The ISC Int'l Journal of Information Security
ISeCure

July 2013, Volume 5, Number 2 (pp. 119–140)
http://www.isecure-journal.org

A Semantic-Aware Role-Based Access Control Model for Pervasive Computing Environments

Seyyed Ahmad Javadi [1] and Morteza Amini [1,*]
[1] *Data and Network Security Lab (DNSL), Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

ARTICLE INFO.

Article history:
Received: 17 October 2013
Revised: 8 February 2014
Accepted: 12 March 2014
Published Online: 20 March 2014

Keywords:
Access Control, Non-Monotonic Logic, Pervasive Computing Environment, Context-Aware.

ABSTRACT

Access control in open and dynamic Pervasive Computing Environments (PCEs) is a very complex mechanism and encompasses various new requirements. In fact, in such environments, context information should be used in access control decision process; however, it is not applicable to gather all context information completely and accurately all the time. Thus, a suitable access control model for PCEs not only should be context-aware, but also must be able to deal with imperfect context information. In addition, due to the diversity and heterogeneity of resources and users and their security requirements in PCEs, supporting exception and default policies is a necessary requirement. In this paper, we propose a Semantic-Aware Role-Based Access Control (SARBAC) model satisfying the aforementioned requirements using $MKNF^+$. The main contribution of our work is defining an ontology for context information along with using MKNF$^+$ rules to define context-aware role activation and permission assignment policies. Dividing role activation and permission assignment policies into three layers and using abstract and concrete predicates not only make security policy specification more flexible and manageable, but also make definition of exception and default polices possible. The expressive power of the proposed model is demonstrated through a case study in this paper.

© 2013 ISC. All rights reserved.

1 Introduction

Access control policy determines what, where, when, and how subjects can access databases, web services, electronic devices, and other resources. Access control in Pervasive Computing Environments (PCEs) imposes some new requirements which are not covered by traditional access control models. In fact, important features of resources in PCEs such as context-awareness and heterogeneity require an access control model for PCEs to be context-aware as well as to have a high expressive policy specification language. Logics can play a crucial role in addressing such requirements. Moreover, using logics in access control models has advantages including clean foundations, flexibility, expressiveness, declarativeness, and inference capability [4]. There are two classes of logics that can be considered for this purpose:

(1) Monotonic logics, where current conclusions are not invalidated by adding new information and premises. Classical logics such as propositional and first-order logic are monotonic.
(2) Non-monotonic logics, where some of the current conclusions may be retracted by adding new information and premises.

* Corresponding author.
Email addresses: ajavadi@ce.sharif.edu (S. A. Javadi), amini@sharif.edu (M. Amini)
ISSN: 2008-2045 © 2013 ISC. All rights reserved.

Although non-monotonic logics are more complex than the monotonic ones, some particular characteristics of PCEs motivate us to leverage the non-monotonic logics for access control in such environments. The main related characteristics of such environments are:

- In PCEs, it is impossible to gather all context information completely and accurately all the time [16]. For example, a user's location might be unknown due to the communication failure, the sensor failure, or any other types of failures, or the information provided by sensors may be inconsistent. Furthermore, access control system knowledge about the environment might be limited and inaccurate.
- Dynamicity of PCEs and need for easy management of access control rules necessitate the support of exceptions in an access control model for PCEs. If definition of exceptions is supported by an access control system, new and probably specific authorizations can be defined using exceptions without changing the existing conflicting general access control rules [4].
- To ensure the completeness of access policies, a default policy is required when neither permission nor prohibition about a request is inferred [4]. In addition, a conflict resolution strategy is needed to deal with conflicts.

Additionally, context-awareness should be considered as an important requirement. Therefore, the appropriate access control model for PCEs not only must use a strong context modeling approach, but also it must be able to make access decisions in the presence of imperfect context information and support exception and default security policies as well as conflict resolution strategy.

According to Figure 1, DL is suitable for modeling context information and ASP [11] is an appropriate tool for provision of non-monotonic requirements. Therefore, a hybrid logic, combining DL and ASP together, would be an appropriate candidate for our purpose. In this paper, $MKNF^+$ [14], as a combination of ASP and DL, is used as a formal basis for the security policy specification language in our proposed access control model named the Semantic-Aware Role-Based Access Control (SARBAC) model.

As Figure 2 shows, in SARBAC, the security policy is broken into the Role Activation Policy (RAP) and Permission Assignment Policy (PAP). Both policies are defined using logical rules and predicates which brings us easy management and inference capability. Our main contributions are as follows:

- We classify both RAP and PAP into three layers, as depicted in Figure 3. In fact, each regular security rule can be considered as an exception to the default rules. In addition, each exception security rule can be considered as an exception to regular security policy rules. Therefore, both exception and default rules can be considered as exceptions which can be specified in $MKNF^+$ by its only non-monotonic feature, i.e. negation-as-failure. This approach, is the main contribution of this paper for dealing with non-monotonic aspects of access control in PCEs.
- In order to make the specification of RAP and PAP easier, we proposed the idea of using abstract and concrete predicates as well as system rules. In fact, abstract predicates are used for the specification of RAP and PAP by authorities. On the other hand, concrete predicates are the intermediate predicates used for inference of the final decision about the role activation and permission assignment. In truth, these predicates are used by the access control system not by authorities. In addition, a set of predefined rules in each layer of RAP and PAP, which are called system rules, are defined. These rules are a fixed part of the proposed model and responsible for enforcing different inheritance as well as translating abstract predicates to the concrete ones.

To improve the applicability of the model, the idea in [10] for dividing the context information into the long-term and short-term context is used. Therefore, the role activation policy is defined based on the long-term contextual conditions and the permission assignment policy is defined based on the short-term contextual conditions. When a session request is received, a set of activated roles are assigned to the session based on the long-term context information. After that, the user is permitted to perform the operations, which their permissions have been assigned dynamically to the session's roles during the session based on the short-term context information.

The rest of this paper is organized as follows. Section 2 analyses the non-monotonic requirements of access control in PCEs. Narrative as well as formal specification of our proposed model, named SARBAC, is represented in Section 3. In Section 4, the policy specification language of the proposed model is described. Section 5 describes the access control procedure. A case study for clarifying the applicability of the approach is mentioned in Section 6. Section 7 surveys the related work. Finally, Section 8 concludes the paper and draws some future directions.

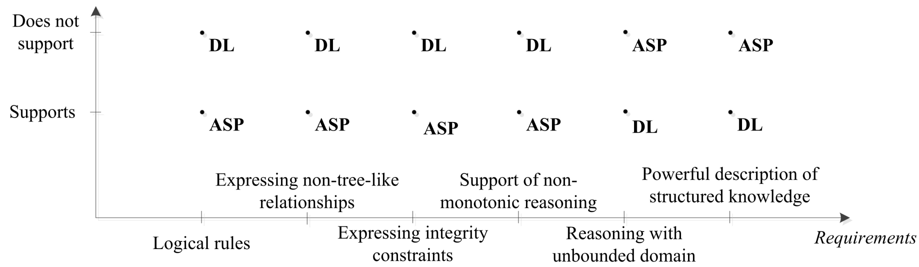

Figure 1. Shortcomings and strengths of DL and ASP [14].

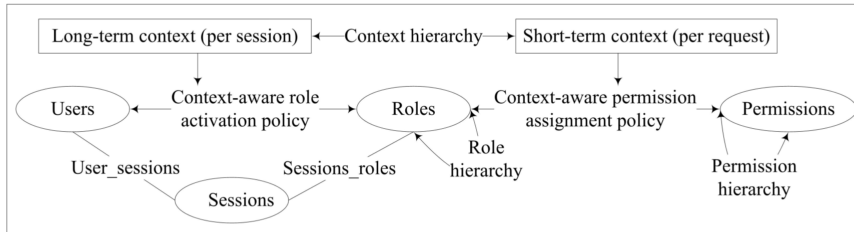

Figure 2. Overall view of SARBAC.

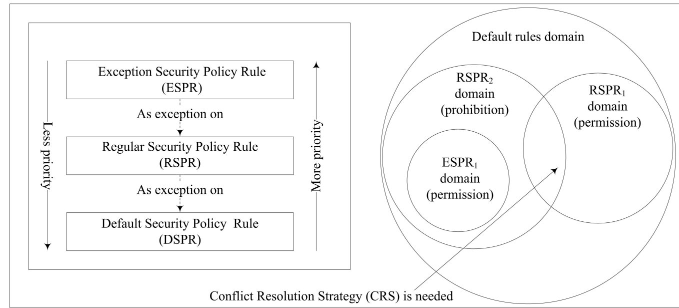

Figure 3. Different types of security policy rules

2 Analysis of Non-monotonic Requirements and Our Proposed Solution

Before explanation of our proposed model in more details, justification of ASP capabilities for addressing the mentioned requirements is necessary. Therefore, three subsequent sections explain and compare capabilities of some of the existing non-monotonic logics including ASP from this point of view. The notation used throughout the paper is stated in Table 1.

2.1 Dealing with Imperfect Context Information

One possible way for dealing with incomplete context information is to make decision based on the knowledge which is known to be true at the decision time [16]. The knowledge includes all information which can be inferred consistently from the security knowledge base. Therefore, it is expected that decisions are made temporarily and retracted when new information is added to the security knowledge base. Non-monotonic logics are used for modeling such de-feasible inferences. Examples of non-monotonic logics, which are used for this purpose, are Default logic [17], auto-epistemic logic programming [3], and ASP.

Default logic was the first non-monotonic logic used as policy specification language by Woo and Lam [20]. L^k as a knowledge base formal language has been proposed to specify authorization domains with incomplete information by Bai [3]. Three types of propositions namely initial, objective, and subjective propositions are defined for this purpose. The semantics of L^k is defined based on the world view semantics of epistemic logic programs. Table 2 shows the propositions and their translation to epistemic logic program, in which ϕ is a conjunctive or disjunctive fact expression, and ψ, β, and γ are conjunctive fact expressions.

ASP is an appropriate decidable logic, which extends the general logic program to obtain capability of modeling incomplete information. ASP supports negation-as-failure. Also in this logic, a truth value of a ground proposition can be true, false, or unknown. Noorollahi and Fallah [16] described how negation-as-failure provides the ability to deal with different types

Table 1. Notations used in the rules represented in the paper.

su:	is a subject	ro:	is a role
ac:	is a action	av:	is a activity
ob:	is a object	v:	is a view
u:	is a user	c:	is a context
ltc:	is a long-term context	stc:	is a short-term context
Act:	is a abbreviation of “Activate”	Deact:	is a abbreviation of “Deactivate”

Table 2. Three propositions in L^k and their translation to epistemic logic program. [3]

Proposition name	Proposition form	Translation
Initial	*initially* ϕ	$\rightarrow \phi$
Objective	ϕ *if* ψ *with absence* γ	$\psi, \mathbf{not}\ \gamma \rightarrow \phi$
Subjective	ϕ *if* ψ *with absence* γ *knowing* β	$\psi, \mathbf{not}\ \gamma, \mathbf{K}\ \beta \rightarrow \phi$
Subjective	ϕ *if* ψ *with absence* γ *not knowing* β	$\psi, \mathbf{not}\ \gamma, \neg\mathbf{K}\ \beta \rightarrow \phi$

of imperfect context information. Table 3 compares qualitatively three main approaches for dealing with imperfect context information from the computational complexity and expressiveness perspectives.

