An RBAC Framework for Time Constrained Secure Interoperation in Multi-domain Environments

Smithi Piromruen, James B. D. Joshi
{*smithip*, *jjoshi*}*@mail.sis.pitt.edu*
Department of Information Sciences and Telecommunications, University of Pittsburgh

Abstract

In emerging e-commerce applications, time constrained information sharing between different systems is becoming a common phenomenon. A flexible and efficient mechanism is needed to support short term time-based sharing policies between transient partners. In particular, the interacting domains need to establish a time-based inter-domain access policy without violating the original time-based security policies of the individual systems. In this paper, we address this issue using the Generalized Temporal Role Based Access Control (GTRBAC) framework. The proposed mechanism involves a system processing an inter-domain access requirement specification to extend or restructure its local GTRBAC policy with proper temporal constraints to allow its external partner domain to access its resources. The transformed local GTRBAC policy facilitates the inter-domain accesses while still conforming to the original local policy requirements.

1. Introduction

An important requirement of emerging systems is to be able to share information with other systems [5, 6]. When a system needs to allow previously unknown entities to access its resources, mechanisms should be in place to ensure that the accesses granted are limited to pre-defined sharing requirements. When such inter-domain accesses are allowed between multiple systems with individual systems employing its own security policies, the systems are said to form a multi-domain environment. The accesses allowed may be time-constrained to ensure that resources are made available only when required. For instance, if a company is providing a consulting service to a bank for a specified period of time, then the bank needs to ensure that the required information and other resources are made available to the consulting company within relevant intervals of time within their contractual period. Such multi-domain environments have manifested in various forms of emerging systems. Those particularly becoming prominent include web-services, peer-to-peer systems, grid-based systems, etc. [7, 10]. Multidomain environments can be *loosely* or *tightly* coupled (federated) [7]. In a loosely coupled environment, independent systems dynamically agree to share information for a brief period of time. In a federated multi-domain environment, one system is typically designated as the master and others as local domains. The master is responsible for mediating accesses to individual systems by maintaining a global policy. Such systems may also characterize merged organizational systems, in which the policies are integrated.

In this paper, we emphasize that when requirements-driven interoperation is needed in a loosely coupled environment there should be a sound mechanism to facilitate an external entity to access its resources by mapping external entities to local entities, which may need to be newly created (such as roles) to be created to fulfill requested access. While there has been several work done in the areas of policy integration and trust negotiation/management [1, 4, 12], we believe the approach we present is simple, intuitive and more desirable for a loosely coupled environment. Existing policy integration work focuses on creating a new complex policy by combining the multi-domain policies [2, 11] – such an approach is more suited to tightly coupled environments and is usually very complex. Existing work on trust negotiation do not address policy mapping issues and focus on establishing trust between unknown partners [7]. Hence, the work presented here can be combined with trust negotiation techniques to generate a holistic solution to secure interoperation in loosely coupled environments that characterize many of the currently emerging applications such as P2P and mobile applications.

In this paper, we assume that a pair of security domains trying to interoperate employs Generalized

Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05)
0-7695-2347-1/05 $20.00 © 2005 IEEE

Temporal RBAC policies. The requesting domain specifies its access requirements and the provider domain reconfigures or extends its local policy to satisfy them by creating "exported" roles.

The paper is organized as follows. In Section 2, we present background on the GTRBAC framework particularly relevant to the present secure-interoperation approach. In Section 3, we present our approach and the algorithms for inter-domain policy generation that incorporates temporal access control requirements. We present related work in Section 4 and conclude in Section 5.

2. GTRBAC Overview

2.1. Periodic Expression

The GTRBAC framework uses periodic time to express temporal constraints. Periodic time is represented by as a tuple ⟨P, begin, end⟩, where P is a *periodic expression* denoting an infinite set of periodic time instants, and [begin, end] is a time interval denoting the lower and upper bounds for the instants in P [8]. The periodic time uses the notion of *calendar* defined as a countable set of contiguous intervals. We assume a set of calendars containing the calendars *Hours*, *Days*, *Weeks*, *Months*, and *Years*, where *Hours* is the calendar with the finest granularity. Given two calendars C_1 and C_2, C_1 is said to be a sub-calendar of C_2, written as $C_1 \sqsubseteq C_2$, if each interval of C_2 is covered by a finite number of intervals of C_1. Calendars can be combined to represent more general *periodic expressions* denoting periodic intervals such as the set of *Mondays*.

