of data, (3) the rights of subjects to access and modify their datasets. Schwartmann [55] proposes an attributable RBAC. The idea is to attach constraints in the form of attribute-values (e.g. attending physician of a specified patient) to the permissions.

Jih et al. [42] offer Context Aware Access Control (CAAC) explaining a sample scenario, with sensing and collecting contexts. The proposed context-aware rule engine is intended to run on resource-limited mobile devices. Jahnke et al. [41] provide a context-aware information service for healthcare systems. They introduce an ontology-based context management system that allows a user to define contexts employing terms from the medical field. Their context ontology is composed of domain independent (e.g. time and location) and domain dependent (following HL7 RIM) parts. Toninelli et al. [56] present a secure collaboration mechanism. They use CAS and semantic modeling technologies to provide a semantic CAAC framework. Semantic technologies are used for context/policy specification to allow high-level description and reasoning.

Hung et al. [36] discuss issues in developing a privacy access control model for supporting mobile and ad hoc healthcare applications. They develop their model, considering privacy rules for Protected Health Informatics (i.e. limitation on collection, disclosure, use, and retention) and mobility. They use a policy specification language to specify and for further implementation.

Blobel [7] proposes a generic access control model for electronic health record systems that deals with policy description including policy agreements, authentication, certification, and directory services. These elements form a privilege management infrastructure. A model is proposed for each of the following areas: domain,

[1] AOP is based on the idea that computer systems are better programmed by separately specifying the various concerns of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program [18].

policy, role, privilege management, and information distance (between originator, producer, and administrator of information) [6].

Hafner et al. [23] identify a number of use cases in the healthcare domain such as dynamic access control, delegation, break glass, 4-eyes-principle, and usage control. They specialize the SELECT-Framework for model driven security for the healthcare domain based on UCON (a security policy model for usage control). They use UML and OCL to express constraints and relations. Verhanneman et al. [58] support usage of fine-grained and dynamic policies in the healthcare domain. They focus on a reconfigurable implementation and identify a number of shortcomings of current technologies for aiding their implementation. Chandramouli [14] identifies the emergency access requirement and proposes a logic driven role based dynamic model supporting context constraints. He introduces a concept called ‘domain’ to group objects. Anderson [2] proposes a security policy model for the healthcare domain comparable to other application domains such as banking and the military.

Bhatti et al. [5] have investigated use cases introduced by healthcare standards to design a context-aware policy specification language called XML-based Generalized Temporal Role-Based Access Control. They use the HL7 CDA standard as a resource hierarchy and ideas from Hippocratic databases to impose restrictions on the resource side. Their proposed language is expressive enough to capture healthcare environment requirements. They intend to shift information management from being organization oriented to patient centric. Among the related work presented here, this work has more in common with our research purposes. They provide a refinement of possible temporal restrictions which has some overlaps and distinctions with our *timeConstraint* relation. Emergency conditions are discussed when a request is made near an emergency department, but the authors do not consider the case where an emergency occurs in other locations. Similar to our approach, HL7 CDA is used as the resource hierarchy. However we employ healthcare standards not only in the resource hierarchy, but also in role hierarchy and the policy description. Also our work targets interoperability, visualization and behavior analysis, formal definition, audit trail, and administrative aspects which are not discussed in their work. Finally while the focus of their work is introducing a specification language, we intend to reuse existing specification languages.

Table 7 compares our proposed model with the above models. The numbers in the second row are the bibliographical citations of the methods which are being compared. In the table, *y* indicates that the method incorporates the corresponding feature, *n* means that the method does not incorporate the feature, and *n/a* means that the feature does not apply to the method (when a feature which is specific to the healthcare domain is applied to a generic access control method, this notation is used). We base our judgment on the explicit claims of authors about covering these requirements. We do not however specify to what extent each work serves a requirement. While in the following we briefly mention how our model satisfies the requirements, we restrict our comments about other works to the contents of Table 7.

Table 7
Comparison of different access control methods

Features	[2]	[5]	[4]	[6]	[7]	[23]	[33]	[42]	[45]	[55]	[56]	[58]	Our model
Context-awareness	n/a	y	y	n	n	y	y	y	n	n	y	y	y
Dynamicity	n	y	y	n	n	y	y	y	y	y	y	y	y
Delegation	y	n	n	n	y	y	n	n	y	y	n	n	y
Standards compatibility	n	y	n/a	y	y	n	n	n	n/a	n/a	n	n	y
Semantic interoperability	n/a	n	n/a	y	y	n	n	n	n	n/a	y	n	y
Emergency handling	y	y	n/a	n	n	y	n	n	n/a	n	n/a	n	y
Audit trail	y	n	n/a	y	n	y	n	n	n	n	n/a	y	y
Formal definition	n	y	y	n	n	n	y	n	y	y	n	n	y
Policy flexibility	y	y	y	n	y	y	y	y	y	y	y	y	y
Visualization	n	n	y	n	n	n	n	n	y	n	n	n	y
Workgroup access	y	n	n/a	n	n	n	n	n	n	n	n	n	y
Implementation	n/a	y	y	y	n	n	y	y	n	n	n	y	y