2.2 Access Control Exceptions

There are two types of exceptions namely *negative exceptions* and *positive exceptions* (e.g. see strong exceptions on weak authorizations in Orion model [15]).

Monotonic logics such as propositional logic can not address this requirement appropriately. For example, suppose the following security policy [20]:

(1) Subject A is not permitted to write on file X.
(2) The subject, who is not permitted to write on file X, is also not permitted to read X except it belongs to group G or have permission to read file Y.

A simple specification of the policy is as the follows:

$R_1 : Prohibition(A, Write, X)$
$R_2 : Prohibition(su, Write, X) \wedge \neg(IsMemebrOf(su, G) \vee Permission(su, Read, Y)) \rightarrow Prohibiton(su, Read, X)$
$R_3 : Prohibition(su, Write, X) \wedge (IsMemebrOf(su, G) \vee Permission(su, Read, Y)) \rightarrow Permission(su, Read, X)$

Although R_1, R_2, and R_3 specify the mentioned security policy, they are inflexible and error-prone; because we should specify each condition that the given exception does not have conflict with, in the premise of a separate rule (e.g. here rules R_2 and R_3). Also, if neither $\neg IsMemebrOf(su, G)$ nor $\neg Permission(su, Read, Y)$ can be inferred, the read privilege of file X for su is not prohibited explicitly. In other words, in this approach, in presence of an exception we cannot have a complete set of rules that decide (infer *Prohibition* or *Permission*) in each possible condition. Such a situation is more challenging in access control for PCEs, which faces with incomplete information.

A better way to support exception is specifying general security policies as a set of general rules (e.g. using default rules) and defining subsequent desired exceptions using negative/positive exceptions. The rules represented in the third and fourth rows of Table 4, show how exceptionable general rules can be defined using Default logic. Consequently, the rules represented in the first and second rows of Table 4, can be used to define exceptions to the above general rules. For example, the previous security policy can be defined as follows:

$R_1 : Prohibition(A, Write, X),$

$R_2 : \dfrac{Prohibition(su, Write, X) : \neg Exception^{+}(su, Read, X)}{Prohibition(su, Read, X)},$

$R_3 : IsMemberOf(su, G) \rightarrow Exception^{+}(su, Read, X),$

$R_4 : Permission(su, Read, Y) \rightarrow Exception^{+}(su, Read, X).$

It is obvious that the new exceptions can be added easily, and due to the semantics of Default logic if an exception is not derived explicitly, it is assumed not to be hold, which is an suitable property. In other words, in the semantics of Default logic the predicate $\neg Exception^{+}(su, Read, X)$ is consistent with the security knowledge base if $Exception^{+}(su, Read, X)$ does not inferred from it. Accordingly, the read privilege of file X for subjects who are not known explicitly as exceptions (for example due to imperfect context information in PCEs) is prohibited explicitly, which is truly consistent with the concept of exception and provides more appropriate level of security than the previous approach.

Using negation-as-failure in ASP is another approach to define exceptions. The general rules specified in the fifth and sixth rows of Table 4 determines

Table 3. Three approaches used for dealing with imperfect information.

Approach	Computational complexity	Expressiveness
Default logic	undecidable	very high
Auto-epistemic logic programming	high	high
Answer set programming	medium	medium

Table 4. Exception definition using default logic and ASP.

Negative exception	$Exception\ Conditions \rightarrow Exception^{-}(su, ac, ob)$
Positive exception	$Exception\ Conditions \rightarrow Exception^{+}(su, ac, ob)$
Default logic	$\frac{[Preconditions] : \neg Exception^{-}(su, ac, ob), ...}{Permission(su, ac, ob)}$
	$\frac{[Preconditions] : \neg Exception^{+}(su, ac, ob), ...}{Prohibission(su, ac, ob)}$
ASP	$[Preconditions], \textbf{not}\ Exception^{-}(su, ac, ob) \rightarrow Permission(su, ac, ob)$
	$[Preconditions], \textbf{not}\ Exception^{+}(su, ac, ob) \rightarrow Permission(su, ac, ob)$

how exceptions can be defined using ASP. As an example, the previous example policy can be defined as:

$R_1 : Prohibition(A, Write, X),$

$R_2 : Prohibition(su, Write, X), \textbf{not}\ Exception^{+}(su, Read, X) \rightarrow Prohibition(su, Read, X),$

$R_3 : IsMemberOf(su, G) \rightarrow Exception^{+}(su, Read, X),$

$R_4 : Permission(s, Read, Y) \rightarrow Exception^{+}(su, Read, X).$

2.3 Default security policy

Three main approaches for the definition of default security policies are as follows:

(1) The first approach is to use non-monotonic logics which support default rule. For example, suppose a default access control rule as "by default, students are privileged to access the Internet". It can be defined by a default rule as follows:

$$\frac{Student(s) : \neg Prohibition(s, Access, Internet)}{Permission(s, Access, Internet)}.$$

This rule means that if the subject s is a student and he is not prohibited to access Internet explicitly (i.e. $\neg Prohibition(s, Access, Internet)$ is consistent with the security knowledge base), he can access the Internet. The first two general default access control rules shown in Table 5 can be used for this purpose.

(2) Using negation-as-failure in ASP is the second approach. The last two general default rules shown in Table 5 can be defined in ASP. The third rule means that if no responses (neither permission nor prohibition) are inferred for the request (su, ac, ob) and a set of prerequisite conditions are satisfied, su will have permission to do ac on ob. In fact, the closed-world assumption is applied to $Permission$ and $Prohibition$ predicates using *not* as negation-as-failure operator. The next rule is interpreted similarly. The previous example of default access rule can be defined in this approach as follows:

$$\textbf{not}\ Permission(su, Access, Internet), \quad (1)$$
$$\textbf{not}\ Prohibition(su, Access, Internet), Student(su) \rightarrow Permission(su, Access, Internet).$$

In this approach, we can specify different default actions for different situations (or contextual states). In contrast to default logic, answer set programming is decidable.

(3) Although using logics such as default logic and answer set programming as policy specification language brings high expressiveness, due to their high complexity, most access control models restrict themselves to a general open/close policy. In this approach, when no response about a request is inferred, the system permits/denies the requested access whether open/close policy is determined. Low complexity and applicability are the benefits of this approach and low expressiveness is its main weakness.

2.4 Our proposed approach to use ASP

Our analysis as well as other research such as [16] and [4] show that, ASP is an appropriate logic for provision of the requirements in PCEs. However, ASP is not as suitable as Description Logic (DL) for modeling context information. In this paper, we use $MKNF^{+}$ to propose a powerful context-aware access control model for PCEs.

Each $MKNF^{+}$ knowledge base is a pair $K = (O, P)$, where O is a DL knowledge base and P is a program(finite set of $MKNF^{+}$ rules). Predicates defined in O are called DL-predicates and other predicates are

Table 5. Default access control rules definition using Default and ASP logics.

Default logic	$\frac{PCs : \neg Prohibition(su, ac, ob), ...}{Permission(su, ac, ob)}$
	$\frac{PCs : \neg Permission(su, ac, ob), ...}{Prohibition(su, ac, ob)}$
ASP	$\textbf{not}\ Permission(su, ac, ob), \textbf{not}\ Prohibition(su, ac, ob), PCs \rightarrow Permission(su, ac, ob)$
	$\textbf{not}\ Permission(su, ac, ob), \textbf{not}\ Prohibition(su, ac, ob), PCs \rightarrow Prohibition(su, ac, ob)$

called non-DL-predicates. DL-predicates are unary or pair predicates but non-DL-predicates are not bounded. Moreover, two types of modal atoms namely **K**-atom and **not**-atom are defined in this formalism. **K**-atom is denoted by $\mathbf{K}A$ and **not**-atom is denoted by $\mathbf{not}A$. The structure of an $MKNF^+$ rule is as follows:

$$B_1, ..., B_n \rightarrow H_1 \vee ... \vee H_m.$$

Where, B_i can be a non-modal predicate, a **K**-atom, or a **not**-atom, whereas, H_i would be either a non-modal predicate or a **K**-atom. To preserve decidability of $MKNF^+$, the DL-safety restriction must be applied; each variable in a rule should appear in the body of the rule in some non-DL-K-atom. Appendix A provides more detailed explanation about this logic. In the rest of this paper, DL-atom names are indicated by initial capital words and non-DL-atom names are demonstrated by lower case words. In addition, variables are represented by lowercase names, and constants are represented by initial capital words.

3 Access Control Model

This section represents our proposed model which addresses the specified access control requirements. We first describe an overall framework for access control in PCEs, then we describe our proposed model based on the overall framework.

3.1 Overall Framework

Similar to the approach that we proposed for semantic-aware open environments in [1], a PCE can be divided into a number of security domains in our framework (see Figure 4). Each security domain includes the following components:

- A set of under-protection resources, which are registered in the security domain. Resources are distributed in the environment.
- An authority who specifies the security policy of the domain for the resources (objects) registered in the domain. Security policy consists of Role activation Policy (RAP) and Permission Assignment Policy (PAP). RAP is a context-aware policy, which uses long-term context information to determine the roles that must be activated for a user in a session. PAP is a context-aware policy, which uses short-term context information for assigning permissions to the activated roles in the session. Using long-term context information for RAP and short-term context information for PAP, increases the applicability and performance of the access control system developed based on our model. In fact, since long-term contextual constraints used in a role activation rule are verified only before establishing a session, lower overhead is enforced to the access control system.
- A security agent, which infers and enforces security policy rules (specified by the authority). The security agent receives the session requests from a user and activates a set of roles for the user based on the domain's role activation policy. After determination of active roles, the security agent sets up a session for the user based on the activated roles.

3.1.1 Security Agent Architecture

For implementing a security agent in a security domain, we suggest the architecture shown in Figure 5. The main components of the security agent are as follows:

- Policy Administration Point (PAP): This unit provides an interface for the authorities to state their security policy rules.
- Security Knowledge Base (SKB): In our proposed model, security related context information is modeled as ontology called Security Ontology (SO). Also, Security Policy (SP), including RAP and PAP, is specified using $MKNF^+$ predicates and rules. In fact, SKB is an $MKNF^+$ knowledge base where SO and SP construct the DL knowledge base and the logic program in an $MKNF^+$ knowledge base respectively.
- Context Management Point (CMP): CMP provides a framework for management of required context information. CMP gathers the information from the context sensors and other sources and has the responsibility of keeping the SO updated and accurate.
- Role-Activation Policy Decision Point (RAPDP): RAPDP uses an $MKNF^+$ inference engine for role activation decision making. It sends a set of activated role names in response to the session

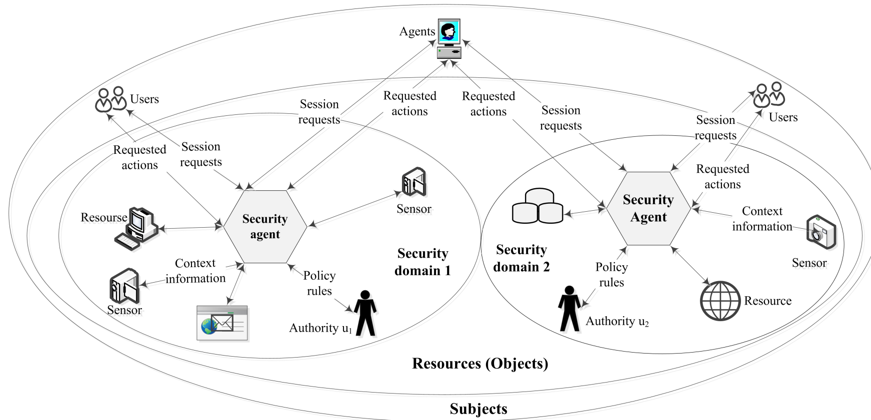

Figure 4. Ovaerall access control framework for a PCE [1].

