Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 96 (2016) 245 – 254

Procedia Computer Science

20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

Dynamic delegation based on temporal context

Ouarda Bettaz[a]*, Narhimene Boustia[b], Aicha Mokhtari[c]

[a]RIIMA Laboratory, ENS, Bachir El Ibrahimi, B.P, Kouba 16050, Algiers, Algeria
[b]RIIMA Laboratory, University of Blida 1, Route de Soumaa, BP 270, Blida, Algeria
[c]RIIMA Laboratory, USTHB, BP 32 El Alia 16111 Bab Ezzouar, Algiers, Algeria

Abstract

Delegation is a very important part of the administrative process in access control systems; it provides resiliency and flexibility regarding to the management procedure. Delegation is the process of granting a specific authorization from a user to another user of the same system to carry out some functions on his behalf. The delegation, although widely used, is modeled in very little security policies because of its complexity. In this paper we aim to consider the delegation dynamically based on temporal context, to this end we redefine delegation for OrBAC using temporal nonmonotonic description logic. OrBAC is an access control model; it provides the mean to specify contextual authorizations, which facilitates modeling delegation features such as temporary delegation, multiple delegation, revocation, etc. The description logic that we use for the re-formalization process is T-JClassic$_{\delta\varepsilon}$. This logic gives the mean to specify nonmonotonic authorizations, and a better representation of the temporal aspects specific to a given delegation. This new representation augments the expressivity of the model and therefore it facilitates even more the representation and the management of the delegation characteristics.

Peer-review under responsibility of KES International

Keywords: access control; delegation; revocation; nonmonotonic description logic; temporal description logic.

1. Introduction

Delegation of access rights is a very important function in the business world and in health care, this fact forces the organizations to rethink their security policies and revaluate their efficiencies. Organizations establish a set of security policies that regulates how information and resources should be managed in a dynamic environment. To

* Corresponding author. Tel.: 00 213 21 29 75 11; fax: 00 213 21 28 20 67.
E-mail address: ouarda.bettaz.nehab@gmail.com.

Peer-review under responsibility of KES International
doi:10.1016/j.procs.2016.08.137

reach these requirements, we present a delegation approach as a process for supporting organizational flexibility in information systems, and ensuring delegation of authority in access control system. However, only few works were done in this field[9, 10, 11, 12, 21, 32, 33]. In fact the majority was dedicated to RBAC Model[30], RBAC is a role based access control model, the shortcoming of this model is the lack of expressiveness, therefore it is not adequate enough to deal with all delegation requirements such as permanence, level of delegation, revocation, etc. To overcome this expressiveness problem RBAC was extended to OrBAC[1, 25], by adding new types of roles and permissions. A work managing the delegation in OrBAC was performed[12], the authors propose a flexible and complete approach for OrBAC to deal with various security requirements including the delegation characteristics. However OrBAC is represented using first order logic. The problem with this approach is that first order logic is semi decidable, plus the fact that we cannot represent an exception of exception of a context, and so on, while delegating an authorisation. The other point is that the conjunction or disjunction of contexts; which allows the use of more than one context at once, are used as arguments inside OrBAC first order logic predicates, this makes the decidability even more complex. Delegation, as the term is used in this paper, is motivated by a situation in which users are temporarily unable to perform one or more of their tasks, because they are too busy, or away from their job due to illness or vacation. The user who normally has certain permissions, called the delegator, grants one or more of these permissions (or a role) to a delegatee. It is characterized by being temporary, and can be accompanied by a time limit. In this paper we aim to present a new delegation model. This is performed by a redefinition of the delegation rules of OrBAC. We will use the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$ [14] for this purpose. This approach provides a better expressivity than the delegation model of OrBAC, while maintaining a polynomial complexity. The improved expressivity of our model comparing to OrBAC, is translated by the fact that it provides the possibility to represent an exception of an exception of a context, and so on, in the definition of delegation's permission. This maneuver is not possible in OrBAC. In addition, the utilization of the temporal components of T-JClassic$_{\delta\varepsilon}$ [14] allows a better representation of the temporal aspects proper to a given delegation, and also a better management of the delegation characteristics, where time is an important factor. We point here to the fact that the access control model OrBAC was extended with temporal elements in a previous research[15, 17]. It was actually reformalized by using the temporal nonmonotonic description logic T-JClassic$_{\delta\varepsilon}$. The purpose behind this process was to represent a temporal context for the sake of representing temporal authorizations, and thus providing the possibility to deal with temporal access queries. We will present in this paper a new definition of AdOrBAC[23] the administration model of OrBAC, Subsequently we will redefine the delegation rules. The two steps are realized using the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$[16].