Definition 2.1 (Periodic time): *A periodic expression* P *is defined as*: $P = \bigcup_{i=1}^{n} O_i.C_i \triangleleft x.C_d$, *where*, C_d, C_1, .., C_n *are calendars,* $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup all$, $C_i \sqsubseteq C_{i-1}$ *for* $i = 2,.., n$, $C_d \sqsubseteq C_n$, *and* $x \in \mathbb{N}$.

Symbol $\triangleleft$ separates the first part of the periodic expression indicating the set of starting points of the intervals, from the specification of the duration of each interval in terms of calendar C_d. For example, $\{all.Years + \{3, 7\}.Months \triangleleft 2.Months\}$ represents the set of intervals having a duration of 2 months with their starting times synchronized with the same instant as the third or the seventh month of every year. A set of time instants corresponding to a periodic expression P is denoted by *Sol*(*I*, *P*).

2.2. The GTRBAC Model

The GTRBAC model introduces the separate notion of role enabling and role activation, and provides

Table 2.1 Constraint Expressions

Categories	Constraints		Expression	Type
Periodicity Constraint	User-role assignment		(*I*, *P*, *pr*:assign$_U$/deassign$_U$ *r* to *u*)	C_{Urp}
	Role enabling		(*I*, *P*, *pr*:enable/disable *r*)	C_{Rp}
	Role-permission assignment		(*I*, *P*, *pr*:assign$_P$/deassign$_P$ *p* to *r*)	C_{PRp}
Duration Constraints	User-role assignment		([(*I*, *P*)| *D*], D_U, *pr*:assign$_U$/deassign$_U$ *r* to *u*)	C_{Urd}
	Role enabling		([(*I*, *P*)| *D*], D_R, *pr*:enable/disable *r*)	C_{Rd}
	Role-permission assignment		([(*I*, *P*)| *D*], D_P, *pr*:assign$_P$/deassign$_P$ *p* to *r*)	C_{PRd}
Duration Constraints on Role Activation	Total active role duration	Per-role	([(*I*, *P*)| *D*], D_{active}, [$D_{default}$], *pr*:active$_{R_total}$ *r*)	C^a_{dr}
		Per-user-role	([(*I*, *P*)| *D*], $D_{uactive}$, *u*, *pr*:active$_{UR_total}$ *r*)	C^a_{dur}
	Max role duration per activation	Per-role	([(*I*, *P*)| *D*], D_{max}, *pr*:active$_{R_max}$ *r*)	C^a_{mr}
		Per-user-role	([(*I*, *P*)| *D*], D_{umax}, *u*, *pr*:active$_{UR_max}$ *r*)	C^a_{mur}
Cardinality Constraint on Role Activation	Total no. of activations	Per-role	([(*I*, *P*)| *D*], N_{active}, [$N_{default}$], *pr*:active$_{R_n}$ *r*)	C^a_{nr}
		Per-user-role	([(*I*, *P*)| *D*], $N_{uactive}$, *u*, *pr*:active$_{UR_n}$ *r*)	C^a_{nur}
	Max. no. of concurrent activations	Per-role	([(*I*, *P*)| *D*], N_{max}, [$N_{default}$], *pr*:active$_{R_con}$ *r*)	C^a_{nmr}
		Per-user-role	([(*I*, *P*)| *D*], N_{umax}, *u*, *pr*:active$_{UR_con}$ *r*)	C^a_{nmur}
Trigger	E_1 ,…, E_n , C_1 ,…, C_k → *pr*:*E* after Δt			C_{tr}
Constraint Enabling	pr:enable/disable *c* where $c \in \{(D, D_x, pr{:}E), (C), (D, C)\}$			C_c
Run-time Requests	*Users' activation request*		(s: (de) activate *r* for *u* after Δt))	C_u
	Administrator's run-time request		(pr:assign$_U$/de-assign$_U$ *r* to *u* after Δt)	C_{admin}
			(pr:enable/disable *r* after Δt)	C_{admin}
			(pr:assign$_P$/de-assign$_P$ *p* to *r* after Δt)	C_{admin}
			(pr:enable/disable *c* after Δt)	C_{admin}