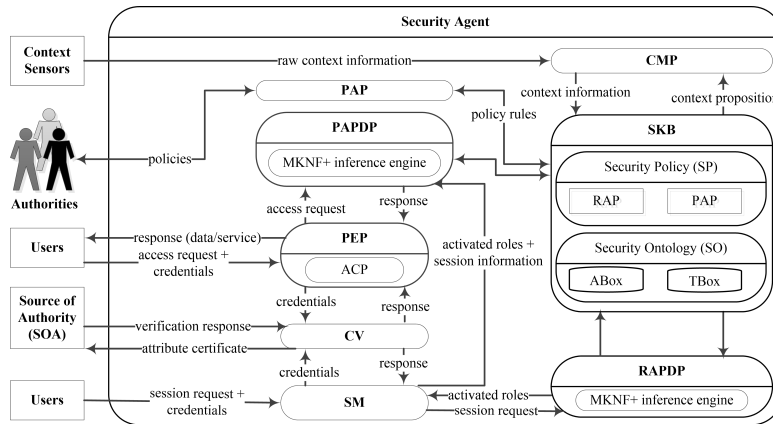

Figure 5. Architecture of a security agent of a security domain.

manager's request.

- Session Manager (SM): SM receives the session requests and conducts needed actions for setting up a session.
- Permission Assignment Policy Decision Point (PAPDP): PAPDP uses an $MKNF^+$ inference engine to make decision about granting/revoking permissions to the session roles.
- Policy Enforcement Point (PEP): PEP receives an access request from a subject in a session and determines the response using PAPDP.
- Credential Verifier (CV): This unit verifies the validity of the provided credentials (in an access request) using a source of authority (SOA).

The proposed architecture enables the security agent to verify the users' certificates, infer the activated roles for the users' sessions and the assigned permissions for the roles dynamically, and enforce them. The details of the required access control procedure, which should be followed by the security agent, are described in Section 5.

3.1.2 Security Knowledge Base

As shown in Figure 6, in our proposed model, access control elements and context information are modeled in an ontology. However, it is not applicable to cover all entities and context information in different domains in a predefined ontology. Thus, the context model is divided into the upper-level ontology and the domain-specific ontology by Wang *et al.* [19] for resolving the issue. We follow the same idea, and thus, our proposed ontology consists of two layers:

(1) The upper-level ontology is a high-level ontology, which represents the main general entities and their attributes in a security domain. Such entities can be used in different domains and are required for our proposed access control model.
(2) The domain-specific ontology is a detailed ontology, which describes domain-specific concepts and their relationships existing in a PCE, in addition to the main concepts (specified in the upper-level ontology). In contrast to the upper-level ontology which is fix for all PCEs, the domain-specific ontology might be defined and customized for each PCE separately.

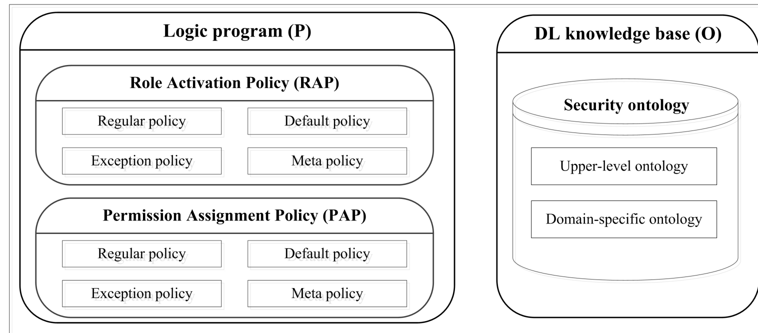

Figure 6. The structure of the security knowledge base in our proposed model.

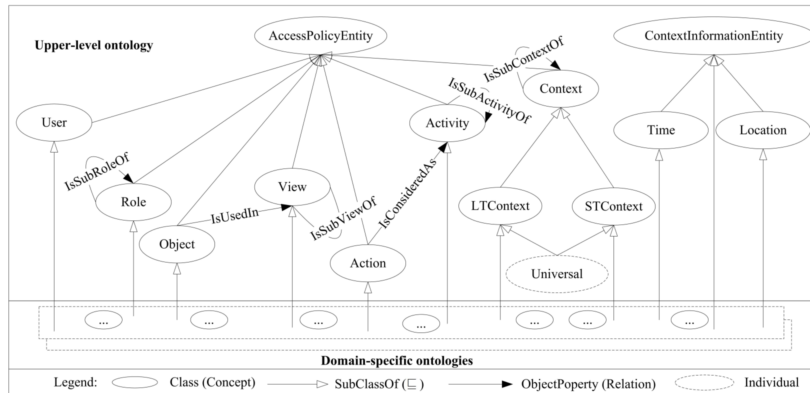

Figure 7. Partial definition of the upper-level ontology.

Table 6. Entities (concepts) of upper-level ontology.

Entity	Description
User	Representing active entities
Role	The entity used to link users to their access privileges. In fact privileges are granted (revoked) to (from) the roles instead of users.
Object	Representing inactive (passive) entities (such as a data file, an email), which are accessed by users.
View	Representing a group of objects on which the same security rules apply.
Action	Computer actions such as "read" and "write".
Activity	A group of actions that follow the same principles.
LTContext	Used to specify the concrete long-term circumstances where security domains activate/deactivate roles for users
STContext	Used to specify the concrete short-term circumstances where security domains grant (revoke) permissions to (from) roles for performing activities on views.

Figure 7 depicts the upper-level ontology consisting of the main concepts in SARBAC model represented in Table 6. In order to make our proposed model stronger, the concepts of view and activity represented in OrBAC model [12] are used. Table 7 describes the relationships existing in the upper-level ontology. Note that, the two important context information including time and location are considered in the upper-level ontology.

Henceforth in this paper, non-DL-predicates *cie(ContextInformationEntity)* and *ape(AccessPolicy Entity)* are used to apply DL-safety restriction. Predicates *'ape'* and *'cie'* hold for all access policy entities and context information entities respectively. For example, if domain D_1 defines a new role called *Guest*, *Role(Guest)* and *ape(Guest)* are added to the security knowledge base. As another example, if D_1 defines a new location called $Room_2$, $Location(Room_2)$ and $cie(Room_2)$ are added to the security knowledge base. By doing so, the DL-safety restriction is satisfied in

Table 7. Relations (Roles) of upper-level ontology.

Relation	Description
IsSubRoleOf(Role, Role)	The relation that is used for the role hierarchy definition. IsSubRoleOf(ro_1, ro_2) means that ro_1 is the sub-role of ro_2.
IsUsedIn(Object,View)	IsUsedIn(ob,v) means that object ob is used in view v.
IsSubViewOf(View,View)	IsSubViewOf(v_1, v_2) means that view v_1 is the sub-view of view v_2.
IsConsideredAs(action, activity)	IsConsideredAs(ac,av) means that action ac is fell within activity av.
IsSubActivityOf(activity, activity)	IsSubactivityOf(av_1, av_2) means that activity av_1 is the sub-activity of av_2.
IsSubContextOf(Context, Context)	IsSubContextOf(c_1, c_2) means that context c_1 is the sub-context of context c_2, or c_1 is true when c_2 is true.

the specified $MKNF^+$ rules.

3.2 Formal Specification of the Model

Following the narrative description of the proposed model, we formally define the SARBAC model for PCEs in the following.

Definition 1. Semantic-Aware Role-Based Access Control Model (SARBAC): SARBAC for a PCE is defined as a pair $\langle SDS, ACP \rangle$ where:

- $SDS = \{SD_i | SD_i \textit{ is a security domain}\}$ is a set of security domains in the environment. A security domain is formally defined at the end of this section.
- ACP is an Access Control Procedure which is defined in Section 5.

Prior to specifying security domains, we need to define Security Knowledge Base (SKB). In fact, each security domain has its own SKB.

Definition 2. Security Knowledge Base (SKB): An SKB is an $MKNF^+$ knowledge base and it is defined as a pair $\langle SO, SP \rangle$, in which SO is a security ontology defined in DL and SP is a security policy corresponding to the logic program part of $MKNF^+$ knowledge bases.

An ontology (like SO) is defined as a pair $\langle TBox, ABox \rangle$ where:

- $TBox = \{C_i \sqsubseteq C_j\} \cup \{R_i \sqsubseteq R_j\}$ in which C_i and C_j are either primitive or complex concepts specified in DL. Similarly R_i and R_j are relations either primitive or complex specified in DL.
- $ABox = \{C(a) | a^I \in \triangle\} \cup \{R(a,b) | a^I, b^I \in \triangle\}$ where $\triangle$ is the universe of entities (objects).

Note that we suppose, TBox is acyclic and each concept has at most one definition.

Definition 3. Security Ontology (SO): SO is defined as a pair $\langle TBox, ABox \rangle$, where TBox is the terminological box and ABox is the assertional box in $MKNF^+$:

- TBox at least includes the concepts and the relationships which are represented in Table 6 and Table 7, respectively. Furthermore, TBox includes other concepts and relationships defined in the domain-specific ontology of the PCE.
- ABox is the set of assertions about the individuals existing in the PCE and includes *LTContext(Universal)* and *STContext(Universal)* (see 4.1.3 and 4.2.3 for further information).

In what follows, we explain different components of a security policy briefly. More details on the format of the required predicates and policy rules are provided in the next section.

Definition 4. Security Policy (SP): SP is a pair $\langle RAP, PAP \rangle$ where:

- Role Activation Policy(RAP): RAP is defined as five-tuple $\langle LTCD, RRAP, ERAP, DRAP, MetaPolicy \rangle$ where:
  - Long-Term Context Definition (LTCD) is a set of long-term context definition rules (see Section 4.1.1).
  - Regular Role Activation Policy (RRAP) is union of the system rules represented in Table 9 and a set of the Abstract Regular Role Activation (arra) predicates specified by the authorities.
  - Exception Role Activation Policy (ERAP) is union of the system rules represented in Table 10 and a set of Abstract Exception Role Activation (aera) predicates specified by the authorities.
  - Default Role Activation Policy (DRAP) is union of the system rules represented in Ta-

ble 11 and a set of Abstract Default Role Activation (adra) predicates specified by the authorities.
  - MetaPolicy is the set of system rules represented in Table 12.
- Permission Assignment Policy (PAP): A PAP is a five-tuple ⟨*STCD*, *RPAP*, *EPAP*, *DPAP*, *MetaPolicy*⟩ where:
  - Short-Term Context Definition Rules (STCD) is a set of short-term context definition rules (see Section 4.2.1).
  - Regular Permission Assignment Policy (RPAP) is union of the system rules represented in Table 14 and a set of Abstract Regular Permission Assignment (arpa) predicates specified by the authorities.
  - Exception Permission Assignment Policy (EPAP) is union of the system rules represented in Table 15 and a set of Abstract Exception Permission Assignment (aepa) predicates specified by the authorities.
  - Default Permission Assignment Policy (DPAP) is union of the system rules represented in Table 16 and a set of Abstract Default Permission Assignment (adpa) predicates specified by the authorities.
  - MetaPolicy is the set of system rules represented in Table 17.

