Non-monotonocity in OrBAC through Default and Exception Policy Rules

Seyyed Ahmad Javadi, Morteza Amini, Rasool Jalili
Data and Network Security Lab (DNSL)
Department of Computer Engineering
Sharif University of Technology
Tehran, IRAN
{ajavadi@ce., amini@, jalili@}sharif.ir

***Abstract*—Context-awareness is an essential requirement of modern access control models. Organization-Based Access Control (OrBAC) model is a powerful context-aware access control model defined by first-order logic. However, due to the monotonicity nature of the first-order logic, OrBAC suffers from the incapability of making decision based on incomplete context information as well as the definition of default and exception policy rules. This paper proposes augmenting non-monotonicity features to OrBAC using MKNF$^+$ logic, which is a combination of Description Logic (DL) and Answer Set Programming (ASP). Along with the use of DL to define ontology for main entities and context information in OrBAC; MKNF$^+$ rules are used to define access control, default, and exception policy rules. The proposed model inherits the advantages of ontological representation of OrBAC entities and context information (such as interoperability among systems) as well as the ASP advantages in non-monotonic reasoning through *closed-world principle* and *negation as failure*. The expressive power of the model is also demonstrated through a case study.**

Keywords*: Role-Based Access Control, Non-monotonic Logic, Default Policy Rule, Exception Policy Rule*

I. INTRODUCTION

Access control policy determines what, where, when, and how subjects can access databases, web services, electronic devices, and any other resources. Context-awareness is an essential requirement of recent access control models. Organization-Based Access Control (OrBAC) [1] is one of such models, which attempts to overcome the limitations of previous access control models through the consideration of organization and context concepts.

The use of logics in access control models has advantages including clean foundations, flexibility, expressiveness, declarativeness, and inference capability [2]. Two overall classes of logics are:

- Monotonic logics, where inference of a sentence *A* from a set *T* of sentences, implies its inference from any arbitrary superset of *T*. In other words, current conclusions are not invalidated by adding new information and premises. Classical logics such as propositional and first-order logic are monotonic.
- Non-monotonic logics, where some of the current conclusions may be retracted by adding new information and premises.

Based on the definition, in a non-monotonic access control system, adding new information or access control rules may invalidate some of the previous conclusions (permissions/ prohibitions). The following four requirements are introduced as the motivation of introducing non-monotonic access control systems [2], [3]:

1. Context information may be imperfect as it is impossible to gather all context information completely and accurately all the time. For example, a user location might be unknown due to the communication failure, the sensor failure, or any other types of failures. Incomplete context information cannot be modeled by monotonic logics such as propositional and first order logic. Answer Set Programming (ASP) [4] is an appropriate decidable logic for handling incomplete context information. ASP supports negation as failure (in addition to classical negation), using which, the default context information can be defined.
2. In some conditions, permissions (prohibitions) should be granted (revoked) exceptionally. In addition, ability to define exceptions to access control rules increases the expressiveness of the policy specification language. If exception is supported by policy specification language, *general* access control rules can be defined initially and the *specific* authorizations can be defined subsequently using exceptions [2].
3. When an access control model supports defining both permission and prohibition on the same target, some strategies are required to resolve the possible conflicts in access decision process.
4. A default policy is required when neither permission nor prohibition about a request is inferred.

Several approaches have been proposed for modeling context information so far [5]. Among these approaches, using ontology has some more advantages such as interoperability among systems (easier knowledge sharing), and deducing the high level conceptual context from the low level context [6]. Additionally, context reasoning based on ontology is supported by optimized automatic tools [5]. However, DL, which is normally used to specify ontologies, has some shortcomings as represented in TABLE I. Accordingly, while DL is appropriate for context modeling, it is not strong enough for access control policy specification. The shortcomings of

978-1-4673-2386-4/12/$31.00 ©2012 IEEE

DL can be compensated by ASP [7]. A hybrid logic, combining DL and ASP together, would be a good solution to use the features found in the both.

In this paper, MKNF$^+$ [7], as a combination of ASP and DL, is used to handle non-monotonicity in OrBAC. Main entities of OrBAC and context information are modeled as an ontology using DL, part of MKNF$^+$, while various contextual conditions (e.g. default context) and access control rules are specified using rule specification capability of MKNF$^+$. In addition, exception and default policy rules are added to OrBAC using negation as failure.

In summary, our proposed model uses the DL's strengths in *semantic technology* alongside the ASP's strengths in *non-monotonic reasoning*. In addition, the principle which is considered in this paper is that *exception* policy rules can be considered as *exception* to *regular* access control rules and regular access control rules can be considered as *exception* to *default* policy rules. In both cases, the exceptions can be defined in MKNF$^+$ using negation as failure. Thus, all the non-monotonic features that we like to have in OrBAC can be augmented to it using negation as failure feature of MKNF$^+$.

The rest of this paper is organized as follows. Section II briefly overviews some preliminaries including imperfect context information, OrBAC, and MKNF$^+$. Section III surveys related work. The ontologies of main entities of OrBAC and context information are represented in Section IV. In Section V, our proposed approach to meet the non-monotonic access control requirements are described. A case study for clarifying the applicability of the approach is mentioned in Section VI. Finally, Section VII concludes the paper and draws some future directions.

II. PRELIMINARIES

Various types of imperfect context information, OrBAC, and MKNF$^+$ are discussed in the following as preliminaries.

A. Imperfect Context information

The following four categories of imperfect context information have been characterized by Henricksen *et al.*[8]:

1. Unknown: when no information about a contextual property is available or cannot be incorporated into the access control system. This type of imperfection is also known as *incomplete* information [3].
2. Ambiguous: when several different values are reported about the property; e.g. two or more distinct locations are reported by separate positioning devices for a given person.
3. Imprecise: when the reported values of the property are approximately correct; e.g. the imprecise geographical position of a person.
4. Erroneous: when reported values of the property do not match the actual values. Erroneous context information can be arisen as a result of human error.