Table 7 ‘features’ include healthcare standard requirements. More specifically, context-awareness, delegation, emergency handling, audit trail, visualization, semantic interoperability, and workgroup access are described in Section 2 as standard requirements. We identify other features included in the table as supplementary features needed to implement standard features. In our model, the context related items included in the constraints and architecture provide context-awareness. The *delegation* relation makes delegation constraints possible. Usage of role, resource, and service hierarchies; introducing the policy model as an extension to HL7 RIM (including context ontology); and the *representation* layer in our architecture makes our model semantically interoperable. Emergency handling is discussed in Section 3 and reflected in Algorithm 1. Audit trail is captured by the *behavior* concept and the *audit trail* unit of the architecture. Visualization is captured by the *common behavior* concept and the *administration* layer of the architecture. The *Team* attribute of the action tuple implements the Workgroup access. Finally we note that the term implementation used in Table 7 does not necessarily include deployment in a real world project or institution. It rather refers to a prototype used as a proof of concept that offers a practical approach to implement a proposed model. This is the case for almost all of the works considered.

Comparing our work with those above, we have based our arguments and design on healthcare standards. The emergence of different standards and governmental financial and strategic supports aimed at providing interoperability among healthcare systems illustrates the fact that ad-hoc methods would not be reusable or even appropriate in the future. While many of the related works introduced here consider healthcare use cases, few of them have checked for compliancy with standards. Also in order for an access control model to be effective in this domain, it must be capable of dealing with all identified requirements. However some of these works remain silent on these requirements. Due to the distributed nature of the healthcare domain, interoperability and adaptability of the model is an essential feature not discussed in some of the related works. Finally our work is distinguishable due to the *behavior* concept that allows policy rules to describe more complex constraints. The framework facilitates monitoring and therefore adaptability to the deployed environment, a feature not discussed systematically in the literature.

8. Conclusion

In this paper we presented a detailed framework for provision of security aspects in distributed healthcare systems. The model is generic in the sense that it has a layer that allows the user to map environment specific contexts onto a standard internal set of contexts. Consequently, these contexts are fed into an access control engine to make an access decision for the corresponding user. The proposed access control method models the user action as a tuple of major contexts which in turn allows us to demonstrate different attributes of a user.

The model is designed to be compatible with HL7 RIM. It extends RIM's class diagram to incorporate the proposed behavior-based access control. The proposed model satisfies requirements of the healthcare domain such as patient consent, authorization in emergency situations, auditing of all events and considering care givers activities.

In terms of future work, there are several important avenues:

- It is desirable to deploy our framework in a realistic environment to further evaluate and improve our method. However getting involved and leading projects in the healthcare domain is not an easy task.
- The analysis and algorithms introduced for the components of the architecture can be improved to determine more complex behaviors and better use them to make access control decisions. This might require extending the definition of *user behavior*.
- As another application domain, the *user behavior* concept and its corresponding model can be employed to provide guidelines for care givers based on their behavior. In this way, a user is able to gain recommendations based on his behavior and his colleagues' behaviors.

Appendix A. Additional constraints

In this appendix the formal definition of some constraints introduced in Section 3.2 are described.

AssociationConstraint: $\mathit{AssCon}: U \times (\mathit{att}_1)_{\mathit{val}} \mapsto 2^{(\mathit{att}_2)_{\mathit{val}}}$, is a function returning the set of all possible attribute values (the exclusive set, meaning that no other attribute values are allowed) for att_2 given that $(\mathit{att}_1)_{\mathit{val}}$ is assigned to att_1 for a user u, where $\mathit{att}_1, \mathit{att}_2 \in \mathit{ATT}$ and $\mathit{att}_1 \neq \mathit{att}_2$. For example $\mathit{AssCon}(\mathit{Jane}, \mathit{nurse}) = \{\mathit{SaraProfile}, \mathit{NeroProfile}, \mathit{NancyProfile}\}$.