Definition 5. Security Domain (SD): A security domain is formally defined as a triple $\langle d, O, K\rangle$, where d is the name of the security domain, K is the local SKB of the security domain, and O is the set of under-protection objects registered in the security domain.

4 Security Policy Specification

Figure 8 shows the overall structure of a security policy in SARBAC model. Accordingly, both RAP and PAP consist of three separate policies. Each of them has its own propagation and conflict resolution rules. Although different conflict resolution strategies can be used in these policies, for the sake of simplicity, the Denial-Takes-Precedence (DTP) strategy is used in all of them. Subsequent sections provide more details about the structure.

4.1 Role Activation Policy

Table 8 represents the required predicates for specification of role activation policy. The following subsections describe the predicates.

4.1.1 Regular Role Activation Policy

Definition of required long-term contexts is the first step for definition of regular role activation rules. Long-term context is a type of context that does not change during a session with a high probability. Predicate $ltcd(User, LTContext)$ is used for long-term context definition and $ltcd(u, ltc)$ means that long-term context ltc is true for user u. The conditions under which a specific context holds, are described by an $MKNF^+$ rule of the following form:

$$B_1, ..., B_n \rightarrow \mathbf{K}\, ltcd(u, ltc).$$

The conclusion of the rule is the $ltcd$ predicate **K**-atom and the premises of the rule contain a set of **K**-atoms and **not**-atoms such that DL-saftey restriction is satisfied. For example, consider the following long-term context definition rule defined by domain D_1:

$$\mathbf{K}\, ape(u), \mathbf{K}\, cie(lloc), \mathbf{K}\, HasLogicalLocation(u, lloc), \mathbf{K}\, HasLocationZone(lloc, D_1_Net) \rightarrow \mathbf{K}\, ltcd(u, Internal_User)$$

This rule means that a user is an internal user if he uses an IP address in the location zone of the domain. Remember that, $LTContext(Internal_User)$ and $ape(Internal_User)$ should be added to the security knowledge base, in addition to the above rule. As another example, D_1 can apply the closed-world assumption to the $Internal_User$ context and define the $External_User$ context using negation-as-failure:

$$\mathbf{not}\, ltcd(u, Internal_User) \rightarrow \mathbf{K}\, ltcd(u, External_User)$$

After the definition of long-term contexts, the regular role (de)activations can be defined using the *arra(Role,LTContext,Act/Deact)* abstract predicate by the authorities of a domain. Predicate $arra(r, ltc, Act)$ means that the role r is activated when the long-term context ltc being true. Similarly, *arra(r,ltc,Deact)* means that r must not be activated when the long-term context ltc being true. For example, D_1 can activate the role *Guest* in the Internal_User context using the following predicate:

$$arra(Guest, Internal_User, Act)$$

Domain D_1 may also want to deactivate some specific roles explicitly. For example, D_1 deactivates the *Administrator* role in the External_User context:

$$arra(Administrator, External_User, Deact).$$

A regular role activation policy includes two types of system rules represented in Table 9. These rules have the following two responsibilities:

1. Inheritance enforcement rules: The first two rules and the third rule enforce inheritance over role and context hierarchies respectively.
2. Translation of the abstract-level predicates to the concrete-level ones: Predicate *crra(Role, User, Act/Deact)* is defined for this purpose. Accordingly, *crra(r,u,Act)* means that role r must be activated for user u and *crra(r,u,Deact)* means the opposit. Translation must be done such that the possible conflicts do not transferred to the concrete level. The translation rules in Table 9 simply gives more priority to role-deactivations.

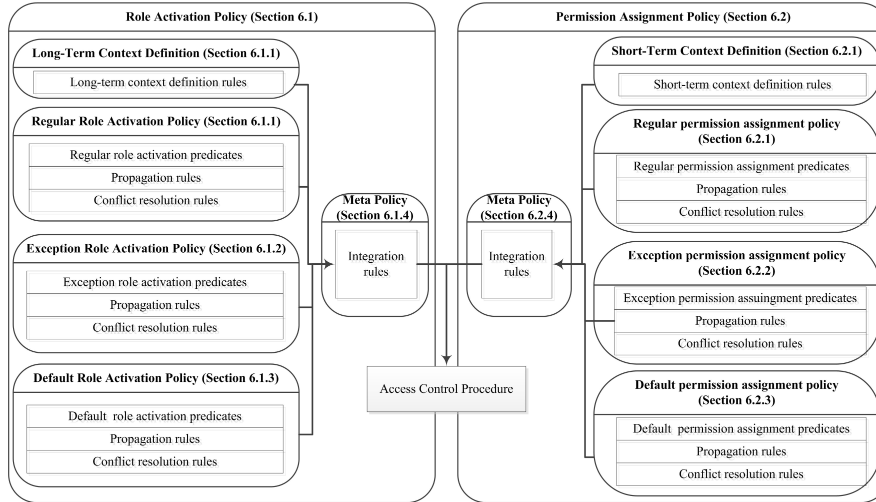

Figure 8. Overall structure of a security policy in the proposed model.

Table 8. Role activation policy predicates.

Type	Description	predicate
Context	Long-Term Context Definition	*ltcd(User, LTContext)*
Abstract	Regular Role activation	*arra(Role, LTContext, Act/Deact)*
	Exception Role activation	*aera(Role, LTContext, Act/Deact)*
	Default Role Activation	*adra(Role, LTContext, Open/Close)*
Concrete	Regular Role Activation	*crra(Role, User, Act/Deact)*
	Exception Role Activation	*cera(Role, User, Act/Deact)*
	Default Role Activation	*cdra(Role, User, Open/Close)*
	Role Activation	*cra(Role, User, Act/Deact)*

Table 9. System rules of a regular role activation policy.

Inheritance over the role hierarchy:
$\mathbf{K}\ arra(ro_1, c, Act), \mathbf{K}\ IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}\ arra(ro_2, c, Act)$
$\mathbf{K}\ arra(ro_2, c, Deact), \mathbf{K}\ IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}\ arra(ro_1, c, Deact)$
Inheritance over the context hierarchy:
$\mathbf{K}\ arra(ro, c_1, d), \mathbf{K}\ IsSubContexOf(c_2, c_1) \rightarrow \mathbf{K}\ arra(ro, c_2, d)$
Translating the abstract predicates to the concrete ones:
$\mathbf{K}\ arra(r, c, Deact), \mathbf{K}\ ltcd(u, c) \rightarrow \mathbf{K}\ crra(r, u, Deact)$
$\mathbf{K}\ arra(r, c, Act), \mathbf{K}\ ltcd(u, c), \mathbf{not}\ crra(r, u, Deact) \rightarrow \mathbf{K}\ crra(r, u, Act)$

4.1.2 Exception Role Activation Policy

In presence of role hierarchy, two general types of exceptions worth to be discussed:

(1) Local exceptions: The kind of exceptions that are defined locally on a role, and they are not inherited over the role hierarchy. The definition of local exceptions impose that the users having sub-roles are not permitted to login as parent roles. Such a limitation contradicts the existence of role hierarchy.

(2) General exceptions: The kind of exceptions that are inherited over the role hierarchy.

In this paper, due to the limitations of local exceptions, we take general exceptions into account.

Each regular role activation rule adds a set of concrete role activations and deactivations to the security knowledge base. So, an exception can be defined in either of following two ways:

(1) Rule-specific exception; which is defined over a specific role activation rule.

Table 10. System rules of an exception role activation policy.

Inheritance over the role hierarchy:
$\mathbf{K}aera(ro_1, c, Act), \mathbf{K}IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}aera(ro_2, c, Act)$
$\mathbf{K}aera(ro_2, c, Deact), \mathbf{K}IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}aera(ro_1, c, Deact)$
Inheritance over the context hierarchy:
$\mathbf{K}aera(ro, c_1, t), \mathbf{K}IsSubContextOf(c_2, c_1) \rightarrow \mathbf{K}aera(ro, c_2, t)$
Translating the abstract predicates to the concrete ones:
$\mathbf{K}\ aera(ro, c, Deact), \mathbf{K}\ ltc(u, c) \rightarrow \mathbf{K}\ cera(ro, u, Deact)$
$\mathbf{K}\ aera(ro, c, Act), \mathbf{K}\ ltc(u, c), \textbf{not}\ cera(ro, u, Deact) \rightarrow\ \mathbf{K}\ cera(ro, u, Act)$

(2) Rule-independent exception; which is defined independently from the defined role activation rules. So, an exception role activation (deactivation) overwrites all other conflicting deactivations (activations).

In this paper, we take rule-independent exceptions into account. In our proposed model, global and rule-independent exceptions are defined using negation-as-failure. Predicate *aera(Role,LTContext,Act/Deact)* is used for exception role activation definition. Thus *aera(r,ltc,Act)* means tha,t role *r* must be activated in the *ltc* context exceptionally and *aera(r,ltc,Act)* means that, *r* must be deactivated in the *ltc* context exceptionally.

Table 10 represents the system rules. The first three rules enforce the propagation of role activation and deactivation through the role and context hierarchies. The predicate *cera(Roel, User, Act/Deact)* is the concrete version of the predicate *aera*. Accordingly, *cera(ro,u,Act)* means that, role *ro* must be activated and *cera(ro,u,Deact)* means *ro* must be deactivated exceptionally. Table 10 shows the translation rules by giving more priority to role-deactivations.

4.1.3 Default Role Activation Policy

In almost all of the previous access control models one of the *Open* or *Close* policies is used as a default policy. However, the default policy can be determined based on the *context*. For example, a domain may prefer to define *Open* policy for the specific roles at working hours and to define *Close* policy for them at non-working hours. The predicate *adra(Role, LTContext, Open/Close)* is used to define the default rules. Covering all roles and users in a domain is an important requirement of default policy. To meet this requirement, we define the *Universal* long-term context, which is true for all users who registered in the domain. Each domain must choose *Open* or *Close* policy for all roles in the *Universal* context. In other words, one of the rules represented in the fourth and fifth row of Table 11 should be added to each RAP. Additionally, all long-term contexts are defined as a sub-context of the *Universal* context.

Table 11 represents the system rules of a default role activation policy. The first two rules enforce the *Universal* context related propositions. Default rules can be inherited over the roles and context hierarchies. Each sub-role inherits *Open* policy from its parent role and each parent role inherits *Close* policy from its sub-roles. For instance, if *IsSubRoleOf(Guest, Administrator)* is true, *adra(Administrator, c, Open)* will result *adra(Guest, c, Open)* as well as *adra(Guest, c, Close)* will result *adra(Administrator, c, Close)*. The last two rules in Table 11 translate abstract default role activation rules to the concrete ones.