In this paper, we consider incomplete information and use negation as failure to deal with such kind of imperfect context (see Section IV.B). Handling the other three categories is leaved for our future research.

TABLE I. The shortcomings and strengths of DL and ASP [7]

Logic Name	Shortcomings	Strengths
DL	• No support for logical rules • Unfeasibility to express non-tree-like relationships • Impossibility to express integrity constraints • Lack of support for non-monotonic reasoning (i.e. closed-world reasoning)	• Support for reasoning with unbounded or infinite domain • Powerful description of structured knowledge
ASP	• Lack of support for reasoning with unbounded or infinite domain	• Support for negation as failure and closed-world reasoning

B. Introduction to OrBAC

OrBAC is a context-aware access control model with the aim of overcoming the previous access control models' limitations. Main entities of OrBAC and their definitions are represented in TABLE II. There are eight basic sets of entities in OrBAC: ORG (set of organizations), SU (set of subjects), RO (set of roles), OB (set of objects), V (set of views), AC (set of actions), AV (set of activities), and C (set of context). The main predicates and their descriptions in OrBAC are represented in TABLE III. Predicates *Prohibition* and *Is-prohibited* are defined similar to *Permission* and *Is-permitted* respectively. Axiom (1) describes how abstract permissions among roles, views, and activities can be transformed into concrete permissions among subjects, objects, and actions. The axiom for prohibition is defined similarly.

$$\begin{aligned}&\forall su \in SU\, \forall ob \in OB\, \forall ac \in AC\, \forall ro \in RO\, \forall av \in AV\, \forall v \in V\, \forall c \in C \\ &Permission(Org, ro, av, v, c) \wedge Employ(Org, su, ro) \wedge \\ &Use(Org, ob, v) \wedge Consider(Org, ac, av) \wedge \\ &Define(Org, su, ac, ob, c) \rightarrow Is\text{-}permitted(su, ac, ob)\end{aligned} \quad (1)$$

Different types of context information have been modeled by Cuppens *et al.* [9] in OrBAC using first-order logic. As an example, suppose the following rule used by hospital H_1 to define the context *location(so)*:

$$\forall su \in SU, \forall ac \in AC, \forall ob \in OB, (Is\_located(H_1, su, so) \rightarrow Define(H_1, su, ac, ob, location(so)).$$

Where, the function *location* maps each spatial object in set SO to a physical spatial context (e.g. Office_32). This rule means that the context *location(so)* is *true* among subject *su*, action *ac*, and object *ob*; if *su* is located in the area of spatial object *so*.

TABLE II. Main entities of OrBAC [1]

Entity	Definition
Organization	An organized group of active entities (i.e. subjects) playing some role for specific goals.
Subject	An active entity (i.e. user) or an organization.
Role	The entity used to structure the link among subjects and organizations.
Object	An inactive (passive) entity such as a data file, an email.
View	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 partake of the same principles.
Context	Used to specify the concrete circumstances where organizations grant permissions to roles for performing activities on views.

TABLE III. Main predicates and their description in OrBAC [1]

Predicate Name	Domain (Dom.), Description(Des.) and Example (e.g.)
Employ	Dom: ORG×SU×RO Des: *Employ(org, su, ro)* means that *org* employs subject *su* in the role *ro*. e.g.: Employ(H_1, Bob, Physician)
Use	Dom: ORG×OB×V Des: *Use(org, ob, v)* means that *org* uses object *ob* in the view *v*. e.g.: Use(H_1, F1.doc, Medical_Record)
Consider	Dom: ORG×AC×AV Des: *Consider(org, ac, av*) means that *org* considers that action *ac* falls within the activity *av*. e.g.: Consider(H_1, Read, Consult)
Define	Dom: ORG×SU×AC×OB×C Des: *Define(org, su, ac, ob, c)* means that within *org*, context *c* is *true* among subject *su*, object *ob* and action *ac*. The required conditions for a specific context are described by logical rules. e.g.: $\forall su\in SU, \forall ac\in AC, \forall ob\in ob(Name\_Patient(ob,p)\wedge Patient(su,p))\rightarrow Define(H_1,su,ac,ob,Attending\_Physician)$
Permission	Dom: ORG×RO×AV×V×C Des: *Permission(org, ro, av, v, c)* means that *org* grants role *ro* permission to perform activity *av* on view *v* within context *c*. e.g.: Permission(H_1, Physician, Consult, Medical_Record, Attending_Physician)
Is-permitted	Dom: SU×AC×OB Des: *Is-permitted(su, ac, ob)* means that subject *su*, is concretely permitted to perform action *ac* on object *ob*. e.g.: Is-permitted(Bob, Read, F1.doc)
Sub-role	Dom: ORG×RO×RO Des: *Sub-role(org, ro_1, ro_2)* means that in *org*, role ro_1 is sub role of role ro_2. e.g.: Sub-role (H_1, Administrator, Physician)

C. Introduction to MKNF$^+$

MKNF$^+$ [7] is a formalism for the combination of DL and ASP. Each MKNF$^+$ knowledge base is a pair *K= (O, P)*, where *O* is a DL knowledge base and *P* is a program. Predicates defined in *O* are called DL-predicates and other predicates are called non-DL-predicates. DL-predicates are unary or binary 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* (read "*A* is known to hold") and not-atom is denoted by *not A* (read "*A* can be false") [7]. 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 rules only to individuals that are explicitly mentioned by name in the knowledge base. In the rest of this paper, names of DL-atoms are indicated by initial cap words and names of non-DL-atoms are demonstrated by lowercase words. In addition, names of variables begin with lowercase letters while names of constants begin with uppercase letters.

III. Related Work

For the first time, *default logic* as a powerful (but not decidable) non-monotonic logic has been used in access control policy specification by Woo and Lam [10]. To address the complexity issues of default logic, using the fragment of default logic corresponding to stratified, extended logic program was proposed by them. Although the proposed access control model is a powerful model, context information modeling is not considered in it. Ordered logic programs which supports negation by failure has been proposed as policy specification language by Bertino *et al*. [11]. This model supports exception but it does not pay attention to the context information.