TimeConstraint: $\mathit{TimCon}: U \times (\mathit{att}_{\mathit{val}})_1 \times (\mathit{att}_{\mathit{val}})_2 \mapsto T \times T$, is a function expressing the valid time interval $[t_1, t_2]$ (where $t_1, t_2 \in T$ and $t_1 \neq t_2$) between the occurrence of two values $(\mathit{att}_{\mathit{val}})_1$ and $(\mathit{att}_{\mathit{val}})_2$ for a user u. This function can also be used to assert the minimum time interval required to record action tuples from two locations.

CombinationConstraint: $\mathit{ComCon}: U \times 2^{(\mathit{att}_1)_{\mathit{val}}} \times \mathit{Mandatory} \times \mathit{MinLength} \mapsto 2^{(\mathit{att}_2)_{\mathit{val}}}$, is a function describing the valid set of attribute values for att_2 if a certain order of attribute values occurs for att_1. This constraint is a combination of order and association constraints.

Appendix B. *orderConstraint* specification

In this appendix we specify *OrderConstraint*. One of the possible ways to express the *expected behavior*, using Rei syntax is through Rei constraints. As an example we describe *OrderConstraint* (defined in Section 3.3) here. The following is added to the *ontology file*:

```
<rdf:Class rdf:label="behaviorConstraint">
 <rdf:subClassOf rdf:resource="Constraint"/>
</rdf:Class>

<rdf:Class rdf:label="orderConstraint">
 <rdf:subClassOf rdf:resource="behaviorConstraint"/>
</rdf:Class>

<rdf:Property rdf:label="valueSet">
 <rdf:domain rdf:resource="&orderConstraint"/>
 <rdf:range rdf:resource="&literal"/>
</rdf:Property>

<rdf:Property rdf:label="mandatory">
 <rdf:domain rdf:resource="&orderConstraint"/>
 <rdf:range rdf:resource="&literal"/>
</rdf:Property>

<rdf:Property rdf:label="minLength">
 <rdf:domain rdf:resource="&orderConstraint"/>
 <rdf:range rdf:resource="&integer"/>
</rdf:Property>
```

```
<rdf:Property rdf:label="actionAttribute">
 <rdf:domain rdf:resource="&orderConstraint"/>
 <rdf:range rdf:resource="&literal"/>
</rdf:Property>

<rdf:Class rdf:label="dailyBehavior">
 <rdf:subClassOf rdf:resource="Resource"/>
</rdf:Class>

<rdf:Property rdf:label="allValues">
 <rdf:domain rdf:resource="&dailyBehavior"/>
 <rdf:range rdf:resource="&literal"/>
</rdf:Property>

<rdf:Property rdf:label="user">
 <rdf:domain rdf:resource="&dailyBehavior"/>
 <rdf:range rdf:resource="&person"/>
</rdf:Property>

<inst:orderConstraint rdf:ID="AnOrderConstraint">
```

In the above entries, the *valueSet* property of the *orderConstraint* class expresses the valid attribute value order for the attribute specified by the *actionAttribute* property. The *mandatory* attributes describe the necessity of each of the attribute values in the *valueSet*. The *allValues* property of the *dailyBehavior* class contains the values for all of the attributes of an action tuple in the current day (this should be built online). The following should be added to the *policy file*:

```
<policy:Granting >
 <policy:to rdf:resource="var1"/>
 <policy:deontic rdf:resource="Perm_X"/>
 <policy:requirement rdf:resource="followsOrder"/>
</policy:Granting>

<constraint:SimpleConstraint rdf:ID="followsOrder">
 <constraint:subject rdf:resource="var1"/>
 <constraint:predicate rdf:resource="matches"/>
 <constraint:object rdf:resource="AnOrderConstraint"/>
</constraint:SimpleConstraint>
```

Since we are developing the Rei engine ourselves, we will properly handle the *match* property on an *orderConstraint* class to perform the desired procedure (which is evaluating a *daily behavior* based on an *OrderConstraint*). The predicate *matches* on an *orderConstraint* requires that the *allValues* property of the *dailyBehavior* class for *user* follows the order specified by the *valueSet* and *mandatory* properties of the *orderConstraint* for the *actionAttribute* attribute.

An instance of this constraint can be defined in the *instance file*, as follows:

```
<hosp:orderConstraint rdf:ID="JaneOrderConstraint">
  <hosp:actionAttribute rdf:resource="requested profile">
  <hosp:valueSet rdf:resource="Jane-Sara-Neo-Nancy">
  <hosp:mandatory rdf:resource="TTFT">
  <hosp:minLength rdf:resource=3>
</hosp:orderConstraint>
```

As another example the *AssociationConstraint* (defined in Appendix A) can be described in a similar manner. The properties required to be defined for the *associationConstraint* class are: *sourceAttribute, sourceAttributeValue, targetAttribute,* and *targetAttributeValueSet*.