This paper is organized as follows: In the second section the preamble is divided into two parts, the first part is about OrBAC, AdOrBAC and its delegation process, and in the second part we present the temporal nonmonotonic description logic T-JClassic$_{\delta\varepsilon}$**.** In section 3 we present our new delegation model with it characteristics, illustrated with examples. We end this paper by a related work in section 4 and a conclusion in section 5.

2. Preamble

Before proceeding we will need to make a preface about the access control model OrBAC with its administration part and its delegation model, and provide a slight state of art about T-JClassic$_{\delta\varepsilon}$ to better understand the contribution of our work.

2.1. OrBAC, AdOrBAC and delegation model

OrBAC is an organization based access control model[1, 25]. It is an enhancement of RBAC Role based access control[30] in which static authorizations are granted to users depending on their Roles. However in some situations some conditions must be satisfied to activate security rules, these conditions whether temporal, spatial or others were defined in OrBAC as the notion of context[22, 24]. In this model authorizations are granted to users depending on their roles in the organization and also on the context[22, 24]. OrBAC is therefore a dynamic access control model. The formalization of OrBAC is based on first order logic. For instance the security policies of the hospital *X* can comprise the following facts: *Permission (X, Doctor, Consult, Medical–Record, Emergency)*. This indicates that the hospital *X* grants to doctors the permission to consult any medical record in an emergency context.

AdOrBAC[23] is the administration model for OrBAC. It uses the concept of views to define the administration tasks. The language rules used to define permission to administer the policy is completely similar to the rest of OrBAC. The two administrative views used in AdOrBAC are Role Assignment View and License View [12]. They are defined as follows: *License view;* this view is used to denote and manage the security policy. Inserting an object in this view will enable an authorized user to assign permission to a role or assign permission to a user. Objects belonging to the License View have the following attributes: Grantee: subject to which the license is granted, Privilege: action permitted by the license, Target: object to which the license grants an access and Context: specific conditions that must be satisfied to use the license. The existence of a valid license is interpreted as permission by the following rule: *Permission (Subject, Action, Object, Context):- Use (L, license), Grantee (L, Subject), Privilege (L, Action), Target (L, Object), Context (L, Context)*.

Role assignment view; inserting an object in this view will enable to assign a user to a role. It is associated with the following attributes: *Assignee*: subject to which the role is assigned and *Assignment*: role assigned by the role assignment. There is the following rule to interpret objects of the role assignment view:

Empower (Subject, Role):- Use (RA, Role –Assignment), Assignee (RA, Subject), Assignment (RA, Role).

Deleting an object from these views will enable a user to perform a revocation.

In OrBAC delegation allows giving a specific user privileges without giving this privilege to all people with the same role as him[12]. The delegation is modeled by the use of the notion of contexts and administrative views as defined in AdOrBAC. Two types of views are denoted: the *License–Delegation View* to delegate rights (partial delegation), and the *Role–Delegation View* to delegate roles (total delegation), these views are defined as follows:

Permission (Grantor, Action, Object, Context):- Use (L, License–Delegation), Grantee (L, Grantor), Privilege (L, Action), Target (L, Object), Context (L, Context).

Empower (Grantor, Role):- Use (RD, Role–Delegation), Assignee (RD, Grantor), Assignment (RD, Role).

The management of delegation policy consists of defining which grantor (role or user) have an access to these views and in which context[12]. This is defined by the following permissions:

Permission (Grantor, Delegate, License–Delegation, Context).

Permission (Grantor, Delegate, Role–Delegation, Context).

For instance in the medical domain, we can consider two users: *James* a doctor and *Alice* a nurse. The role nurse is usually not allowed to have access to the view *Patient–Records*. However, *James* decides to delegate permission to *Alice* to consult the records of his patients. Clearly, *James* must have permission to delegate this right. For this purpose the administrator should first create the administrative view *Record–Delegation* as follows:

Use (L, Record–Delegation):- Use (L, License–Delegation), Privilege (L, Consult), Target (L, Patient–Records).

The view *Record–Delegation* is derived from *License–Delegation* view and only contains licenses to consult *Patient–Records*. The administrator should now give the role doctor the permission to delegate licenses in this view:

Permission (Doctor, Delegate, Record–Delegation, Nominal).

Using this authorization *James* can delegate to *Alice* a permission to consult the medical records of his patients (*James–Patient–Records*), which is a *Sub-View* of *Patient–Records*, *Nominal* here is the default context.

Permission (Alice, Consult, James–Patient–Records, Nominal).

We notice here that if the grantor has the permission to delegate a license *L* then he also has the permission to delegate its sub license *L'*.

The different characteristics of delegation[12] will be just briefly described here; we will consider them in detail in the upcoming sections with their new formalization: *Permanence* refers to types of delegation in terms of their time duration, *Monotonicity* it means that upon delegation the grantor maintains the permission he has delegated, with a non-monotonic delegation, the grantor loses this permission for the duration of the delegation, *Multiple delegation* refers to the number of grantees to whom a grantor can delegate the same right at any given time, *Level of*

delegation it defines whether or not each delegation can be further delegated and how many times, *Revocation* it is the process of withdrawing the delegated license or role.