L^k as a knowledge base formal language has been proposed to specify authorization domains with incomplete information by Bai [12]. Three types of propositions namely *initial, objective,* and *subjective* are defined for this purpose. The semantics of L^k is defined based on the *world view semantic* of epistemic logic programs. TABLE IV shows the propositions and their translation to epistemic logic program, where ϕ is a conjunctive or disjunctive fact expression, and ψ , β , and γ are conjunctive fact expressions. Note that $\neg \mathbf{K}\beta$ can be used in this logic; but, it is not supported in MKNF$^+$.

OrBAC is defined by first-order logic, which is monotonic and dose not satisfy non-monotonic requirements. Boustia *et al.* [13] proposed a new extension that handles non-monotonicity in OrBAC using their proposed logic named JClassic$^-_{\delta\varepsilon}$. JClassic$^-_{\delta\varepsilon}$ is a DL based system augmented with two operators δ (for default) and ε (for exception). JClassic$^-_{\delta\varepsilon}$ inherits some shortcomings from DL including lack of support for logical rules, arity restriction, inability to axiomatize integrity constraints, and lack of support for *closed-world reasoning* and *negation as failure*.

Several frameworks have been proposed for combining DLs and rules. Description logic ALC and positive Datalog programs are combined in AL-log [14]. Rosati*et al.* extended AL-log and proposed DL-log [15]. In comparison with AL-log, disjunctive Datalog with negation and binary 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 head must occur in a non-DL-atom of the body. Motik *et al.* [7] showed that each DL-log knowledge base can be encoded to a MKNF$^+$ knowledge base. However, MKNF$^+$ is more flexible than DL-log.

TABLE IV. Three propositions in L^k and their translation to epistemic logic program [12]

Proposition Name (PN), Proposition Form (PF)	Translation
PN: initial PF: **initially** ϕ	$\rightarrow\phi$
PN: objective PF: ϕ **if** ψ **with absence** γ	$\psi, \mathbf{not}\ \gamma\rightarrow\phi$
PN: subjective PF: ϕ **if** ψ **with absence** γ **knowing** β or ϕ **if** ψ **with absence** γ **not knowing** β	$\psi, \mathbf{not}\ \gamma, \mathbf{K}\beta\rightarrow\phi$ or $\psi, \mathbf{not}\ \gamma, \neg\mathbf{K}\beta\rightarrow\phi$

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 [7].

In this paper, we propose an extension to OrBAC in which MKNF$^+$ used to representing context information and defining access policy rules. In MKNF$^+$, predicates can be defined with arbitrary arities. Furthermore, MKNF$^+$ supports non-monotonic inference rule such as *negation as failure* which can be used to handle incomplete context information and default policy rules specification. To the best of our knowledge, MKNF$^+$ is the most powerful decidable formalism proposed for combination of DL and rules, and thus, it is used in this paper.

IV. ORBAC AND CONTEXT INFORMATION ONTOLOGY

Since, ontology is chosen as data model in our proposed access control model, the details of modeling of main entities of OrBAC as well as context information (especially the incomplete one) are discussed in the following.

A. Ontological Representation of Main Entities of OrBAC and Context Information

Context information may be provided by different service providers with different data models. However, a common understanding between client and service provider is needed for interoperability among systems. Semantic technology and especially OWL language are widely used to data modeling independent of the underlying models. As Figure 1 shows, in our proposed model main entities of OrBAC 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. Context model is divided into upper level ontology and specific ontology in CONON ontology [16] for resolving the issue. We use the same idea, so our proposed ontology consists of two layers:

1. **Upper Level Ontology** is a high-level ontology which represents the main general entities and their attributes. Such entities can be used in different domains.
2. **Domain Specific Ontology** is a detailed ontology which describes domain specific concepts and their relationship, in addition to the main concepts.

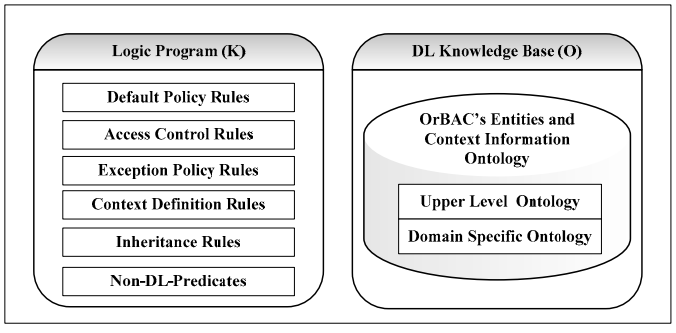

Figure 1. Proposed model security knowledge base structure

Figure 2 depicts the upper level ontology's entities consisting of main concepts in OrBAC, i.e., organization, context, role, activity, view, subject, action, and object and their properties. Additionally, the two important context information including *time* and *location* are considered in the upper level ontology.

In the rest of this paper, non-DL-predicates cie(ContextInformationEntity) and ape(AccessPolicyEntity) are used to apply DL-safety restriction. All the access policy entities and context information entities (sub-classes or individuals) are defined as members of predicates "ape" and "cie" respectively. For example, if hospital H1 defines a new role called Surgeon, Role(Surgeon) and ape(Surgeon) are added to the security knowledge base. As another example, if H_1 defines a new location called Surgery_Room2, Location(Surgery_Room2) and cie(Surgery_Room2) are added to the security knowledge base. By doing so, the DL-safety restriction is satisfied in the MKNF$^+$ rules, which are used in our proposed access control model.