References

[1] J. Al-Muhtadi et al., Cerberus: A context-aware security scheme for smart spaces, in: *PERCOM'03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, 2003, p. 489.

[2] R.J. Anderson, A security policy model for clinical information systems, in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 30–42.

[3] J.E. Bardram, Applications of context-aware computing in hospital work: Examples and design principles, in: *SAC'04: Proceedings of the 2004 ACM Symposium on Applied Computing*, 2004, pp. 1574–1579.

[4] R. Bhatti et al., A trust-based context-aware access control model for web-services, in: *ICWS'04: Proceedings of the IEEE International Conference on Web Services*, 2004, p. 184.

[5] R. Bhatti et al., Engineering a policy-based system for federated healthcare databases, *IEEE Trans. Knowl. Data Eng.* **19** (2007), 1288–1304.

[6] B. Blobel, Trustworthiness in distributed Electronic Healthcare Records – basis of shared care, in: *Computer Security Applications Conference*, Vol. 17, 2001, pp. 433–441.

[7] B. Blobel, Authorization and access control for electronic health record systems, *International Journal of Medical Informatics* **73** (2004), 251–257.

[8] Canada Health Infoway, EHRi Privacy and Security Use Cases, November 2004.

[9] Canada Health Infoway, EHR Privacy and Security Requirements, v1.1, February 2005.

[10] Canada Health Infoway, EHRi Privacy and Security Conceptual Architecture, v2, June 2005.

[11] Canada Health Infoway, iEHR Scope and Package Tracking Framework, IE50102-PM99, April 2007.

[12] Canada Health Infoway, Example Privacy & Security Audit Logging Requirements, v0.2, February 2008.

[13] Canadian Nurses Association (CNA) website, www.cna-aiic.ca.

[14] R. Chandramouli, A framework for multiple authorization types in a healthcare application system, in: *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001, pp. 137–148.

[15] CNA, Advances nursing practice – A national framework, April 2002.

[16] E. Damiani et al., New paradigms for access control in open environments, in: *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology*, 2005, pp. 540–545.

[17] N. Damianou et al., The ponder Policy specification language, in: *POLICY'01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, 2001, pp. 18–38.

[18] T. Elrad et al., Aspect-oriented programming: Introduction, *Communications of the ACM* **44**(10) (2001), 29–32.

[19] M. Evered and S. Bögeholz, A case study in access control requirements for a health information system, in: *Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, Vol. 32, 2004, pp. 53–61.

[20] A. Ferreira et al., Access control: How can it improve patients' healthcare?, *Study in Health Technology and Informatics* **127** (2007), 65–76.

[21] C.K. Georgiadis et al., Flexible team-based access control using contexts, in: *SACMAT'01: Proceedings of the Sixth ACM Symposium on Access Control models and Technologies*, 2001, pp. 21–27.

[22] L. Giuri and P. Iglio, Role templates for content-based access control, in: *RBAC'97: Proceedings of the Second ACM Workshop on Role-Based Access Control*, 1997, pp. 153–159.

[23] M. Hafner et al., Modeling and enforcing advanced access control policies in healthcare systems with Sectet, in: *Models in Software Engineering*, LNCS, Vol. 5002, Springer, 2008, pp. 132–144.

[24] Health Insurance Portability and Accountability Act (HIPAA), available at: www.hipaa.org.

[25] HIPAA, Security standards: Technical safeguards, version 2, March 2007.

[26] HL7, Healthcare requirements for emergency access, January 2009.

[27] HL7, HL7 Clinical Document Architecture, Release 2.0, August 2004.

[28] HL7, RBAC healthcare scenarios, v2, November 2005.

[29] HL7, RBAC role engineering process, v1.1, November 2005.

[30] HL7, HL7 healthcare scenario roadmap, v2.2, September 2006.

[31] HL7, HL7 RBAC constraint catalog, v1.1, May 2008.

[32] HL7, RBAC healthcare permission catalog, v4.1, January 2010.

[33] J. Hu and A.C. Weaver, Dynamic, context-aware security infrastructure for distributed healthcare applications, in: *Proceedings of the First Workshop on Pervasive Privacy Security, Privacy, and Trust*, 2004.

[34] R.J. Hulsebosch et al., Context sensitive access control, in: *SACMAT'05: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, 2005, pp. 111–119.

[35] P. Hung, Towards a privacy access control model for e-healthcare services, in: *PST'05: Third Annual Conference on Privacy, Security and Trust*, 2005.