4.1.4 Meta Policy

Meta policy is a set of $MKNF^+$ rules, which combines the regular, exception, and default role activation concrete predicates to result the final decision about the activation and deactivation of a specific role for a specific user. The predicate *cra(Role,User,Act/Deact)* is defined for this purpose. If *cra(ro,u,Act)* can be inferred from the security knowledge base, the security agent can activate role *ro* for user *u* in the requested session. Otherwise, the security agent must not activate *ro* for *u* in the requested session. Based on the priorities of the different types of role activation rules, *cra* can be determined by the following general rules:

(1) Exception role activation rules have the highest priority. Therefore, they are translated to the *cra* predicates directly.
(2) Regular role activation concrete predicates are translated to the *cra* predicates, if there exist no opposite exception concrete predicates.
(3) Default role activation rules determine the *cra* predicates for a role and a user which neither *activation* nor *deactivation* is inferred for them from the regular and exception policies.

Table 12 represents the logical rules that enforce the above rules.

Table 11. System rules of a default role activation policy.

Universal context:
$\mathbf{K}\, ace(u), \mathbf{K}\, User(u) \rightarrow \mathbf{K}\, LTContext(u, Universal)$
$\mathbf{K}\, ace(ltc), \mathbf{K}\, LTContext(ltc) \rightarrow \mathbf{K}\, IsSubContextOf(ltc, Universal)$
One of the following rules:
$\mathbf{K}\, ace(ro), \mathbf{K}\, Role(ro) \rightarrow \mathbf{K}\, adra(ro, Universal, Open)$
$\mathbf{K}\, ace(ro), \mathbf{K}\, Role(ro) \rightarrow \mathbf{K}\, adra(ro, Universal, Close)$
Inheritance over the role hierarchy:
$\mathbf{K}adra(ro_1, c, Open), \mathbf{K}IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}adra(ro_2, c, Open)$
$\mathbf{K}adra(ro_2, c, Close), \mathbf{K}IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}adra(ro_1, c, Close)$
Inheritance over the context hierarchy:
$\mathbf{K}adra(ro, c_1, t), \mathbf{K}IsSubContextOf(c_2, c_1) \rightarrow \mathbf{K}adra(ro, c_2, t)$
Translating the abstract predicates to the concrete ones:
$\mathbf{K}\, adra(ro, c, Close), \mathbf{K}\, ltc(u, c) \rightarrow \mathbf{K}\, cdra(ro, u, Close)$
$\mathbf{K}\, adra(ro, c, Open), \mathbf{K}\, ltc(u, c), \mathbf{not}\, cdra(ro, u, Close) \rightarrow \mathbf{K}\, cdra(ro, u, Open)$

Table 12. Role activation meta policy

Exception role activation enforcement:
$\mathbf{K}\, cera(ro, u, t) \rightarrow \mathbf{K}\, cra(ro, u, t)$
Regular role activation enforcement:
$\mathbf{K}\, crra(ro, u, Act), \mathbf{not}\, cera(ro, u, Deact) \rightarrow \mathbf{K}\, cra(ro, u, Act)$
$\mathbf{K}\, crra(ro, u, Deact), \mathbf{not}\, cera(ro, u, Act) \rightarrow \mathbf{K}\, cra(ro, u, Deact)$
Default role activation enforcement:
$\mathbf{not}\, cra(ro, u, Act), \mathbf{not}\, cra(ro, u, Deact), \mathbf{K}\, cdra(ro, u, Open) \rightarrow \mathbf{K}cra(ro, u, Act)$
$\mathbf{not}\, cra(ro, u, Act), \mathbf{not}\, cra(ro, u, Deact), \mathbf{K}\, cdra(ro, u, Close) \rightarrow \mathbf{K}\, cra(ro, u, Deact)$

4.2 Permission Assignment Policy

Table 13 represents different predicates used in the PAP. The predicates and their usage are discussed in the following sections.

4.2.1 Regular Permission Assignment Policy

Definition of required short-term contexts is the first step of defining permission assignment rules. A short-term context is defined as a context that may change during a session frequently. Predicate *stcd(User, action, Object, STContext)* is used for short-term context definition. Intuitively,*stcd(u, ac, ob, stc)* means that short-term context *stc* is true for user *u*, action *ac*, and object *ob*. The required conditions for a specific context are described by the $MKNF^+$ rule similar to the long-term context. For example, the following rule defines short-term context for physical location "L_1".

$$\mathbf{K}\, ace(u), \mathbf{K}\, IsLocated(u, L_1) \rightarrow \mathbf{K}\, stcd(u, ac, ob, L_1)$$

Abstract predicate *arpa(Role, activity, View, STContext, +/−)* is used for regular permission assignment definition. Predicate *arpa(ro, av,v,stc,+)* permits role *ro* to perform activity *av* on view *v* within context *stc*. Similarly, *arpa(ro, av,v,stc,−)* prohibits *ro* from doing *av* on *v* within context *stc*. Regular permission assignment can be inherited over the following hierarchies:

- Role hierarchy: each parent role inherits the permissions of its sub-roles and each sub-role inherits the prohibitions of its parent roles.
- Activity hierarchy: if a permission for performing activity *av* is assigned to a role, the permission is propagated to the all sub-activities of *av*. In addition, prohibition of performing *av* is propagated to the all activities that *av* is a sub-activity of them.
- View hierarchy: the interpretation of inheritance over the view hierarchy is similar to the activity hierarchy.
- Context hierarchy: *IsSubContextOf*(stc_1, stc_2) means that, stc_1 is true whenever stc_2 is true. Therefore, assigning permission in stc_2 implies assigning it in stc_1.

Table 14 represents the system rules of regular permission assignment policy. These rules enforce authorization inheritance over the role, activity, view, and context hierarchies. Furthermore, the last two

Table 13. Predicates required in the specification of the permission assignment policy

Type	Name of predicate, Predicate
Context	Short-Term Context Definition (stcd) $stcd(User, action, Object, STContext)$
Abstract	Regular Permission Assignment (arpa) $arpa(Role, activity, View, STContext, +/-)$
	Exception Permission Assignment (aepa) $aepa(Role, activity, View, STContext, +/-)$
	Default Permission Assignment (adpa) $adpa(Role, activity, View, STContext, +/-)$
Concrete	Regular Permission Assignment (crpa) $crpa(User, action, Object, +/-)$
	Exception Permission Assignment (cepa) $cepa(User, action, Object, +/-)$
	Default Permission Assignment (cdpa) $cdpa(User, action, Object, Open/Close)$
	Permission Assignment (cpa) $cpa(User, action, Object, +/-)$

Table 14. System rules of a regular permission assignment policy.

Rules
Inheritance over the role hierarchy:
$\mathbf{K}\, arpa(ro_1, av, v, stc, +), \mathbf{K}\, IsSubRoleOf(ro_1, ro_2) \rightarrow \mathbf{K}\, arpa(ro_2, av, v, stc, +)$
$\mathbf{K}\, arpa(ro_2, av, v, stc, -), \mathbf{K}\, IsSubRoleOf(ro_1, ro_2) \rightarrow \mathbf{K}\, arpa(ro_1, av, v, stc, -)$
Inheritance over the activity hierarchy:
$\mathbf{K}\, arpa(ro, av_1, v, stc, +), \mathbf{K}\, IsSubactivityOf(av_2, av_1) \rightarrow \mathbf{K}\, arpa(ro, av_2, v, stc, +)$
$\mathbf{K}\, arpa(ro, av_2, v, stc, -), \mathbf{K}\, IsSubactivityOf(av_2, av_1) \rightarrow \mathbf{K}\, arpa(ro, av_1, v, stc, -)$
Inheritance over the view hierarchy:
$\mathbf{K}\, arpa(ro, av, v_1, stc, +), \mathbf{K}\, IsSubViewOf(v_2, v_1) \rightarrow \mathbf{K}\, arpa(ro, av, v_2, stc, +)$
$\mathbf{K}\, arpa(ro, av, v_2, stc, -), \mathbf{K}\, IsSubViewOf(v_2, v_1) \rightarrow \mathbf{K}\, arpa(ro, av, v_1, stc, -)$
Inheritance over the context hierarchy:
$\mathbf{K}\, arpa(ro, av, v, stc_1, t), \mathbf{K}\, IsSubContextOf(stc_2, stc_1) \rightarrow \mathbf{K}\, arpa(ro, av, v, stc_2, t)$
Translation of the abstract level predicates to the concrete ones:
$\mathbf{K}\, arpa(ro, av, v, stc, -), \mathbf{K}\, cra(ro, u, act), \mathbf{K}\, IsConsideredAs(ac, av),$ $\mathbf{K}\, IsUsedIn(ob, v), \mathbf{K}\, stcd(u, ac, ob, stc) \rightarrow \mathbf{K}\, crpa(u, ac, ob, -)$
$\mathbf{K}\, arpa(ro, av, v, stc, +), \mathbf{K}\, cra(ro, u, act), \mathbf{K}\, IsConsideredAs(ac, av),$ $\mathbf{K}\, IsUsedIn(ob, v), \mathbf{K}\, stcd(u, ac, ob, stc), \mathbf{not}\; crpa(u, ac, ob, -) \rightarrow \mathbf{K}\, crpa(u, ac, ob, +)$

rules in the table translate the abstract level predicates to the concrete ones considering the prohibition-takes-precedence conflict resolution strategy. The first translation rule means that user u can perform action ac on object ob in a session, if permission to perform activity av on view v within context stc is granted to the role ro, also ro is activated for u (in the session), ac is considered as av, ob is used in v, stc is true for u, ac, and ob as well as u is not prohibited for this permission (for preventing possible conflicts).

Table 15. System rules of an exception permission assignment policy.