Table 2.2 Status predicates

Predicate	Meaning	Axioms
enabled(*r, t*)	*Role r is enabled at time t*	*For all r* ∈ Roles, *u* ∈ Users, *p* ∈ Permissions, *s*∈ Sessions, *and time instant t* ≥ 0, *the following implications hold*:
u_assigned(*u, r, t*)	*User u is assigned to role r at time t*	
p_assigned(*p, r, t*)	*Permission p is assigned to role r at time t*	*1. assigned*(*p, r, t*)→ *can_be_acquired*(*p, r, t*)
can_activate (*u, r, t*)	*User u can activate role r at time t*	*2. assigned*(*u, r, t*) → *can_activate* (*u, r, t*)
can_acquire (*u, p, t*)	*User u can acquire permission p at time t*	*3. can_activate* (*u, r, t*) ∧ *can_be_acquired*(*p, r, t*) → *can_acquire* (*u, p,t*)
can_be_acquired(*p, r, t*)	*Permission p can be acquired through role r at time t*	
active(*u, r, s, t*)	*Role r is active in user u's session s at time t*	*4. active*(*u, r, s, t*) ∧ *can_be_acquired*(*p, r, t*) → *acquires*(*u, p, s, t*)
acquires(*u, p, s, t*)	*User u' acquires permission p in session s at time t*	

constraints and event expressions associated with both [8]. An enabled role indicates that a valid user can activate it, whereas an activated role indicates that at the least one user has activated the role. The basic GTRBAC model proposed in [8], allows specification of the following set of constraints: (*i*) *Temporal constraints on role enabling/disabling* allow specification of intervals and durations in which a role is enabled; (*ii*) *Temporal constraints on user-role and role-permission assignments* allow specifying intervals and durations in which a user or permission is assigned to a role; (*iii*) *Activation constraints*: These constraints allow specification of restrictions on the activation of a role. These include, for example, specifying the total duration for which a user may activate a role, or the number of concurrent activations of the role at a particular time; (*iv*) *Run-time events* allow an administrator and users to dynamically initiate the various role events, or enable the duration or activation constraints; (*v*) *Constraint enabling* events that enable or disable duration and role activation constraints mentioned earlier; and (*vi*) *Triggers* that allow expressing dependencies among events and conditions

Table 2.1 summarizes the constraint types and expressions of the GTRBAC model. The periodic time expression and duration expressions are used to express temporal constraints. *D* expresses the duration specified for a constraint. In the duration and role activation constraint expressions, D_x and N_x indicate the constrained durations and cardinalities. If the subscript *x* starts with *u* then it is a per-user-role constraint otherwise it is a per-role constraint. For instance, D_{active} indicates how long the specified role can be active, whereas, $D_{uactive}$ indicates the duration for which the specified user may activate the specified role. The following example illustrates the specification of a GTRBAC policy. In the duration and role activation constraint expressions, D_x and N_x indicate the constrained durations and cardinalities. If the subscript *x* starts with *u* then it is a per-user-role constraint otherwise it is a per-role constraint. For more details, we refer the readers to [8].

2.3. Temporal Role Hierarchies

The GTRBAC model includes different hierarchy types and time-based semantics [8]. Various predicate notations are used in defining the semantics of these hierarchies, as shown in Table 2.2. Predicates *enabled*(*r, t*), *assigned*(*u, r, t*) and *assigned*(*p, r, t*) refer to the status of roles, user-role and role-permission assignments at time *t*. Predicate *can_activate*(*u, r, t*) indicates that user *u* can activate role *r* at time *t* implying that user *u* is implicitly or explicitly assigned to role *r*. *active*(*u, r, s, t*) indicates that role *r* is active in user *u*'s session *s* at time *t* whereas, *acquires*(*u, p, s, t*) implies that *u* acquires permission *p* at time *t* in session *s*. The axioms in Table 2.2 capture the key relationships among these predicates and identify precisely the permission-acquisitions and role-activations allowed in GTRBAC [8]. Axiom (1) states that if a permission is assigned to a role, then it *can be acquired* through that role. Axiom (2) states that all users assigned to a role *can activate* that role. Axiom (3) states that if a user *u can activate* a role *r*, then all the permissions that *can be acquired* through *r can be acquired* by *u*. Similarly, axiom (4) states that if there is a user session in which a user *u* has activated a role *r* then *u acquires* all the permissions that *can be acquired* through role *r*. We note that axioms (1) and (2) indicate that permission-acquisition and role-activation semantics is governed by explicit user-role and role-permission assignments.