2.2. T- JClassic$_{\delta\varepsilon}$

We will use for the formalization of our new delegation model the temporal nonmonotonic description logic T-JClassic$_{\delta\varepsilon}$[14]. Before presenting T- JClassic$_{\delta\varepsilon}$, we outline first the nonmonotonic description logic and the temporal description logic to show how it emerged.

Nonmonotonic description logic: Description logics are good formalisms for knowledge base representation[8]. However classical forms of description logics do not permit to represent neither default nor exception facts about concepts, for example: they do not allow representing the fact that all birds fly by default, but a penguin is a bird that exceptionally doesn't fly. The impossibility of representing this kind of information leaves the knowledge base partially defined which subsequently affects the inference process. The solution to represent such kind of knowledge is to rely on nonmonotonic reasoning that is based on the use of default description logic. Many approaches were proposed in the literature[7, 27, 28, 29]. The problem with these approaches is that they use a limited form of default reasoning; where concepts are defined only by using strict properties while default knowledge is represented using incidental rules, considering the fact that most of concepts can't be just defined by the use of strict properties, that will leave the knowledge base inevitably partially defined, consequently the classification process won't be complete. The approach that overcomes this problem was proposed by Coupey and Fouqueré [20]. In fact they developed a new nonmonotonic description logic named AL$_{\delta\varepsilon}$ that permitted the introduction of the notion of default and exception in concepts definition, it was elaborated by adding to the description logic AL[8] two connectives: (δ) to represent default facts and (ε) to represent exception facts. This language was improved by the addition of connectors from C-classic which permitted to augment its expressivity and thus make it usable from a practical point of view. This new language was called JClassic$_{\delta\varepsilon}$[17, 18]. Using this description logic we can define the concept *Tree* as having by default branches and always having a trunk and roots: *Tree* ≡ *δWith–branches* ⊓ *With–trunk* ⊓ *With–roots*. Now if we want to define the concept *Scion* as being a tree that is by default one year old and exceptionally has branches we will write it this way: *Scion* ≡ *δOne–year–old* ⊓ *Tree* ⊓ *With–branches*$^{\varepsilon}$. In this example the concept *Scion* that is subsumed by the concept *Tree* will only inherit the properties *With–trunk* and *With–roots*, but not the property *With–branches* since this property is an exception for the concept *Scion*.

Temporal description logic: Temporal Logics are designed for representing and reasoning about information qualified in term of time. They have been widely used in several domains such as databases, natural language processing, planning, etc. In the research field on temporal description logic (TDL)[2, 3, 4, 5, 8, 19] two approaches for modeling the notion of time were considered: the modal temporal logic and the reified temporal logic[26]. T- JClassic$_{\delta\varepsilon}$ was modeled with the modal approach in which the connectors □ and ◊ represents respectively the notion of (always in the future) and (sometimes in the future). The flow of time can be taken from two different angles, we can actually consider time as a set of points (instances) or as a set of intervals. In [2, 5, 8, 19], the different approaches on temporal logic based on points and intervals have been widely spread. T- JClassic$_{\delta\varepsilon}$ uses an interval based approach to define the specific interval at which a concept is valid. Concerning this approach many studies were undertaken. Artale and Franconi [2, 3, 5] put into evidence a TDL inspired by Schmiedel's[31] approach, that they restricted by discarding the □ operator for decidability matters. Example[6]: ◊ *(X Y) (Y starts X).(Student@ X* ⊓ *Bachelor–student @Y)*. In this example, we have two intervals *X* and *Y*, where *X* and *Y* start at the same time but *Y* is ended before *X*. So the described persons are students during the interval *X* and they are specifically Bachelor students for the initial sub-interval *Y* of *X*. The temporal part TL that was used for the conception of T-JClassic$_{\delta\varepsilon}$ is the one used in the TDL defined by Artale and Franconi [2, 3, 5].

T-Jclassic$_{\delta\varepsilon}$: it is a temporal nonmonotonic description logic. It was developed to permit a better management of the time aspect in a variety of domains such as reasoning about actions and plans, enhancing natural languages comprehension and also allowing the improvement of access control. T-JClassic$_{\delta\varepsilon}$ allows representing temporal concepts while having default knowledge. Differing from the existing temporal description logics where temporal components are added to classical description logics. We will just describe the part of the syntax[14] of T-JClassic$_{\delta\varepsilon}$ that we will be needing for our work, it consists of: a set of atomic concepts P and atomic roles R, the two constants ⊤ (Top) and ⊥ (Bottom) that represent respectively the universal and the bottom concept, a set of individuals I

called 'classic individuals', the concepts C and D, the unary connectives δ (Default) and ε (Exception), the binary conjunction $\sqcap$, the quantifier $\forall$that enables universal quantification on role values, and the temporal qualifier @ to represent the interval '*X*' at which a concept C applies, u is a real number, n is an integer, *Ii* are 'classic individuals'.