Inheritance over the role hierarchy:
$\mathbf{K}\ aepa(ro_1, av, v, stc, +), \mathbf{K}\ IsSubRoleOf(ro_1, ro_2) \rightarrow \mathbf{K}\ aepa(ro_2, av, v, stc, +)$
$\mathbf{K}\ aepa(ro_1, av, v, stc, -), \mathbf{K}\ IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}\ aepa(ro_2, av, v, stc, -)$
Inheritance over the activity hierarchy:
$\mathbf{K}\ aepa(ro, av_1, v, stc, +), \mathbf{K}\ IsSubactivityOf(av_2, av_1) \rightarrow \mathbf{K}\ aepa(ro, av_2, v, stc, +)$
$\mathbf{K}\ aepa(ro, av_1, v, stc, -), \mathbf{K}\ IsSubactivityOf(av_1, av_2) \rightarrow \mathbf{K}\ aepa(ro, av_2, v, stc, -)$
Inheritance over the view hierarchy:
$\mathbf{K}\ aepa(ro, av, v_1, stc, +), \mathbf{K}\ IsSubViewOf(v_2, v_1) \rightarrow \mathbf{K}\ aepa(ro, av, v_2, stc, +)$
$\mathbf{K}\ aepa(ro, av, v_1, stc, -), \mathbf{K}\ IsSubViewOf(v_1, v_2) \rightarrow \mathbf{K}\ aepa(ro, av, v_2, stc, -)$
Inheritance over the context hierarchy:
$\mathbf{K}\ aepa(ro, av, v, stc_1, t), \mathbf{K}\ IsSubContextOf(stc_2, stc_1) \rightarrow \mathbf{K}\ aepa(ro, av, v, stc_2, t)$
Translation of the abstract level predicates to the concrete ones:
$\mathbf{K}\ aepa(ro, av, v, stc, -), \mathbf{K}\ cra(ro, u, act), \mathbf{K}\ IsConsideredAs(ac, av),$ $\mathbf{K}\ IsUsedIn(ob, v), \mathbf{K}\ stcd(u, ac, ob, stc) \rightarrow \mathbf{K}\ cepa(u, ac, ob, -)$
$\mathbf{K}\ aepa(ro, av, v, stc, +), \mathbf{K}\ cra(ro, u, act), \mathbf{K}\ IsConsideredAs(ac, av),$ $\mathbf{K}\ IsUsedIn(ob, v), \mathbf{K}\ stcd(u, ac, ob, stc), \mathbf{not}\ cepa(u, ac, ob, -) \rightarrow \mathbf{K}\ cepa(u, ac, ob, +)$

4.2.2 Exception Permission Assignment Policy

Predicate *aepa(Role, activity, View, STContext, +/−)* is used for exception permission assignment. In fact *aepa(ro,av,v,stc,+)* permits role *ro* to perform activity *av* on view *v* within short-term context *stc* exceptionally. Also, *aepa(ro,av,v,stc,−)* means appositely. Similar to the regular permission assignment rules, propagation rules and the rules that translate the abstract predicates to the concrete ones are defined in Table 15.

4.2.3 Default Permission Assignment Policy

Predicate *adpa(Role, activity, View, STContext, Open/Close)* is used for the default permission assignment. Similar to the default role activation policy the short-term context *Universal*, which is true for all users, actions, and objects, is defined. Each domain must choose *Open* or *Close* policy for the *Universal* context. Additionally, all short-term contexts are defined as a sub-context of the *Universal* context. Table 16 represents the system rules of default permission assignment policy.

4.2.4 Meta Policy

The meta policy integrates the regular, exception, and default concrete permission assignment to conclude the final decision. Predicate *cpa(User, Action, Object, +/−)* is defined as a final concrete permission assignment policy. The following rules integrate the predicates *crpa*, *cepa*, and *cdpa* similar to the meta policy of role activation policy:

(1) Exception permission assignment rules have the most priority. Therefore, they are translated to the *cpa* predicates directly.
(2) Regular permission assignment concrete predicates are translated to the *cpa* predicates, if there exist no opposite exception concrete predicate.
(3) Default permission assignment rules determines the *cpa* predicates for a user, action, object when neither + nor − is inferred from the regular and exception rules.

Table 17 shows the rules required to enforce the above meta policy.

5 Access Control Procedure

The access control procedure, which is used by the security agent of a domain SD=$\langle d, O, K\rangle$, contains the following three overall steps:

(1) Session establishment: In this step activated roles for a requested session are determined and according to the result, other required actions are done.
(2) Controlling access requests during the session: In this stage, each access request during the session is evaluated and the appropriate response is generated.
(3) Session termination: In this step, assertions or predicates, which are added to the security knowledge base during the session are removed.

Table 16. System rules of a default permission assignment policy.

Universal context:
$\mathbf{K}\, ace(u), \mathbf{K}\, User(u) \rightarrow \mathbf{K}\, STContext(u, Universal)$
$\mathbf{K}\, ace(stc), \mathbf{K}\, STContext(stc) \rightarrow \mathbf{K}\, IsSubContextOf(stc, Universal)$
One of the following rules:
$\mathbf{K}\, ace(ro), \mathbf{K}\ \ Role(ro) \rightarrow \mathbf{K}\, adpa(ro, Universal, Open)$
$\mathbf{K}\, ace(ro), \mathbf{K} Role(ro) \rightarrow \mathbf{K}\, adpa(ro, Universal, Close)$
Inheritance over the role hierarchy:
$\mathbf{K}\, adpa(ro_1, av, v, stc, Open), \mathbf{K}\, IsSubRoleOf(ro_2, ro_1) \rightarrow \mathbf{K}\, adpa(ro_2, av, v, stc, Open)$
$\mathbf{K}\, adpa(ro_1, av, v, stc, Close), \mathbf{K}\, IsSubRoleOf(ro_1, ro_2) \rightarrow \mathbf{K}\, adpa(ro_2, av, v, stc, Close)$
Inheritance over the activity hierarchy:
$\mathbf{K}\, adpa(ro, av_1, v, stc, Open), \mathbf{K}\, IsSubactivityOf(av_2, av_1) \rightarrow \mathbf{K}\, adpa(ro, av_2, v, stc, Open)$
$\mathbf{K}\, adpa(ro, av_1, v, stc, Close), \mathbf{K}\, IsSubactivityOf(av_1, av_2) \rightarrow \mathbf{K}\, adpa(ro, av_2, v, stc, Close)$
Inheritance over the view hierarchy:
$\mathbf{K}\, adpa(ro, av, v_1, stc, Open), \mathbf{K}\, IsSubViewOf(v_2, v_1) \rightarrow \mathbf{K}\, adpa(ro, av, v_2, stc, Open)$
$\mathbf{K}\, adpa(ro, av, v_1, stc, Close), \mathbf{K}\, IsSubViewOf(v_1, v_2) \rightarrow \mathbf{K}\, adpa(ro, av, v_2, stc, Close)$
Inheritance over the context hierarchy:
$\mathbf{K}\, adpa(ro, av, v, stc_1, t), \mathbf{K}\, IsSubContextOf(stc_2, stc_1) \rightarrow \mathbf{K}\, adpa(ro, av, v, stc_2, t)$
Translation of abstract level predicates to concrete ones:
$\mathbf{K}\, adpa(ro, av, v, stc, Close), \mathbf{K}\, cra(ro, u, act), \mathbf{K}\, IsConsideredAs(ac, av), \mathbf{K}\, IsUsedIn(ob, v),$ $\mathbf{K}\, stcd(u, ac, ob, stc) \rightarrow \mathbf{K}\, cdpa(u, ac, ob, Close)$
$\mathbf{K}\, adpa(ro, av, v, stc, Open), \mathbf{K}\, cra(ro, u, act), \mathbf{K}\, IsConsideredAs(ac, av), \mathbf{K}\, IsUsedIn(ob, v),$ $\mathbf{K}\, stcd(u, ac, ob, stc), \mathbf{not}\ cdpa(u, ac, ob, Close) \rightarrow \mathbf{K}\, cdpa(u, ac, ob, Open)$

Table 17. Permission assignment meta policy

Translation of the concrete exception permission assignment:
$\mathbf{K}\, cepa(ro, u, t) \rightarrow \mathbf{K}\, cpa(ro, u, t)$
Translation of concrete regular permission assignment:
$\mathbf{K}\, crpa(ro, u, +), \mathbf{not}\ cepa(ro, u, -) \rightarrow\ \mathbf{K}\, cpa(ro, u, +)$
$\mathbf{K}\, crpa(ro, u, -), \mathbf{not}\ cepa(ro, u, +) \rightarrow\ \mathbf{K}\, cpa(ro, u, +)$
Translation of concrete default permission assignment:
$\mathbf{not}\ cpa(ro, u, +), \mathbf{not}\ cpa(ro, u, -), \mathbf{K}\, cdpa(ro, u, Open)$ $\rightarrow \mathbf{K} cpa(ro, u, +)$
$\mathbf{not}\ cpa(ro, u, +), \mathbf{not}\ cpa(ro, u, -), \mathbf{K}\, cdra(ro, u, Close)$ $\rightarrow \mathbf{K}\, cpa(ro, u, -)$

Session establishment process consists of the following steps:

(1) Session request reception: Session Manager (SM) receives a session request in the form $\langle u_r, Crd_r \rangle$ from a user u_r, where Crd_r is the set of credentials that u_r presents.

(2) Request validation: SM checks the validity of the received credentials using Credential Verifier and SOA, and sends the validated credentials to RAPDP.

(3) SKB initialization:

(a) RAPDP generates a set of assertions based on the received credentials and inserts them to the ABox of SO in security knowledge base K. The assertions are of the form $C(u_r)$, where $C \in Crd_r$. For example, if RAPDP receives a credential that says "Alice is a student" and concept *Student* is defined in TBox of SO of K, the assertion *Student(Alice)* is added to the ABox.

(b) By the request of RAPDP, K updates itself with the last change in long-term context information (using Context Management Point).

(4) Determination of activated roles: By the request of SM, RAPDP derives the activated roles for the user using the following inference:

$$activatedRole(u_r) = \{role | K \vdash_{MKNF^+} cra(role, u_r, Act)\} \quad (2)$$

RAPDP sends this set to SM.

(5) Session establishment : SM establishes a session for u_r and sends the session information including the activated roles to the user and PEP.

Controlling access requests during the session has the following steps:

(1) Request reception: PEP receives an access request in the form $\langle u_r, o_r, a_r, Crd_r \rangle$ from a user, where u_r, and o_r and a_r are a user (access requester), a resource (object), and an action, respectively. Crd_r is the set of credentials that u_r represents.

(2) Request validation:
  (a) PEP checks whether o_r is registered in the security domain and a_r is an eligible action on o_r (by checking the description of the resource in K).
  (b) PEP checks the validity of the received credentials using Credential Verifier and SOA, and sends the validated credentials to PAPDP.

(3) SKB updating:
  (a) PAPDP generates a set of assertions based on the received credentials and inserts them to the ABox of SO in K. The assertions are of the form $C(u_r)$, where $C \in Crd_r$. Furthermore for each activated role, like ro, PAPDP should add the predicate $cra(ro, u_r, Act)$ to the knowledge base.
  (b) By the request of PAPDP, K updates itself with the last changes in short-term context information (using Context Management Point).

(4) Access decision making: PAPDP sends "Permission" to PEP if $cpa(u_r, o_r, a_r)$ can be inferred from K i.e. $K \vdash_{MKNF^+} cpa(u_r, o_r, a_r)$. Otherwise PAPDP sends "Prohibition" to PEP.

After the session termination , PAPDP removes the assertions and predicates, which had been added to the SKB during the session.

One of the shortcomings of this procedure is that the determination of the activated roles has high computational overhead because computation of $activatedRole(u_r)$ is costly and also should be done for each session request. This operation can decrease the efficiency of the system significantly, especially when there exist numerous amounts of roles in the system and high volume of context information. One approach to decrease the negative effect of role activation phase is to ask users to determine their desirable roles. In this approach, users should send a subset of the existing roles with their session requests. Then, the system determines which of these roles should be activated. By doing so, not only smaller number of roles are checked for each session request, but also the *least privilege* principle is satisfied. An example of this approach is used in most of the Editorial Systems, where a user can determine his roles that he wants to have in the requested session at the login time.

6 Case Study

To demonstrate the applicability of the proposed model, a case study is discussed in this section. Figure 9 shows a partial definition of specific ontology for a healthcare domain. In addition to the general classes defined in the proposed upper-level ontology, a number of concrete sub-classes are defined to model more specific context information in a given environment (e.g. class *Channel* is defined).