B. Dealing with Incomplete Context Information

Some aspects of environment may be unknown to the access control system. MKNF$^+$ rules support negation as failure, to cope with deduction in presence of incomplete information. Generally, the following two states are possible for each predicate [4]:

1. Information about the predicate is complete and the *closed-world assumption* is applicable. The following rule can be used to define *closed-world assumption* for the predicate *p(x)*:

   $$\mathbf{not}\ \mathrm{p}(\mathrm{x}) \rightarrow \neg \mathrm{p}(\mathrm{x}).$$

   For example, hospital H_1 can assume that information about the attending physician is complete, so it can use Rule (2) to apply the *closed-world assumption* to the predicate *define* for context Attending_Physician.
2. Information about the predicate is incomplete. The rule "the predicate can be assumed *true* if some prerequisite conditions be *true* and there exists no clear negative information about the predicate" can be used as a general rule to deal with incompleteness. Rule (3) expresses this general rule for predicate *p(x)*. As an example, IsLocatedIn(Subject, Location) is a useful predicate which may be unknown for some subjects. For default context information, H_1 can define the following rule: "at the working hours, a hospital nurse is in hospital if he is not on vacation and there is no contrary information". Rule (4) defines the predicate workingHours. Rule (5) is the default context rule for the previous general rule.

$\mathbf{not}\ \mathrm{define}(H_1, \mathrm{su}, \mathrm{ac}, \mathrm{ob}, \mathrm{Attending\_Physicion}) \rightarrow \neg \mathrm{define}(H_1, \mathrm{su}, \mathrm{ac}, \mathrm{ob}, \mathrm{Attending\_Physicion})$	(2)
$\mathrm{Prerequisite\_Conditions}, \mathbf{not}\ \neg \mathrm{p}(\mathrm{x}) \rightarrow \mathrm{p}(\mathrm{x})$	(3)
$\mathbf{K}\ \mathrm{After}(08:00), \mathbf{K}\ \mathrm{Before}(19:00), \mathbf{K}\ \neg \mathrm{OnDay}(\mathrm{Saturday}), \mathbf{K}\ \neg \mathrm{OnDay}(\mathrm{Sunday}) \rightarrow \mathbf{K}\ \mathrm{workingHours}()$	(4)
$\mathbf{K}\ \mathrm{ape}(\mathrm{su}), \mathbf{K}\ \mathrm{IsEmployedIn}(\mathrm{su}, \mathrm{Nurse}), \mathbf{K}\ \mathrm{workingHours}(), \mathbf{not}\ \mathrm{IsInVacation}(\mathrm{su}, \mathrm{True}), \mathbf{not}\ \neg \mathrm{IsLocatedIn}(\mathrm{su}, H_1) \rightarrow \mathbf{K}\ \mathrm{IsLocatedIn}(\mathrm{su}, H_1)$	(5)

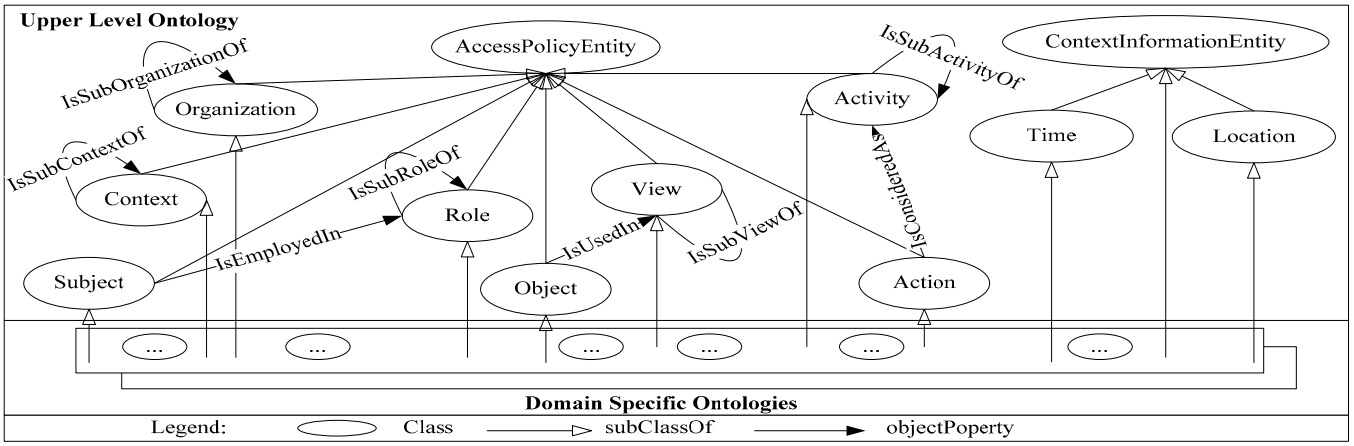

Figure 2. Partial definition of upper level ontology

V. NON-MONOTONICITY IN ORBAC

In order to augment non-monotonicity features to OrBAC, using MKNF$^+$ logic, we first stratify the security policy to three layers. Then it is described that how policy rules in each layer are specified considering their non-monotonic nature.

A. Security Policy

Generally, security policy rules can be categorized as:

1. *Regular Access policy rules*, which are defined using the predicates *permission* and *prohibition*.
2. *Exception policy rules*, which are defined using the predicate *exception.* Different types of exceptions as well as our approach for exception policy rule definition are discussed in the next section.
3. *Default policy rules*, which are defined using the predicate *default*. More discussion on default policy rules is represented in Section V.C.

Following the above categorization, the security policy can be stratified to three layers, as depicted in Figure 3. Exception policy rules have more priority than regular access policy rules and they have higher priority than default policy rules. In fact, each regular access control rule can be considered as an *exception* to default rules because adding them may decrease the applicable domain (subject/ object/ action) of the default rules. Correspondingly, each exception policy rule can be considered as an *exception* to regular access policy rules. Since existing exception and default rules makes the system non-monotonic, such unification of interpretation of different rules clarifies the specification of them using the only non-monotonic feature of MKNF$^+$, i.e., negation as failure. More details on this issue would be found in the next section.