Syntax of T-JClassic$_{\delta\varepsilon}$

C, D → P	(Atomic concept)
|⊤	(Universal Concept)
|⊥	(Bottom concept)
| ¬P	(Atomic negation)
| C ⊓ D	(Intersection)
| Min u	(u is a real number)
| Max u	(u is a real number)
| ONE-OF{I1,…In}	(Concept in extension)
| R FILLS {I1,…In}	(Subset of value for R)
| R AT-LEAST n	Cardinality for R (minimum)
| R AT-MOST n	Cardinality for R (maximum)
| ∀R.C	(Universal quantifier)
| δ C	(Concept C by default)
| C^{ε}	(Exception to the concept C)
| C@X	(Qualifier)

Using this description logic we can represent temporal aspects, the properties: default, exception, exception of exception, and so on. For instance in the case of access control in the medical domain, we can define the concept *Doctor* as being a *Staff member* that *Exercises* officially his function by default and that has the right to *Access* the medical database records of patients during *Working hours*: *Doctor ≡ Staff–Member ⊓ δExercise ⊓ Access – Mdb–Records@(working Hours)*.

Now we can define the concept Resident as a Doctor that exceptionally doesn't Exercise officially since he is still a student: *Resident ≡ Doctor ⊓ Exercise*$^{\varepsilon}$

Here the concept *Resident* will inherit the property *Staff Member* and the right to *Access* the medical database records during working hours but not the property *Exercise* since it is an exception for the concept *Resident*. In the case where we have a context of *Emergency* another exception on the concept *Exercise* is applied for *Resident*:

Resident ⊓ Emergency ≡ Doctor ⊓ (Exercise $^{\varepsilon}$*)* $^{\varepsilon}$

In this case, the exception over an exception omits the exception, therefore in an emergency context *Resident* has the right to *Exercise*. We just point here to the fact that OrBAC does not allow the representation of an exception of exception regarding to a concept definition.

3. Delegation

In this section we present our new delegation model. This approach proposes a redefinition of the delegation model of OrBAC[12] using the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$[14]. OrBAC is formalized with predicates from first order logic. Our approach will augment the expressivity of the model while maintaining a polynomial complexity; it actually permits to represent an exception of an exception for the context, which is not possible in OrBAC, and the use of temporal aspects allows a better representation of the delegation parameters. For this purpose we initially redefine the administration model.

3.1. AdOrBAC using T- JClassic$_{\delta\varepsilon}$

We redefine here the rules for *License* and *Role –Assignment Views* using the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$, the two permissions are represented as follows: The license *View* is rewritten as follows: *δPermission ⊑ UseL.License ⊓ GranteeL.Subject ⊓ PrivilegeL.Action ⊓ TargetL.Object.*

δ represents the context by default, *UseL, GranteeL, PrivilegeL* and *TargeL* are binary relations between the concept *Permission* and respectively the concepts *License, Subject, Action* and *Object*, which are all subsumed by $\top$ (the most general concept). In this view by default an authorized user assigns permission to a role or to a user.

The Role assignment View is rewritten as follows: $Empower \sqsubseteq UseRA.Role–Assignment \sqcap AssigneeRA.Subject \sqcap AssignmentRA.Role.$

UseRA, AssigneeRA, AssignmentRA are binary relations between the concept *Empower* and respectively the concepts *Role–Assignment, Subject* and *Role* which are all subsumed by $\top$. This axiom enables to assign a user to a role.

3.2. Delegation using T- JClassic$_{\delta\varepsilon}$

In this part we come up to our initial objective; redefining the delegation rules of OrBAC with T- JClassic$_{\delta\varepsilon}$. We illustrate with examples using the temporal components proper to this logic, and show how we can represent an exception of exception, and so on, while using delegation permission. First we need to rewrite the axioms of the two types of delegation views namely the *License–Delegation View* for partial delegation, which permits to delegate rights, and the *Role–Delegation View* for total delegation, that allows delegating Roles, these views are redefined as follows: $\delta Permission \sqsubseteq UseL.License–Delegation \sqcap GranteeL.Grantor \sqcap PrivilegeL.Action \sqcap TargetL.Object$.

Here by default a grantor has the right to delegate permission to a grantee.