Suppose that hospital H_1 uses the aforementioned ontology. Table 18 shows the role activation policy defined by H_1. According to the long-term context definition rules, context *IsPhysician* is true for user u if he has a physician certificate. Also context *Unsafe_Channel* is true for u if he uses an unsafe channel to request a session. Accordingly, H_1 activates role *Physician* and *Guest* in the contexts *IsPhysician* and *Universal* respectively. Also, role *Physician* is deactivated in context *Non_Physician*. However, H_1 may want to deactivate role *Physician* when the user uses an unsafe channel due to the potential leakage of sensitive information of patients. Therefore, H_1 defines this rule as an exception to the regular role activation rules. In fact, our proposed model can be used to implement *black list* approach easily. Finally, H_1 selects the *Close* default policy for *Universal* context.

Table 19 shows the permission assignment policy defined by H_1. Accordingly, H_1 grants permission to physicians to consult on a medical record when they are in the attending team of the patient who is the owner of the record. Also, it prohibits physicians whose context *Non_Attending_Physician* is true to access the medical records. However, in emergency conditions these physicians should be able to access the medical record of a patient. Therefore, H_1 defines an exception, which allows the physicians not belonging to the patient's attending team to access his medical

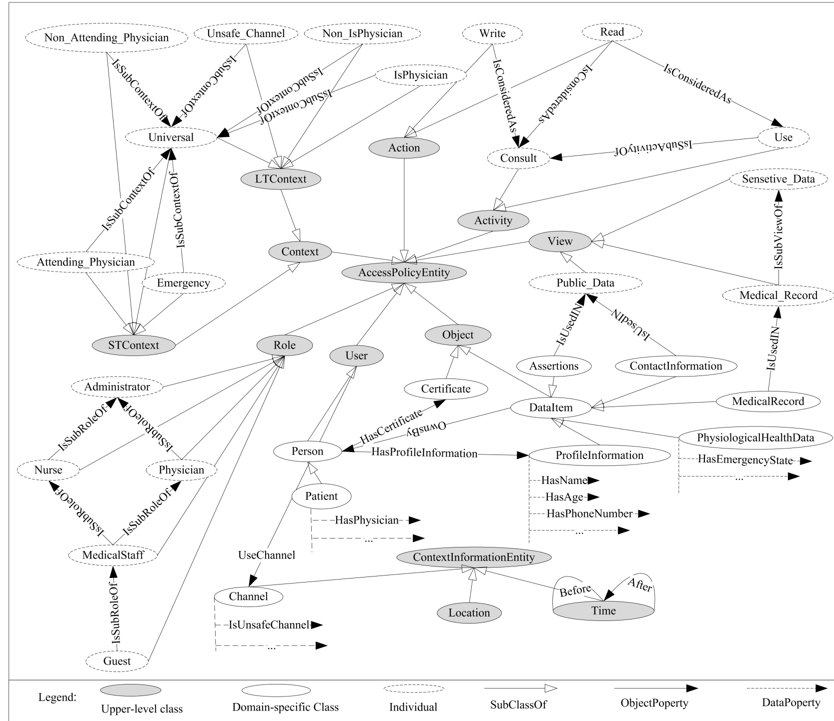

Figure 9. Partial definition of a specific ontology for healthcare domain

Table 18. An example of role activation policy

Long-term context definition:
$\mathbf{K}\, ape(u), \mathbf{K}\, HasCertificate(u, Physician_Cert) \rightarrow \mathbf{K}\, ltc(u, IsPhysician)$
$\mathbf{not}\; ltc(u, IsPhysician) \rightarrow \mathbf{K}\, ltc(u, Non_IsPhysician)$
$\mathbf{K}\, ape(u), \mathbf{K}\, cie(c), \mathbf{K}\, UseChannel(u, c), \mathbf{K}\, IsUnsafeChannel(c) \rightarrow \mathbf{K}\, ltc(u, Unsafe_Channel)$
Regular role activation policy:
$arra(Physician, IsPhysician, Act)$
$arra(Physician, Non_IsPhysician, Deact)$
$arra(Guset, Universal, Act)$
Exception role activation policy:
$aera(Physician, Unsafe_Channel, Deact)$
Default role activation policy:
$\mathbf{K}\, ace(ro), \mathbf{K}\, Role(ro) \rightarrow \mathbf{K}\, adra(ro, Universal, Close)$

record when he is in an emergency condition. Finally, H_1 chooses *Close* default policy for the permission assignment policy.

7 Related Work

Context-Aware Access Control (CAAC) is a family of access control solutions, which uses context information in access control process. Many CAAC-based models extend RBAC through adding new contextual constraints as shown in Figure 10. These constraints are used to control User Assignment (UA) and Permission Assignment (PA) functions dynamically. For this purpose, a role is assigned to a user if the user satisfies a set of contextual constraints. Similarly, a permission is assigned to a role if a set of contextual constraints are satisfied by the user who has the role. Many of context-aware access control models for PCEs are based on the CAAC model.

Two main RBAC-based access control models which have been proposed based on this approach are:

(1) Generalized Role-Based Access Control (GRBAC): Covington *et al.* [7] augmented the concept of environmental role to RBAC. This type

Table 19. An example of permission assignment policy

Short-term context definition:
$\mathbf{K}ape(u), \mathbf{K}ape(p), \mathbf{K}ape(o), \mathbf{K}HasPhysician(p,u), \mathbf{K}OwenBy(o,p) \rightarrow \mathbf{K}stc(u, ac, o, Attending_Physician)$
$\mathbf{not}\ stc(u, ac, o, Attending_Physician) \rightarrow \mathbf{K}\ stc(u, ac, o, Non_Attending_Physician)$
$\mathbf{K}ape(p), \mathbf{K}ape(hd), \mathbf{K}ape(mr), \mathbf{K}OwnBy(mr,p), \mathbf{K}OwnBy(hd,p), \mathbf{K}HasEmergencyState(hd, True)$
$\rightarrow stc(u, ac, mr, Emergency)$
Regular permission assignment policy:
$arpa(Physician, Consult, MedicalRecord, Attending_Physician, +)$
$arpa(Physician, Use, MedicalRecord, Non_Attending_Physician, -)$
Exception permission assignment policy:
$aepa(Physician, Consult, MedicalRecord, Emergency, +)$
Default permission assignment policy:
$\mathbf{K}\ ace(ro), \mathbf{K}Role(ro) \rightarrow \mathbf{K}\ adpa(ro, Universal, Close)$

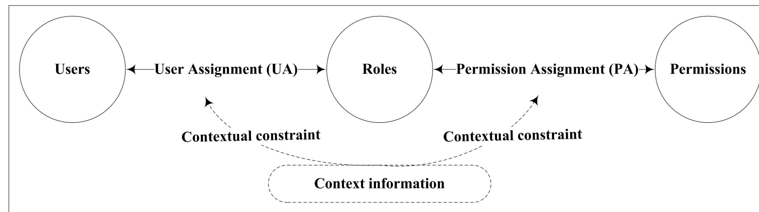

Figure 10. Conceptual view of CAAC

of roles allows the participation of context information in the access control procedure. Using the role concept for modeling of user and environment characteristics redounds to the simplicity and flexibility of policy definition language.

(2) Dynamic Role-Based Access Control Model (DRBAC): DRBAC [21] is one of the complete context-aware extensions of RBAC in which dynamic assignments of roles and permissions are provided. In DRBAC access decisions are made based on the current context/state of the system and the certificates of the user. DRBAC tries to address the two main requirements including dynamic change of assigned roles to a user and the dynamic change of granted permissions to the roles when system information is changed. DRBAC reaches this goal using a state machine for the both assigned roles and granted permissions, and considering state transition by change in the system information.

One of the main shortcomings of the aforementioned access control models is the lack of inference capability. In other words, they are not semantic-aware access control models. According to the advantages of using logics in access control, a wide spread researches have been done for using logics in access control models so far [4]. Organization-based Access Control (OrBAC) [12] is one the famous logic-based access control model which extends the RBAC. OrBAC generalizes the role concept for objects and actions by adding view and activity concepts to RBAC. Different types of context information have been modeled by Cuppens *et al.* [8] in OrBAC using first-order logic.

Logics used in access control models typically are monotonic logics in which previous conclusions are not defeated by adding new information. However, according to the demonstrated requirements, non-monotonicity is a critical and integral part of access control in novel computing environments and these requirements cannot be provided using classic (monotonic) logics.

Based on the needs for provision of non-monotonic access control requirements, some researchers tried to do this using non-monotonic logics. Boustia *et al.* in their set of researches [6] [5] extended description logic with two operators namely default (δ) and exception (ε) to provide the default and exception definition capabilities. This logic named JClassic$_{\delta\varepsilon}$ and makes the access control model to be able to define default and exception knowledge in the conceptual level. However, this logic inherits the shortcomings of description logic such as lack of ability to non-tree like relationship description, lack of support of rules, and lack of support of integrity constraint definition capability.

Several frameworks have been proposed for combining DLs and rules (logic programming). Description logic ALC and positive Datalog programs are com-

bined in AL-log [9]. Rosati *et al.* extended AL-log and proposed DL-log [18]. In comparison with AL-log, disjunctive Datalog with negation and pair predicates are supported in DL-log. To preserve decidability, the weak DL-safety condition has been employed. According to this restriction, each variable in a DL-atom of the conclusion must occur in a non-DL-atom of the premises. $MKNF^+$ logic as a combination of ASP and DL is proposed by Motick *et al.* [14]. They showed that each DL-log knowledge base can be encoded to a $MKNF^+$ knowledge base. However, $MKNF^+$ is more flexible than DL-log. In DL-log, DL-predicates and non-DL-predicates are interpreted under open-world and closed-word assumption respectively, which makes DL-log knowledge base inflexible. In contrast to DL-log, in $MKNF^+$, an open-world or closed-world interpretation of a predicate can be chosen freely through its usage in either a non-modal or a modal atom. To the best of our knowledge, $MKNF^+$ is one of the most powerful decidable formalisms proposed for combination of DL and rules, and thus, it is used in this paper.

8 Conclusion

Non-monotonicity is an important feature in context-aware access control models. Furthermore, semantic technology and semantic modeling languages like OWL are appropriate and widely used mechanisms for context modeling. In this paper, the advantages of semantic technology and answer set programming have been integrated to propose a powerful context-aware access control model where $MKNF^+$ is employed to specify the required policies which satisfying non-monotonic requirements. In the proposed model, DL is used to model main entities of access control as well as the context information. In addition, $MKNF^+$ rules, which are used to express role activation and permission assignment policy rules and specify context information, enable the model to support non-monotonic reasoning in presence of incomplete context information. The default and exception role activation and permission assignment rules are defined using negation-as-failure, which is one of the main non-monotonic features of the $MKNF^+$ logic.

References

[1] M. Amini and R. Jalili, "Multi-level Authorisation Model and Framework for Distributed Semantic-aware Environments," *IET Information Security*, vol. 4, no. 4, pp. 301–321, 2010.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge university press, 2003.