As our proposed model supports both permission and prohibition in both access policy rules and exception policy rules, a conflict resolution strategy is also required. The following conflict resolution strategies are mentioned in the literature:

- Denial-takes-precedence
- Permission-takes-precedence
- Least specific-takes-precedence
- Most specific-takes-precedence
- More prioritized-takes-precedence

Each of the above conflict resolution strategies can be employed in our proposed model. Due to lack of space, we do not further discuss this requirement. It is important to note that, our proposed model allows conflict among regular access policy rules but it does not allow conflict among exception policy rules. In other words, we cannot define a new exception rule over the existing exception rules.

B. Inheritance and Exception Policy Rules

Authorization inheritance is an important concept which should be considered in specification and inference of access control policies. In our proposed model, Rules (6) and (7) in the following are provided to enforce inheritance over role hierarchy. Enforcing inheritance over view and activity hierarchies are as the same. Using the aforementioned rules, each child role inherits the *permissions* of its parent nodes and each parent role inherits the *prohibitions* of its child roles. For example, if the role Administrator is considered as a sub-role of role Physician, all permissions granted to Physician are inherited by Administrator. The predicate *IsSubRoleOf*(ro_1, ro_2) means that the role ro_1 is the child node of role ro_2.

In presence of role (activity, or view) hierarchy, two general types of exceptions worth to be discussed:

1. Local exceptions: kind of exceptions that when defined locally on a role (activity, or view), they are not inherited over the role (activity, or view) hierarchy. The definition of local exception needs that user having sub-roles are not permitted to login as parent roles. Such a limitation contradicts the existence of role hierarchy.
2. General exceptions: kind of exceptions which are inherited over the role (activity, or view) hierarchy.

Due to the limitations of local exceptions, we take general exceptions into account in this paper.

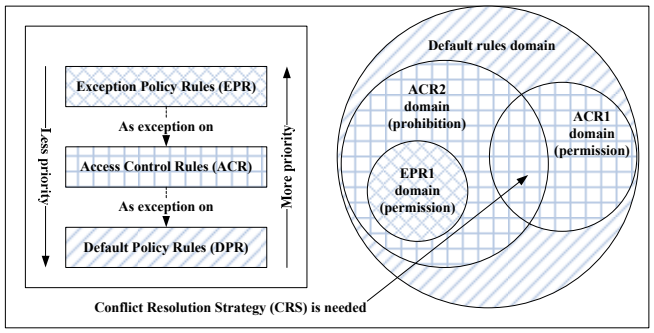

Figure 3. Different types of security policy rules

Each access control rule adds a set of permissions or prohibitions to the security knowledge base. So, an exception can be defined in either of two ways:

1. Rule-specific exception; which is defined on a role, view, or activity of a specific access control rule. For example, *permission(org,ro_1,av_1,v_1,c)* grants permission to role ro_1 to do activity av_1 on view v_1 within context *c*. According to the role hierarchy shown in Figure 4, the roles ro_2, ro_3, ..., ro_{10} inherit this permission. Now if we need to prohibit ro_2 from doing activity av_1 on view v_1, we should define it as an exception on *permission(org,ro_1,av_1,v_1,c)*. In this situation, if ro_2 inherits this permission due to the existence of another rule, a conflict happens. Note that exceptional permissions (prohibitions) should overwrite other prohibitions (permissions). Giving higher priority to exceptional permissions/prohibitions would be a logical strategy for resolving such conflicts.
2. Rule-independent exception; which is defined independently from the defined access control rules. So, an exceptional permission (prohibition) overwrites all other conflicting prohibitions (permissions).

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 *exception(Organization, Role, Activity, View, Context, Type)* is provided as exception policy rule definition. e*xception(org, ro, av, v, c, Per)* defines an exception to role *ro* to do activity *av* on view *v* within context *c* in organization *org*, and likewise, the e*xception(org, ro, av, v, c, Pro)* defines an exception to prohibits *ro* to do *av* on *v* within context *c*. Rule (8) translates abstract level exceptions to the concrete level ones. The predicate *is_excepted* is defined to show the concrete level exceptions. Rules (9) and (10) refine OrBAC axioms translating the abstract level permissions and prohibitions to the concrete level ones. In comparison to the OrBAC axiom, the condition *not is_excepted* is added to the axiom body. In fact, this condition means that a subject *su* has not permitted (prohibited) exceptionally to do action *ac* on object *ob,* unless it is explicitly defined. In addition, Rules (11) and (12) are defined for inheritance of exceptional permissions and prohibitions over role hierarchy. Inheritance rules for view and activity hierarchy are defined similarly. Finally, Rules (13) and (14) infer concrete level permissions and prohibitions from the concrete level exceptions.