$Empower \sqsubseteq UseRD.Role–Delegation \sqcap AssigneeRD.Grantor \sqcap AssignmentRD.Role.$

The above axiom allows the grantor to delegate a Role to a grantee. Now for the process of defining which grantor has an access to these views and in which context, we rewrite the permission axioms that permit this action as follows: $\delta Permission \sqsubseteq PermissionGr.Grantor \sqcap PermissionDL.Delegate \sqcap PermissionLDL.License–Delegation.$

$\delta Permission \sqsubseteq PermissionGr.Grantor \sqcap PermissionDL.Delegate \sqcap PermissionRDL.Role–Delegation.$

We can reconsider the example cited previously in this paper about the doctor who delegates his rights to a nurse; Doctor *James* decides to delegate permission to nurse *Alice* to consult the records of his patients. Before assigning *Role* doctor the permission to delegate this right. The administrator creates first the *Record–Delegation View* and then he grants the *Role* doctor the permission to delegate licenses in the formerly defined view, we model these two axioms as follows: *Record–Delegation View* derived from *License– Delegation View*:

$Use \sqsubseteq UseL.License–Delegation \sqcap PrivilegeL.Consult \sqcap TargetL.Patient–Records.$

The following is the permission that allows the *Role* doctor to delegate licenses in *Record– Delegation View:*

$\delta Permission \sqsubseteq PermissionD.Doctor \sqcap PermissionDL.Delegate \sqcap PermissionRCDL.Record–Delegation.$

Now by default *James* can delegate to *Alice* a permission to consult his patient's medical records (*James–Patient–Records*); that is a sub-license of *Patient–Records* as follows:

$\delta Permission \sqsubseteq PermissionA.Alice \sqcap PermissionCL.Consult \sqcap PermissionJPR.James–Patient–Records.$

Using the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$ we can consider the above delegation to be valid at a given temporal context for example *Working–Hours*, to say that *James* delegates to *Alice* the above permission only during the working hours. We can represent this delegation using the @ operator and write it as follows:

$\delta Permission@(Working–Hours) \sqsubseteq PermissionA.Alice \sqcap PermissionCL.Consult \sqcap PermissionJMR.James–Patient–Records.$

Now if we put an exception on this permission, let say in the case where the medical records are being updated, than exceptionally *Alice* will not be delegated this right, we define it as follow using the (ε) exception operator:

$\delta Permission@(Working–Hours)^{\varepsilon} \sqsubseteq PermissionA.Alice \sqcap PermissionCL.Consult \sqcap PermissionJMR.James–Patient–Records.$

We can also define an exception to the exception of the context on the delegation using T-JClassicδε. For instance we consider the exception context (*Emergency*) and we define an exception on the above permission:

$\delta Permission@(Working–Hours)^{\varepsilon})^{\varepsilon} \sqsubseteq PermissionA.Alice \sqcap PermissionCL.Consult \sqcap PermissionJMR.\ James–Patient–Records.$

Here the exception on the exception will omit the exception, so it brings us to the permission by default where *Alice* is delegated the right to consult *James–Patient–Records*, in the case of the exception context *Record–Update* alongside with the exception context *Emergency.* We present in the following the re-formalization of the characteristics of delegation using T- JClassic$_{\delta\varepsilon}$.

a)Permanence: Delegation can be determined in term of its durability, in fact we can specify the time period at which the delegation is valid, this is referred to as temporal context, in our above example a second license can be created in which doctor *James* delegates a permission to nurse *Alice* to consult his patients medical records for a specific period of time, let say during his *rest–hours,* we will use here the @ operator to append this permission to the time interval *Rest–Hours,* this is formulated as follows:

$Permission@(Rest–Hours) \sqsubseteq PermissionA.Alice \sqcap PermissionCL.Consult \sqcap PermissionJPR.James–Patient–Records.$

This delegated permission is valid only during *James*'s rest hours. We can now consider that we have an exception of the context *Rest–Hours*, let say *Working–Hours*, According to our example, we put an exception on the permission delegated to *Alice*, that states that she has the right to consult *James–Patient–Records* records during *James Rest–Hours*, the excepted permission will be (*Alice* has the right to consult *James–Patient–Records* during *Working Hours*), hence *Alice* has a permission to consult the records the whole time during working hours, which expresses in this case the permanence of the permission.

b)Monotonicity: In the case of a monotonic delegation, the grantor preserves the delegated permission; this is the case of the examples cited previously. However to represent a non-monotonic delegation; where the grantor actually loses the permission he delegates, a *License– Transfer View* is created, this view is a *sub–view* of *License–Delegation View*, thus inserting an object in this view will create a permission to the grantee and a prohibition to the grantor, associated with a highest priority *Max*. We remodel the *License–Transfer View* using the temporal operator @ from T- JClassic$_{\delta\varepsilon}$ as follows: $Use \sqsubseteq UseL.License–Transfer.$

$Prohibition@(Delegation–Duration) \sqsubseteq UseL.\ License–Transfer \sqcap\ GrantorL.Subject \sqcap PrivilegeL.Action \sqcap TargetL.Object.$