Semantically, a role hierarchy expands the scope of the permission-acquisition and role-activation semantics beyond the explicit assignments through the hierarchical relations defined among roles. Within the GTRBAC framework, the following three hierarchy types have been identified: *permission-inheritance-only* hierarchy (*I*-hierarchy), *role-activation-only*

hierarchy (*A*-hierarchy) and the combined *inheritance-activation* hierarchy (*IA*-hierarchy) [8]. Each of these constraints of that type. For example C_{URp} also refers to the set containing the periodicity constraints on user-

Table 2.3 Role hierarchies in GTRBAC

Category	Short form	Notation	The condition *c* holds
Unrestricted hierarchies *(No effect of timing constraints on role)*	*I-hierarchy*	$(x \geq_t y)$	$\forall p, (x \geq_t y) \wedge can\_be\_acquired(p, y, t) \rightarrow can\_be\_acquired(p, x, t)$
	A-hierarchy	$(x \succcurlyeq_t y)$	$\forall u, (x \succcurlyeq_t y) \wedge can\_activate(u, x, t) \rightarrow can\_activate(u, y, t)$
	IA-hierarchy	$(x \succeq_t y)$	$(x \succeq_t y) \leftrightarrow (x \geq_t y) \wedge (x \succcurlyeq_t y)$
Consistency Property: *Let* $<f_1><f_2> \in \{\geq_t, \succcurlyeq_t, \succeq_t\}$. *Let x and y be distinct roles such that* $(x<f_1>y)$; *then the condition* $\neg(y <f_2> x)$ *must hold.*			

hierarchies may be of *restricted* or *unrestricted* type. A *restricted* hierarchy may further be categorized as *weakly* or *strongly* restricted. In Table 2.3, the semantics of each hierarchy type is defined by its corresponding condition (*c*). The condition *c* for the *unrestricted I*-hierarchy, $(x \geq_t y)$, implies that if $(x \geq_t y)$ holds, then the permissions that can be acquired through role *x* include all the permissions that *can be acquired through* role *y*. In other words, permissions of the junior roles are inherited by the senior role. Similarly, the condition *c* corresponding to the *unrestricted A*-hierarchy implies that if user *u can activate* role *x*, and $x \succcurlyeq_t y$ is defined, then user *u can* also *activate* role *y* even if he is not explicitly assigned to *y*. The *IA*-hierarchy is the most general form and includes both permission-inheritance and role-activation semantics.

2.4. A-equivalence, Periodic Time Operators

Given a GTRBAC system, we call the set containing all the constraints it's *Temporal Constraint and Activation Base* (*TCAB*). A *TCAB T* can be represented as (C_{URp}, C_{Rp}, C_{PRp}, C_{URd}, C_{Rd}, C_{PRd}, C^a_{dr}, C^a_{dur}, C^a_{mr}, C^a_{mur}, C^a_{nr}, C^a_{nur}, C^a_{nmr}, C^a_{nmur}, C_{tr}, C_c), where each component is a constraint type as depicted in the last column in Table 2.1. Here, we use a constraint type name to also refer to the set containing role assignments. In the discussion below, we use a shorter version, such as $T = (C_{Rp}, C_{URp})$, when only C_{Rp} and C_{URp} are nonempty sets of constraints. The behavior of a GTRBAC system depends on *T*, the set of users Users, the set of roles Roles, the set of permissions Permissions, and the role hierarchy *RH*. Therefore, we can use the tuple (*T*, Users, Roles, Permissions, *RH*) to indicate a GTRBAC configuration. We use the notation $(u \overset{Cf}{\underset{t}{\blacktriangledown}} p)$ to read "*u acquires permission p at time t under configuration Cf*". Next, we define the notion of *a-equivalence* between two GTRBAC configurations.

Definition 2.2 [9](**Activity-equivalence**): *Given a GTRBAC system with two configurations* $Cf_1 = (T_1,$ Users, Roles_1, Permissions, $RH_1)$ and $Cf_2 = (T_2,$ Users, Roles_2, Permissions, $RH_2)$, *the configurations* Cf_1 *and* Cf_2 *are said to be a-equivalent* (*written as* $Cf_1 \approx Cf_2$) *if, for all pairs* (*u*, *p*) *such that u* ∈ Users, *p* ∈ Permissions, *the following condition holds*:

$$(u \overset{Cf_1}{\underset{t}{\blacktriangledown}} p) \leftrightarrow (u \overset{Cf_2}{\underset{t}{\blacktriangledown}} p).$$

Furthermore, if $Cf_1 \approx Cf_x$ *and* $Cf_x \approx Cf_2$, *then* $Cf_1 \approx Cf_2$ (*transitivity*).