[3] Y. Bai, "A Modal Logic for Authorization Specification and Reasoning," in *IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 1. IEEE, 2009, pp. 264–268.

[4] P. A. Bonatti and P. Samarati, "Logics for Authorization and Security," *Logics for Emerging Applications of Databases*, pp. 277–323, 2003.

[5] N. Boustia and A. Mokhtari, "Representation and Reasoning on ORBAC: Description Logic with Defaults and Exceptions Approach," in *Proceedings of the Third International Conference on Availability, Reliability and Security.* IEEE, 2008, pp. 1008–1012.

[6] ——, "A Contextual Multilevel Access Control Model with Default and Eception Description Logic," in *Proceedings of the International Conference for Internet Technology and Secured Transactions.* IEEE, 2010, pp. 1–6.

[7] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, "Securing Context-Aware Applications using Environment Roles," in *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies.* ACM, 2001, pp. 10–20.

[8] F. Cuppens and N. Cuppens-Boulahia, "Modeling Contextual Security Policies," *International Journal of Information Security*, vol. 7, no. 4, pp. 285–305, 2008.

[9] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf, "Al-log: Integrating Datalog and Description Logics," *Journal of Intelligent Information Systems*, vol. 10, no. 3, pp. 227–252, 1998.

[10] S. S. Emami and S. Zokaei, "A Context-Sensitive Dynamic Role-Based Access Control Model for Pervasive Computing Environments," *ISeCure, The ISC International Journal of Information Security*, vol. 2, no. 1, pp. 47–66, 2010.

[11] M. Gelfond and V. Lifschitz, "Classical Negation in Logic Programs and Disjunctive Databases," *New Generation Computing*, vol. 9, pp. 365–385, 1991.

[12] A. A. E. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin, "Organization Based Access Control," in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks.* IEEE, 2003, pp. 120–131.

[13] V. Lifschitz, "Nonmonotonic Databases and Epistemic Queries," in *Proceedings of the 12th International Conference on Artificial Intelligence*, vol. 1, 1991, pp. 381–386.

[14] B. Motik and R. Rosati, "Reconciling Description Logics and Rules," *Journal of the ACM*, vol. 57, no. 5, pp. 30:1–30:62, June 2008.

[15] F. Rabitti, E. Bertino, W. Kim, and D. Woelk,

"A Model of Authorization for Next-generation Database Systems," *ACM Transactions on Database Systems (TODS)*, vol. 16, no. 1, pp. 88–131, 1991.

[16] A. N. Ravari and M. S. Fallah, "A Logical View of Nonmonotonicity in Access Control," in *SECRYPT*, 2011, pp. 472–481.

[17] R. Reiter, "Readings in Nonmonotonic Reasoning," M. L. Ginsberg, Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, ch. A Logic for Default Reasoning, pp. 68–93.

[18] R. Rosati, "Towards Expressive KR Systems Integrating Datalog and Description Logics," in *Proceedings of the 1999 International Workshop on Description Logics DL*. Citeseer, 1999, pp. 160–164.

[19] X. Wang, D. Q. Zhang, T. Gu, and H. Pung, "Ontology Based Context Modeling and Reasoning using OWL," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. IEEE, 2004, pp. 18–22.

[20] T. Y. Woo and S. S. Lam, "Authorization in Distributed Systems: a Formal Approach," in *IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 1992, pp. 33–50.

[21] G. Zhang and M. Parashar, "Context-Aware Dynamic Access Control for Pervasive Applications," in *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2004, pp. 21–30.

Appendix A. Introduction to *MKNF*$^+$

MKNF$^+$ integrates DL and ASP using the MKNF logic as a semantic infrastructure. To introduce *MKNF*$^+$, a brief overview of its underlying components is necessary.

- Description Logic (DL): DL represents the knowledge of a world by defining the existing concepts in the world (its terminology) and their relationships (roles), and then using the defined concepts and roles to specify individuals existing in the world and their properties. For example, concepts such as *Person* and *Application* can be defined in a PCE as two general category of subjects who use resources. The syntax of DL consists of atomic concepts, atomic roles, and individuals which are corresponding to unary predicates, pair predicates, and constants in first-order logic respectively. A DL knowledge base O consists of two components [2]:
  (1) Terminology Box (TBox): TBox describes the general structure of the world using concepts (classes) and roles (relations).
  (2) Assertion Box (ABox): ABox specifies the objects in the world and their properties. It includes assertions like $C(a)$ (e.g. $Person(Alice)$) and $R(a, b)$ (e.g. *IsLocatedIn(Alice, CS-Department)*) in which C and R are a concept and a relation respectively.
- ASP: ASP is a non-monotonic rule based formalism that can compensate the shortcomings of DL. A literal in ASP is a formula of the form A or $\neg A$, where A is a function-free first-order atom. ASP supports two types of negations:
  (1) Classical or strong negation ($\neg$): The negation is used for specifying explicit negative information. In other words, the ASP program P concludes $\neg A$, if $\neg A$ is explicitly inferred from it.
  (2) Non-monotonic negation-as-failure (**not**): **not** A means that A can be false. In other words, A is false or it is not possible to determine its truth value.

  An answer set program P is a finite set of rules of the form:

$$B_1, ..., B_m, \mathbf{not}\, B_{m+1}, ..., \mathbf{not}\, B_n \rightarrow H_1 \vee ... \vee H_k,$$

  where B_i and H_j are literals. Operator **not**, can be used to specify the default truth value for a predicate. As an example the closed-world assumption for a predicate A can be expressed using the following rule:

$$\mathbf{not}\, A(x) \rightarrow \neg A(x).$$

- MKNF: The logic of Minimal Knowledge and Negation as Failure (MKNF) was proposed by Lifschitz [13] to unify different non-monotonic formalisms, such as default logic, auto-epistemic logic, and logic programming. MKNF extends the first-order logic by **K** and **not** operators.

  Suppose that, $\Sigma = (\Sigma_c, \Sigma_f, \Sigma_p)$ be a first-order signature, where Σ_c, Σ_f, and Σ_p are a set of constants, a set of function symbols, and a set of predicates containing the pair equality predicate $\approx$, respectively. The following grammar in which t_i's are first-order terms and P is a predicate, defines the syntax of MKNF formulas over Σ:

$$\varphi ::= P(t_1, ..., t_n)|(\neg\varphi)|(\varphi \wedge \varphi)|(\exists x : \varphi)|(\mathbf{K}\,\varphi)|(\mathbf{not}\,\varphi)$$

  A formula of the form $\mathbf{K}\,\varphi$ and $\mathbf{not}\,\varphi$ are modal atoms, which named as *modal **K**-atom* and ***not**-atom* respectively. An MKNF formula φ is close if it has no free variable. Accordingly, an MKNF theory is a countable set of closed MKNF formula.

  Assume that Σ is a signature and $\triangle$ is a non-empty set called universe. A first-order interpretation over Σ and $\triangle$ assigns what is shown in Table 20 for each element of Σ. In addition, predicate $\approx$ is interpreted as equality. Before a brief review of the semantics of MKNF, the following three key points should be considered:

Table 20. The first-order interpretation over Σ and $\triangle$ in semantics of MKNF

Element of Σ	Output of the first-order interpretation
constant $a \in \Sigma_c$	An object $a^I \in \triangle$
n-ary function symbol $f \in \Sigma_f$	A function $f^I : \triangle^n \rightarrow \triangle$
n-ary predicate $P \in \Sigma_p$	A relation $P^I \subseteq \triangle^n$

Table 21. Satisfaction of a closed MKNF formula in an MKNF structure [14].

(I,M,N) $\models$ true	for each triple (I, M, N)
(I,M,N) $\models P(t_1, ..., t_n)$	iff $(t_1^I, ..., t_n^I) \in P^I$
(I,M,N) $\models \neg\varphi$	iff *(I, M, N)* $\not\models \varphi$
(I,M,N) $\models \varphi_1 \wedge \varphi_2$	iff *(I, M, N)* $\models \varphi_1$ and *(I, M, N)* $\models \varphi_2$
(I,M,N) $\models \exists x : \varphi$	iff *(I, M, N)* $\models \varphi[n_\alpha/x]$ for some $\alpha \in \triangle$
(I,M,N) $\models$ **K** φ	iff *(I, M, N)* $\models \varphi$ for all $J \in M$
(I,M,N) $\models$ **not** φ	iff *(I, M, N)* $\not\models \varphi$ for some $J \in N$

(1) Existing of a special constant n_α called name: In contrast to standard first-order logic, for each element $\alpha \in \triangle$, the signature Σ must contain this constant such that $n_\alpha^I = \alpha$.
(2) The interpretation of a variable-free term $t = f(s_1, ..., s_n)$: This interpretation is defined recursively as $t^I = f^I(s_1^I, ..., s_n^I)$.
(3) MKNF triple: The MKNF triple over a universe $\triangle$ is a triple *(I, M, N)* such that I is a first-order interpretation over $\triangle$ and Σ; and M and N are non-empty sets of first-order interpretation over $\triangle$ and Σ.

An MKNF triple is used to define the semantics of an MKNF formula over signature Σ. Table 21 shows the definition of satisfiability of a closed MKNF formula in *(I, M, N)*. Accordingly, **K**φ means that φ is known to hold or be true. Also **not** is a weak negation or negation-as-failure.

One of the important characteristics of *MKNF* is that each DL and ASP knowledge base can be translated into the *MKNF* logic. This property is used in the proposing of $MKNF^+$. Each $MKNF^+$ knowledge base is a pair $K = (O, P)$, where O is a DL knowledge base and P is a program (finite set of $MKNF^+$ rules). Predicates defined in O are called DL-predicates and other predicates are called non-DL-predicates. DL-predicates are unary or pair predicates but non-DL-predicates are not bounded. Moreover, two types of modal atoms namely **K**-atom and **not**-atom are defined in this formalism. **K**-atom is denoted by **K** A and **not**-atom is denoted by **not**A. The semantics of $MKNF^+$ is based on the MKNF semantics. In fact, parts O and P of $MKNF^+$ knowledge base are translated into MKNF separately (see [14] for further explanation of the $MKFN^+$ semantics). The structure of an $MKNF^+$ rule is as follows:

$$B_1, ..., B_n \rightarrow H_1 \vee ... \vee H_m$$

where, B_i can be a non-modal predicate, a **K**-atom, or a **not**-atom, whereas, H_i would be either a non-modal predicate or a **K**-atom. To preserve decidability of $MKNF^+$, the DL-safety restriction must be applied; each variable in a rule should appear in the body of the rule in some non-DL-K-atom. The main idea of this restriction is to restrict the applicability of the rules only to the individuals that are explicitly specified by name in the knowledge base.

Seyyed Ahmad Javadi received his B.S. degree in Software Engineering from Ferdowsi University of Mashhad, Iran, in 2010, and his M.S. degree in the same field from Sharif University of Technology, Tehran, Iran, in 2012. His research interests are Access Control in pervasive computing environments, Database Security including Security in Database Outsourcing Scenarios, and Formal Methods in Information security.

Morteza Amini received his Ph.D. in Software Engineering (Data Security field) from Sharif University of Technology. He is currently an assistant professor at Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. His research interests include Database Security, Access Control, Intrusion Detection, and Formal Methods in information security.