The grantor loses this permission only during the period of delegation (Delegation Duration), therefore the temporal context of the prohibition is the same one as the temporal context of the delegated permission.

c)Multiple Delegation: A grantor may delegate the same right to a group of people at any given time; this is referred to as multiple delegation. The administrator fixes the number *Nm* of grantees by using the context *Max–Multi–Delegation*. The permission that represents this delegation is redefined using T- JClassic$_{\delta\varepsilon}$ as follows:

$\delta Permission \sqsubseteq PermissionS.Subject \sqcap PermissionD.Delegate \sqcap PermissionV.View \sqcap PermissionM.Max–Multi–Delegation.$

We redefine in what follows the context *Max–Multi–Delegation*, for this purpose we count the delegation number concerning the same grantor and the same right:

$Hold \sqsubseteq UseL.License–Delegation \sqcap GrantorL.S \sqcap CountL'.C.$
$Count \sqsubseteq UseL'.License–Delegation \sqcap GrantoL'.S \sqcap Equivalent–LicenseL.L' \sqcap Nm' \leq Nm.$

L and *L'* are equivalent; hence they represent the same license.

Example: *δPermission ⊑ PermissionJ.James ⊓ PermissionDL.Delegate ⊓ PermissionRCDL.Record–Delegation ⊓ PermissionM.Max–Multi–Delegation=1*

In this example the context *Max–Multi–Delegation = 1*, which refers to simple delegation, therefore *James* can delegate the right to consult the medical records of his patients only once. In the case where the context *Max–Multi–Delegation* is not used, it follows that the grantor can grant an unlimited number of licenses. Now if we put an exception on the above permission it becomes by default a multiple delegation.

d)Level of Delegation: The level of delegation feature permits to identify if a given delegation can be further delegated and how often. The *Grant–Option–License View* was created for this purpose; we redefine it using T-JClassic$_{\delta\varepsilon}$ as follows: *δPermission ⊑ UseL.Grant–Option–License ⊓ GranteeL.U ⊓ TargetL.License.*

The context *Valid–Level* is redefined as follows:

Hold ⊑ UseL'.Grant–Option–License ⊓ Sub–LicenseL'.L ⊓ GrantorL'.U ⊓ LevelL.V⊓ LevelL'.V'⊓ V' < V.

Level represents the number of authorized delegation steps.

According to our example if we consider the fact that *James* grants *Alice* the permission to delegate his license *L1* (Consult his patient's medical records) with delegation level equals to 3. For this reason *James* generates another license *L3* in the *Grant–Option–License View* with the following characteristics: *Grantee: Alice, Privilege: Delegate, Target: L1, Level: 3, Context: Nominal*. This corresponds to the following rule:

δPermission ⊑ PermissionA.Alice ⊓ PermissionD.Delegate ⊓ PermissionL.L1 ⊓ PermissionV.Valid–Level.

In this permission by default *Alice* can delegate the license *L1* or a *sub–license* of *L1* using the context *Valid–Level*. This means also that she can generate another license *L4*, to grant another user the permission to delegate the license *L1*. In this case the delegation level of *L4* must be lower than 3, given that the delegation level of *L3* is equal to 3. The grantor can also limit the scope of the delegation by adding another context let say a temporal one. For instance *James* can specify in the delegation context that *Alice* can grant another user to delegate *L1*, only during the temporal interval *Working–Hours*. We model this permission using as follows:

δPermission@(Working–Hour) ⊑ PermissionA.Alice ⊓ PermissionD.Delegate ⊓ PermissionL.L1 ⊓ PermissionV.Valid–Level.

This example states that by default *Alice* can delegate the license *L1* that she receives from *James*, but the delegated license will be valid only during the temporal context *Working–Hours*.

e)Revocation: Revocation[13] deals with the process of retrieving the delegated license or role. In the following we redefine the revocation's properties by using the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$.

Grant Dependency: Revocation can be either Grant–Dependent (GD) or Grant–Independent (GID). In the situation of Grant–Dependent revocation, only the Grantor has the right to revoke the delegated license or role. However in the case of Grant–Independent any member of the sponsoring role has the right to revoke the grantee. These two situations are represented with the following permissions by default, while using either the context GD or GID.

δPermission ⊑ PermissionS.Subject ⊓ PermissionR.Revoke ⊓ PermissionLD.License–Delegation ⊓ PermissionD.GD.

δPermission ⊑ PermissionS.Subject ⊓ PermissionR.Revoke ⊓ PermissionLD.License–Delegation ⊓ PermissionID.GID.

The two contexts *GD* and *GID* are redefined as follows:

Hold ⊑ UseL.License–Delegation ⊓ GrantorL.User.

Hold ⊑ UseL.License–Delegation ⊓ GrantorL.GR ⊓ EmpowerGR.Role⊓ EmpowerUser.Role.