$\mathbf{K}\,\text{ape}(ro_1), \mathbf{K}\,\text{permission}(org, ro_2, av, v, c),$ $\mathbf{K}\,\text{IsSubRoleOf}(ro_1, ro_2) \rightarrow \mathbf{K}\,\text{permission}(org, ro_1, av, v, c)$	(6)
$\mathbf{K}\,\text{ape}(ro_2), \mathbf{K}\,\text{prohibition}(org, ro_1, av, v, c),$ $\mathbf{K}\,\text{IsSubRoleOf}(ro_1, ro_2) \rightarrow \mathbf{K}\,\text{prohibition}(org, ro_2, av, v, c)$	(7)
$\mathbf{K}\,\text{exception}(org, ro, av, v, c, t), \mathbf{K}\,\text{IsEmployedIn}(su, ro),$ $\mathbf{K}\,\text{IsUsedIn}(ob, v), \text{IsConsideredAs}(ac, av),$ $\mathbf{K}\,\text{define}(org, su, ac, ob, c) \rightarrow \mathbf{K}\,\text{is\_excepted}(su, ac, ob, t)$	(8)
$\mathbf{K}\,\text{permission}(org, ro, av, v, c),$ $\mathbf{K}\,\text{IsEmployedIn}(su, ro), \mathbf{K}\,\text{IsUsedIn}(ob, v),$ $\mathbf{K}\,\text{IsConsideredAs}(ac, av), \mathbf{K}\,\text{define}(org, su, ac, ob, c),$ $\textbf{not}\,\text{is\_excepted}(su, ac, ob, Pro) \rightarrow \mathbf{K}\,\text{is\_permitted}(su, ac, ob)$	(9)
$\mathbf{K}\,\text{prohibition}(org, ro, av, v, c),$ $\mathbf{K}\,\text{IsEmployedIn}(su, ro), \mathbf{K}\,\text{IsUsedIn}(ob, v),$ $\mathbf{K}\,\text{IsConsideredAs}(ac, av), \mathbf{K}\,\text{define}(org, su, ob, ac, c),$ $\textbf{not}\,\text{is\_excepted}(su, ac, ob, Per) \rightarrow \mathbf{K}\,\text{is\_prohibited}(su, ac, ob)$	(10)
$\mathbf{K}\,\text{ape}(ro_1), \mathbf{K}\,\text{exception}(org, ro_2, av, v, Per),$ $\mathbf{K}\,\text{IsSubRole}(ro_1, ro_2) \rightarrow \mathbf{K}\,\text{exception}(org, ro_1, av, v, Per)$	(11)
$\mathbf{K}\,\text{ape}(ro_2), \mathbf{K}\,\text{exception}(org, ro_1, av, v, Pro),$ $\mathbf{K}\,\text{IsSubRole}(ro_1, ro_2) \rightarrow \mathbf{K}\,\text{exception}(org, ro_2, av, v, Pro)$	(12)
$\mathbf{K}\,\text{is\_excepted}(su, ac, ob, Per) \rightarrow \mathbf{K}\,\text{is\_permitted}(su, ac, ob)$	(13)
$\mathbf{K}\,\text{is\_excepted}(su, ac, ob, Pro) \rightarrow \mathbf{K}\,\text{is\_prohibitted}(su, ac, ob)$	(14)

C. Default Policy Rules

In some conditions, it is impossible to infer either *Is-permitted* or *Is-prohibited* for a request from the authorization knowledge base. However, the access control system finally should decide to accept or reject the request. Default policy is a solution proposed to resolve this problem.

In almost all the previous access control models *Open* and *Close* policy used as the default policy. However, default policy can be determined based on *context*. For example, a hospital may prefer to define Open policy for spatial objects at days and Close policy for them at nights. The hospital may also choose to apply Open and Close policies based on the user's roles. An organization may also desire to define Open policy for non-sensitive data and the Close policy for sensitive data.

The predicate *default(Organization, Role, Activity, View, Context, Type)* is used to define default rules. *default(org, ro, av, v, c, t)* means that the organization *org* forces default policy *t*, which can be Open or Close, to role *ro* to perform activity *av* on view *v* within context *c*. Covering all entities in a domain is an important requirement of default policy. To meet this requirement, we define the *Universal* context, which is *true* for all roles, activities, and views. Each organization should choose Open or Close policy for all roles, activities, and views in the Universal context. Additionally, all contexts are sub-context of Universal context. Rules (15) and (16) enforce these propositions. Default rules can be inherited over the role, activity, and view hierarchies. Each sub-role inherits Open policy from its parent nodes (Rule (17)) and each parent nodes inherits Close policy from its sub-roles (Rule (18)).

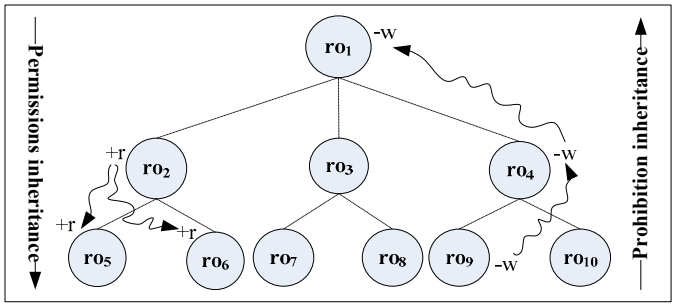

Figure 4. A sample role hierarchy

Since default policy rules support definition of both Open and Close policy, a conflict may occur among the Open and Close policies. The predicate *IsSubContextOf* is used to prioritize default rules. In fact, we use the following conflict resolution rule: "if default rules DR_1 and DR_2 have the same value for Role, Activity, and View entities and the DR_1 context is sub-context of the DR_2 context, DR_1 have more priority than DR_2".When the Open and Close policies are inferred from two default rules with *incomparable* contexts, the Close policy takes higher priority than the Open policy. The predicate has_more_specific_default(Organization, Role, Activity, View, Context, Type) is used to determine the default rules that would be overridden by some existing more specific default rules (in comparison to themselves). Rule (19) is defined to enforce transitivity relations for predicate IsSubContextOf. Rule (20) drives predicate has_more_specific_default from the context hierarchy. Rule (21) translates abstract level default to the concrete level one. Default rule with lower priority prevented to be translated to the concrete level one by using predicate *not has_more_specific_default* in the rule body. Finally Rules (22) and (23) consider concrete level Open and Close default policies to decide to reject or accept a request. Condition *not concrete_default(su,ob,ac,Close)* in the body of Rule (22) gives more priority to the Close policy.