[36] P.C.K. Hung et al., Research issues of privacy access control model for mobile ad hoc healthcare applications with xacml, in: *AINA'07: Advanced Information Networking and Applications Workshops*, 2007, pp. 582–587.

[37] Integrating the Healthcare Enterprise (IHE) official website, www.ihe.net.

[38] Integrating the Healthcare Enterprise, IHE IT infrastructure technical framework white paper – Access control, Revision 1.3, September 2009.

[39] Integrating the Healthcare Enterprise, Integration profile, Revision 7, in: *IHE IT Infrastructure Technical Framework*, Vol. 1, August 2010.

[40] J.H. Jahnke, Toward context-aware computing in clinical care, in: *OOPSLA Workshop on Building Software for Pervasive Computing*, 2005.

[41] J.H. Jahnke et al., Context-aware information services for health care, *Revue d'Intelligence Artificielle* **19** (2005), pp. 459–478.

[42] W. Jih et al., Context-aware access control on pervasive healthcare, in: *MAM'05: Mobility, Agents, and Mobile Services*, 2005, pp. 21–28.

[43] M.H. Kang et al., Access control mechanisms for inter-organizational workflow, in: *SACMAT'01: Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, 2001, pp. 66–74.

[44] B.J. Kozier and G. Erb, *Techniques in Clinical Nursing*, 5th edn, Prentice Hall, Upper Saddle River, NJ, 2004, pp. 887–889.

[45] C.J. Kuo and P. Humenn, Dynamically authorized role-based access control for secure distributed computation, in: *XMLSEC'02: Proceedings of ACM Workshop on XML Security*, 2002, pp. 97–103.

[46] X. Li et al., Fine-granularity access control in 3-tier laboratory information systems, in: *IDEAS'05: Proceedings of the 9th International Database Engineering & Application Symposium*, 2005, pp. 391–397.

[47] V. Murray, *Nursing in Ontario*, Queen's Printer, 1970, pp. 9–16.

[48] G. Neumann and M. Strembeck, A scenario-driven role engineering process for functional RBAC roles, in: *SACMAT'02: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, 2002, pp. 33–42.

[49] U. Nitsche et al., Realization of a context-dependent access control mechanism on a commercial platform, in: *IFIP/Sec'98: Proceedings of the Fourteenth International Information Security Conference*, 1998, pp. 160–170.

[50] Nurse Practitioner's Association of Ontario (NPAO) website, www.npao.org.

[51] Ponder toolkit website, http://ponder2.net/.

[52] P.A. Potter and A.G. Perry, *Basic Nursing: Theory And Practice*, 3rd edn, Mobsy – Year Book, 1994, pp. 5–8, 97–109.

[53] Rei website, http://rei.umbc.edu/.

[54] A. Samuel, Context-aware access control policy engineering for electronic health records, in: *Research Seminar at CIMIC*, 2007.

[55] D. Schwartmann, An attributable role-based access control for healthcare, in: *ICCS'04: International Conference on Computational Science*, 2004, pp. 1148–1155.

[56] A. Toninelli et al., A semantic context-aware access control framework for secure collaborations in pervasive computing environments, in: *ISWC'06: Fifth International Semantic Web Conference*, 2006, pp. 473–486.

[57] F.K. Ueckert and H.U. Prokosch, Implementing security and access control mechanisms for an electronic healthcare record, in: *AMIA Annual Symposium Proceeding Archive*, 2002, pp. 825–829.

[58] T. Verhanneman et al., Adaptable access control policies for medical information systems, in: *Distributed Applications and Interoperable Systems*, LNCS, Vol. 2893, Springer, Berlin/Heidelberg, 2003, pp. 133–140.

[59] K. Wrona and L. Gomez, Context-aware security and secure context-awareness in ubiquitous computing environments, in: *Autumn Meeting of Polish Information Processing Society Conference Proceedings*, 2005, pp. 255–265.

[60] M.H. Yarmand et al., Behavior-based access control for distributed healthcare environment, in: *IEEE Symposium on Computer-Based Medical Systems*, 2008, pp. 126–131.

[61] S.S. Yau et al., Situation-aware access control for service-oriented autonomous decentralized systems, in: *ISADS'05: Proceedings of Autonomous Decentralized Systems*, 2005, pp. 17–24.

[62] G. Zhang and M. Parashar, Dynamic context-aware access control for grid applications, in: *GRID'03: Proceedings of the Fourth International Workshop on Grid Computing*, 2003, p. 101.

[63] L. Zhang et al., A role-based delegation framework for healthcare information systems, in: *SACMAT'02: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, 2002, pp. 125–134.