The two contexts *GD* and *GID* are applicable in the case of license revocation, to revoke a role a similar approach is used to define the contexts *GDR* and *GIDR*.

Cascading revocation: The delegation chain produced from the process of performing multi–step delegation should be indirectly revoked. This operation is possible by the use of contextual license, where the delegation of right is valid only in case the grantor still has this right. Using T-JClassic$_{\delta\varepsilon}$ description logic, we redefine the View *Cascading–Delegation*, which is a *sub–view* of *License–Delegation View*:

$\delta Permission \sqsubseteq UseL.Cascading–Delegation \sqcap GranteeL.Subject \sqcap GrantorL.Gr \sqcap PrivilegeL.Action \sqcap TargetL.Objet \sqcap ContextL.C$

Inserting an object in this view will create permission with an additional context (*Valid–Delegation context*) which verifies if the grantor still has his right. This context is redefined as:

$Hold \sqsubseteq Is–PermittedU.User \sqcap Is–PermittedO.Object \sqcap Is–PermittedA.Action.$ We notice that the delegated permission is valid only if the delegation chain is maintained.

4. Related work

The delegation model of OrBAC is more complete comparing to those proposed for RBAC [9, 11, 21]. In fact RBAC lacks of expressivity to represent the delegation characteristics. It actually doesn't comprehend the necessary components that enable the representation of those delegation features as permanence, multiple delegation, revocation, etc. OrBAC[12] overcomes this problem by extending the delegation model of RBAC by adding other components such as new types of roles, permissions and relations. This model is also self administrated. We proposed in this paper a redefinition of the delegation rules of OrBAC. We used for this process the temporal nonmonotonic description logic T- JClassic$_{\delta\varepsilon}$. This method permits to augment even more the expressivity of the model. Actually it allows representing the cases where we can have an exception of an exception of a given context while delegating permission.etc. This operation is not possible with OrBAC. It provides also a simpler way to represent the temporal aspects of delegation, by using the temporal components of T-JClassic$_{\delta\varepsilon}$. Thus our approach is even more flexible and easier to deal with. Our delegation model reposes on description logic. Thus from the complexity point of view, our model maintains a polynomial complexity, given the fact that OrBAC is modeled with first order logic, and that description logic is a sub set of it.

5. Conclusion

Providing access control mechanisms in information systems to support dynamic delegation of authority is not a trivial task to model and engineer. In this paper, we presented a delegation model based on temporal context. We used for this purpose the temporal non monotonic description logic T-JClassic$_{\delta\varepsilon}$ to specify delegation policies automatically. The motivation of this work is based on a real world process, where authorizations may change during execution. Delegation policies may change according to specific contexts. This approach permits to augment the expressivity of the new model, while maintaining a polynomial complexity. In fact it provides the mean to represent an exception of an exception on the context of a given delegation, and so on, thanks to the nonmonotonic part of T-JClassic$_{\delta\varepsilon}$, which contains the default and exception elements. This framework was implemented and illustrated using medical information system. The next stage is to adapt our approach to other applications.

References

1. A. Abou El Kalam, P. Balbiani, S. Benferhat, F. Cuppens,Y. Deswarte, R. El-Baida, A. Miège, C. Saurel and G. Trouessin, " Organization-Based Access Control ", 4th International Workshop on Policies for Distributed Systems and Networks (Policy'03), Côme, Italie, 4-6 juin (2003), IEEE Computer Society Press, pp. 120-131.
2. A. Artale and E. Franconi, "A survey of temporal extensions of description logics", In Ann. of Mathematics and Artificial Intelligence, 1-4, pp. 171-210, 2000.