$\rightarrow \mathbf{K}\,\text{define}(org, su, ac, ob, \text{Universal})$	(15)
$\mathbf{K}\,\text{ape}(co), \mathbf{K}\,\text{Context}(co) \rightarrow \mathbf{K}\,\text{IsSubContextOf}(co, \text{Universal})$	(16)
$\mathbf{K}\,\text{ape}(ro_2), \mathbf{K}\,\text{default}(org, ro_1, av, v, c, \text{Open}),$ $\mathbf{K}\,\text{IsSubRoleOf}(ro_2, ro_1) \rightarrow \mathbf{K}\,\text{default}(org, ro_2, av, v, c, \text{Open})$	(17)
$\mathbf{K}\,\text{ape}(ro_2), \mathbf{K}\,\text{default}(org, ro_1, av, v, c, \text{Close}, id),$ $\mathbf{K}\,\text{IsSubRoleOf}(ro_1, ro_2) \rightarrow \mathbf{K}\,\text{default}(org, ro_2, av, v, c, \text{Close})$	(18)
$\mathbf{K}\,\text{ape}(c_1), \mathbf{K}\,\text{ape}(c_2), \mathbf{K}\,\text{co}(c_3), \mathbf{K}\,\text{IsSubContextOf}(c_1, c_2),$ $\mathbf{K}\,\text{IsSubContextOf}(c_2, c_3) \rightarrow \mathbf{K}\,\text{IsSubContextOf}(c_1, c_3)$	(19)
$\mathbf{K}\,\text{default}(org, ro, av, v, c_1, t_1),$ $\mathbf{K}\,\text{default}(org, ro, av, v, c_2, t_2), \mathbf{K}\,\text{IsSubContextOf}(c_1, c_2)$ $\rightarrow \mathbf{K}\,\text{has\_more\_specefic\_default}(org, ro, av, v, c_2, t_2)$	(20)
$\mathbf{K}\,\text{default}(org, ro, av, v, c, t), \mathbf{K}\,\text{IsEmployedIn}(su, ro),$ $\mathbf{K}\,\text{IsUsedIn}(ob, v), \mathbf{K}\,\text{IsConsideredAs}(ac, av),$ $\mathbf{K}\,\text{define}(org, su, ob, ac, c),$ $\mathbf{not}\,\text{has\_more\_specefic\_default}(org, ro, av, v, c, t)$ $\rightarrow \mathbf{K}\,\text{concrete\_default}(su, ob, ac, t)$	(21)
$\mathbf{not}\,\text{is\_permitted}(su, ob, ac), \mathbf{not}\,\text{is\_prohibited}(su, ob, ac),$ $\mathbf{K}\,\text{concrete\_default}(su\,ob, ac, \text{Open}),$ $\mathbf{not}\,\text{concrete\_default}(su, ob, ac, \text{Close})$ $\rightarrow \mathbf{K}\,\text{is\_permitted}(su, ob, ac)$	(22)
$\mathbf{not}\,\text{is\_permitted}(su, ob, ac), \mathbf{not}\,\text{is\_prohibited}(su, ob, ac),$ $\mathbf{K}\,\text{concrete\_default}(su, ob, ac, \text{Close})$ $\rightarrow \mathbf{K}\,\text{is\_prohibited}(su, ob, ac)$	(23

VI. CASE STUDY

To demonstrate the applicability of the proposed model, a case study is discussed in this section. Figure 5 shows a partial definition of specific ontology for a healthcare domain. In addition to general classes defined in the proposed upper level ontology, a number of concrete sub-classes are defined to model specific context in a given environment (e.g., abstract class *Location* is classified into two sub-classes *LogicalLocation* and *PhysicalLocation*). Also some individuals for different entities of OrBAC are added to the upper level ontology. For example, individual *Guest* is a role such thath all subject are employed in it.

Suppose that hospital H_1 uses the aforementioned ontology. Table V shows some contexts defined by H_1. Context Universal (Rule (24)), Attending_Physician (Rule (25)), Non_Attending_Physician (Rule (26)), and Emergency (Rule (27)) have been already discussed. Rule (28) defines Internal_IP context and Rule (29) applies closed-world assumption to it to define the context External_IP. According to Rule (28), the context Internal_IP is *true* for subject *su* if he uses a host located in H1's internal network.

Table VI represents the security policy for H_1. H_1 chooses Close policy as its default policy for Universal context using Rule (30). Also it enforces Open policy for subjects who have been employed in role Medical_Staff and uses internal IP

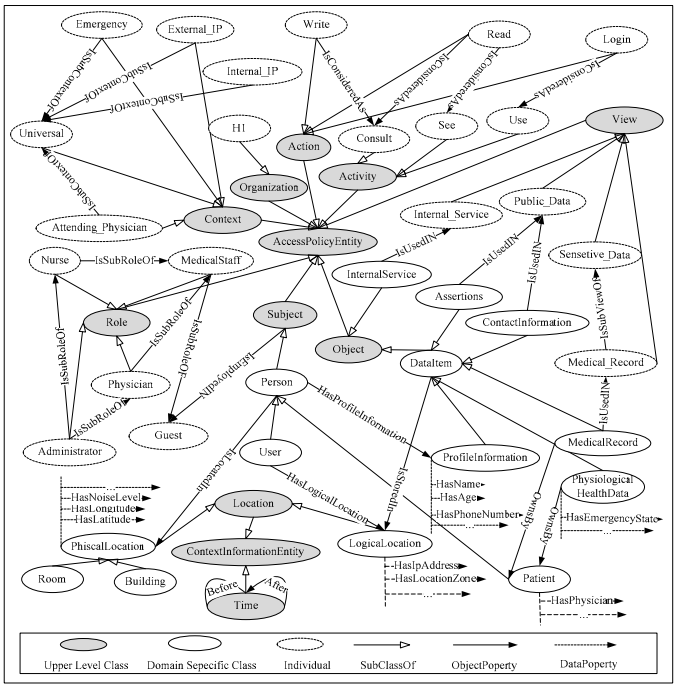

Figure 5. Partial definition of a specific ontology for healthcare domain

address to access the hospital's internal services (Rule (31)). Rule (32) grants permission to subjects employed in role Physician and are in the patient's attending physician team to consult on medical records belonging to the patient. Rule (33) prohibits physicians whose context Non_Attending_Physician is true to access the medical records. Rule (34) grants access to see the public data to role Guest. Rule (35) defines an exception which allows the physicians not belonging to the patient's attending team to access his medical record when he is in emergency condition. Suppose H_1 wants to prohibit all accesses to some sensitive data such as the medical records from the external IP addresses (unsecure networks). It can use Rule (36) to prohibit all accesses to sensitive data from role Administrator. By prohibiting role Administrator and considering the inheritance over role hierarchy, all other roles are prohibited automatically. Finally, H_1 can use Rule (37) to enforce denial-takes-precedence as its conflict resolution strategy.

Table V. Some contexts definition

$\rightarrow \mathbf{K}\,\text{define}(Hl, su, ac, ob, \text{Universal})$	(24)
$\mathbf{K}\,\text{ape}(p), \mathbf{K}\,\text{ape}(su), \mathbf{K}\,\text{ape}(ob), \mathbf{K}\,\text{HasPhysician}(p, su),$ $\mathbf{K}\,\text{OwnsBy}(ob, p) \rightarrow \mathbf{K}\,\text{define}(H_1, su, ac, ob, \text{Attending\_Physician})$	(25)
$\textbf{not}\,\text{define}(H_1, su, ac, ob, \text{Attending\_Physician})$ $\rightarrow \text{Define}(H_1, su, ac, ob, \text{Non\_Attending\_Physician})$	(26)
$\mathbf{K}\,\text{ape}(p), \mathbf{K}\,\text{ape}(hd), \mathbf{K}\,\text{ape}(mr),$ $\mathbf{K}\,\text{Medical Record}(mr), \mathbf{K}\,\text{OwnsBy}(mr, p),$ $\mathbf{K}\,\text{OwnsBy}(hd, p), \mathbf{K}\,\text{HasEmergencyState}(hd, \text{True}),$ $\rightarrow \mathbf{K}\,\text{define}(Hl, su, ac, mr, \text{Emergency})$	(27)
$\mathbf{K}\,\text{ape}(su), \mathbf{K}\,\text{cie}(l\text{loc}), \mathbf{K}\,\text{HasLogicalLocation}(su, l\text{loc}),$ $\mathbf{K}\,\text{HasLocationZone}(l\text{loc}, Hl\_Net)$ $\rightarrow \mathbf{K}\,\text{define}(H_1, su, ac, ob, \text{Internal\_IP})$	(28)
$\textbf{not}\,\text{define}(Hl, su, ac, ob, \text{Internal\_IP})$ $\rightarrow \mathbf{K}\,\text{define}(Hl, su, ac, ob, \text{External\_IP})$	(29)

Table VI. Security policy rules

Default Policy Rules	
$\text{default}(H_1, ro, av, v, \text{Universal}, \text{Close})$	(30)
$\text{default}(H_1, \text{Medical\_Staff}, \text{Use}, \text{Internal\_Service}, \text{Internal\_IP}, \text{Open})$	(31)
Access Policy Rules	
$\text{permission}(H_1, \text{Physician}, \text{Consult}, \text{Medical\_Record}, \text{Attending\_Physician})$	(32)
$\text{prohibition}(H_1, \text{Physician}, \text{Consult}, \text{Medical\_Record},$ $\text{Non\_Attending\_Physician})$	(33)
$\text{permission}(H_1, \text{Guest}, \text{See}, \text{Public\_Data}, \text{Universal})$	(34)
Exception Policy Rules	
$\text{exception}(H_1, \text{Physician}, \text{Consult}, \text{Medical\_Record}, \text{Emergency})$	(35)
$\text{exception}(H_1, \text{Administrator}, av, \text{Sensetive\_Data}, \text{External\_IP})$	(36)
Conflict Resolution Strategy(Denial-Takes- Precedence)	
$\mathbf{K}\,\text{is\_permitted}(su, ac, ob), \mathbf{K}\,\text{is\_prohibited}(su, ac, ob)$ $\rightarrow \mathbf{K}\,\text{is\_prohibited}(su, ac, ob)$	(37)

VII. Conclusion and Future Work

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 to 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 using MKNF$^+$. In the model, DL is used to model main entities of OrBAC as well as the context information. In addition, MKNF$^+$ rules, which are used to express security policy rules and context information, enable the model to support non-monotonic reasoning in presence of incomplete context information. The default policy and exception policy rules are defined using negation as failure, as supported in the MKNF$^+$ logic.

We demonstrated how incomplete context information can be handled using negation as failure based on the closed-world assumption and also how default context information can be specified in a context-ware access control model. However, handling inconsistent and uncertain context information in such an access control model remains as future work.

References

[1] A. Kalam et al., "Organization Based Access Control," in *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, 2003, pp. 120-132.

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

[3] A. Noorollahi and M. S. Fallah, "A Logical View of Nonmonotonicity in Access Control Model," in *International Conference on Security and Cryptography (SECRYPT 2011)*, 2011, pp. 472-481.

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

[5] C. Bettini et al., "A Survey of Context Modelling and Reasoning Techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161-180, Apr. 2010.

[6] E. J. Ko, H. J. Lee, and J. W. Lee, "Ontology-Based Context Modeling and Reasoning for U- HealthCare," *IEICE - Transactions on Information and Systems*, vol. E90–D, no. 8, pp. 1-10, 2007.

[7] B. Motik and R. Rosati, "Reconciling Description Logics and Rules," *Journal of the ACM (JACM)*, vol. 57, no. 5, p. 30, 2010.

[8] K. Henricksen and J. Indulska, "Modelling and Using Imperfect Context Information," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004, pp. 33-37.

[9] F. Cuppens and N. Cuppens-boulahia, "Modelling Contextual Security Policies," *International Journal of Information Security*, vol. 7, no. 4, 2008.

[10] T. Y. C. Woo and S. S. Lam, "Authorization in Distributed Systems˜ : A Formal Approach," in *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, 1992, pp. 33-50.

[11] E. Bertino et al., "A Logical Framework for Reasoning on Data Access Control Policies," in *CSFW '99 Proceedings of the 12th IEEE workshop on Computer Security Foundations*, 1999, pp. 175-190.

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

[13] N. Boustia and A. Mokhtari, "A Dynamic Access Control Model," *Applied Intelligence*, vol. 36, no. 1, pp. 190-207, Sep. 2010.

[14] 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.

[15] R. Rosati, "DL + log˜ : Tight Integration of Description Logics and Disjunctive Datalog," in *The Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR2006)*, 2006, pp. 68-78.

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