3. A. Artale and E. Franconi, "A Temporal Description Logic for reasoning about actions and plans", In J. of Artificial Intelligence Research, 9, pp. 463-506, 1998.
4. A. Artale and E. Franconi, "Introducing Temporal Description Logics", Invited paper at the sixth International Workshop on Temporal Representation and Reasoning (TIME'99), IEEE Computer Society Press, 1999.
5. A. Artale and E. Franconi, "Temporal Description Logics", In Handbook of Time and Temporal Reasoning in Artificial Intelligence, The MIT Press, 2001.
6. A. Artale and C. Lutz, "A correspondence between temporal description logics", in: Workshop Notes of the Int. Workshop on Description Logics, DL-99, Linköping, Sweden, July 1999, pp. 145–149.
7. F. Baader and B. Hollunder, "Embedding defaults into terminological knowledge representation formalisms", In Principles of knowledge representation and reasoning: 3rd international conference, pp 306– 317, 1992.
8. F. Baader, D.L. McGuiness, D. Nardi and P.F. Schneider, "The Description logic handbook: Theory, Implementation and Applications", Cambridge university press, 2008.
9. E. Barka and R. Sandhu, "A Role-based Delegation Model and Some Extensions", In Proceedings of the 23rd National Information Systems Security Conference (NISSC'00), Baltimore, MD, October 2000.
10. E. Barka and R. Sandhu, "Framework for Agent-Based Role Delegation", In Proceedings of the IEEE International Conference on Communications, ICC '07. 2007.
11. E. Barka and R. Sandhu, "Role-Based Delegation Model/Hierarchical Roles (RBDM1) ", InProceedings of the 20th Annual Computer Security Applications Conference (AC-SAC'04), Tucson, Arizona, December 2004.
12. M. Ben-Ghorbel-Talbi, A. Bouhoula, F. Cuppens and N. Cuppens-Boulahia, "Managing Delegation in Access Control Models", In Proceedings of CoRR, 2010.
13. M. Ben-Ghorbel-Talbi, A. Bouhoula, F. Cuppens and N. Cuppens-Boulahia, "Revocations Schemes for Delegation Licences", In Proceedings of the 10th International Conference on Information and Communications Security (ICICS'08). Springer-Verlag, Birmingham, UK (2008).
14. O. Bettaz, N. Boustia and A. Mokhtari "Extending Nonmonotonic description logic with temporal aspects", In Proc. International Symposium on INnovations in Intelligent SysTems and Applications, Albena, ISBN: 978-1-4799-0659-8, June 19-21, 2013.
15. O. Bettaz, N. Boustia and A. Mokhtari " Temporal DL–OrBAC$_{\delta\varepsilon}$: temporal context in access control model", KES 2014.
16. O. Bettaz, N. Boustia and A. Mokhtari "Dynamic delegation based on temporal and nonmonotonic description logic ", ISPS 2015.
17. N. Boustia and A. Mokhtari, "JClassic δε+ A Description Logic Reasoning Tool: Application to Dynamic Access Control", In Proc. The Second International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, Computation Tools'11, Rome, pp. 25-30, September 25-30, 2011.
18. N. Boustia and A. Mokhtari "Modeling Disjunctive Context in Access Control", In International Journal on Advances in Software, volume 5 no1& 2, 2012.
19. M. Bouzid, C. Combi, M. Fisher, and G. Ligozat, "Temporal Representation and Reasoning", Annals of Mathematics in Artificial Intelligence 46(3):231-234. Springer, March 2006.
20. P. Coupey and C. Fouqueré, "Extending conceptual definitions with default Knowledge", Comput Intell 13(2), 1997.
21. J. Crampton and H. Khambhammettu. "Delegation in Role-based access control", International Journal of Information Security, 7(2):123–136, 2008.
22. F. Cuppens and N. Cuppes-Boulahia, "Modeling contextual security policies", International Journal of Information Security (IJIS), Vol. 7, no. 4, August, 2008.
23. F. Cuppens and A. Miège. "Administration Model for Or-BAC", International Journal of Computer Systems Science and Engineering (CSSE), 19(3), May 2004.
24. F. Cuppens and A. Miège, "Modeling Contexts in the Or-BAC Model", 19th Annual Computer Security Applications Conference (ACSAC'03), 2003.
25. F. Cuppens and A. Miège, "Or-BAC Organization Based Access Control", In Distribution des données à grande Echelle (DRUIDE), Le Croisic, France, 2004.
26. J. MA and B. Knight, "Reified Temporal Logics: An Overview," In journal of Artificial Intelligence Review archive, Volume 15 Issue 3, May 2001.
27. L. Padgham and B. Nebel, "Combining Classification and No Monotonic Inheritance Reasoning: a First Step", In t77th International Symposium on Methodologies for Intelligent Systems, pp. 15- 18, Norway. 1993.
28. L. Padgham and T. Zhang, "A Terminological Logic with Defaults: a Definition and an Application", In 13th International Joint Conference on Artificial Intelligence, pp. 663- 668, Chambery, France. 1993.
29. J. Quantz and V. Royer, "Preference Semantics for Defaults in Terminological Logics", In Principals of knowledge Representation and Reasoning: 3rd International Conference, pp. 294- 305, Cambridge, 1992.
30. R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models", IEEE Computer, 29(2):38–47, 1996.
31. A. Schmiedel, "A Temporal Terminological Logic", In Proc. of AAAI-90, pp. 640–645, Boston, MA (1990)
32. C. Ye, Z. Wu, and Y. Fu. "An Attribute-Based Delegation Model and Its Extension", Journal of Research and Practice in Information Technology, 38(1), 2006.
33. X. Zhang, S. Oh, and R. Sandhu. "Pbdm: A Flexible Delegation Model in RBAC", In Proceedings of the 8th ACM Sym-posium on Access Control Models and Technologies (SAC-MAT'03), Como, Italy, June 2003.