The *a-equivalence* between two configurations of a GTRBAC system indicates that by replacing

Table 2.4 Periodic time operators

Relation between $PE_1 = (I_1, P_1)$ *and* $PE_2 = (I_2, P_2)$	If the following condition(s) is (are) satisfied
PE_1 contained *in* PE_2 ($PE_1 \subset PE_2$)	$Sol(I_1, P_1) \subset Sol(I_2, P_2)$
PE_1 *and* PE_2 *are* equivalent ($PE_1 = PE_2$)	$Sol(I_1, P_1) = Sol(I_2, P_2)$
PE_1 *and* PE_2 overlap ($PE_1 \otimes PE_2$)	• $Sol(I_1, P_1) \cap Sol(I_2, P_2) \neq \emptyset$, *and* $\exists\, t, t \in (Sol(I_1, P_1) - Sol(I_2, P_2))$, *and* • $\exists\, t, t \in (Sol(I_2, P_2) - Sol(I_1, P_1))$
PE_1 *and* PE_2 *are* disjoint ($PE_1 @ PE_2$),	$\forall t, (t \in Sol(I_1, P_1) \wedge t \in Sol(I_2, P_2)) \rightarrow t \in (ESol(I_1, P_1) \cap ESol(I_2, P_2))$ *where* $ESol(I, P)$ *is the set of end points of intervals of* (I, P)

configuration Cf_1 by Cf_2, we do not change the accesses that are allowed for each individual user.

2.5. Minimal Disjoint Set

When multiple periodic time expressions may be applied on a GTRBAC entity, it is useful to determine relations between them. The notions of *containment, equivalence, overlapping* and *disjunction* between a pair of periodic time expressions have been introduced in [9], which is summarized in definitions in Table 2.4. Note that the fourth part of the definition implies that if only endpoints of the intervals of two periodic expressions are common, then they are considered disjoint.

Ideally, we want to compute a disjoint set of periodic expressions that is minimal so that these can be associated with temporally distinct roles. The next definition expresses the notion of *minimal disjoint set* (MDS) over a set of periodic expressions.

Definition 2.3 [9] (Minimal Disjoint Set): *Let* $PE = \{PE_1, PE_2, \ldots, PE_n\}$ *be a set of arbitrary periodic expressions*. *The* minimal disjoint set (MDS) over *PE is the least set of* disjoint *periodic expressions,* MDS_{PE}, *defined as*:

$MDS_{PE} = \min_m\{PE'_i \mid 1 \le i \le m\}$, *such that the following hold,*

1. *for all* $1 \le i, j \le m$, $1 \le, i \ne j$, $PE'_i \copyright PE'_j$.
2. $Sol(PE'_1) \cup Sol(PE'_2) \cup \ldots \cup Sol(PE'_m) = Sol(PE_1) \cup Sol(PE_2) \cup \ldots \cup Sol(PE_n)$,
3. *for all* $1 \le i \le m$, $1 \le j \le n$, $PE'_i \subset PE_j$ or $PE'_i \copyright PE_j$

In the definition, the first and second condition indicate that the MDS contains a disjoint set of periodic expressions containing all time instants that are exactly contained in all the original set of periodic expressions PE_is. The third condition ensures that each PE'_i contains time instants that entirely belong to some PE_j. Associated with an MDS, *minimal subset* (MS) *of a periodic expression over a* MDS is defined as follows.

Definition 2.4 [9] (Minimal subset (MS) for a periodic expression over a MDS): *Let* $MDS_{PE} = \min_m\{PE'_i \mid 1 \le i \le m\}$ *be a minimal disjoint set over* $PE = \{PE_1, PE_2, \ldots, PE_n\}$; *The* minimal subset (MS) for a periodic expression $PE_j \in PE$ over the MDS_{PE} *is the set* $MS_{PEx}(MDS_{PE}) = \{PE'_{\pi 1}, PE'_{\pi 2}, \ldots, PE'_{\pi k}\} \subseteq MDS_{PE}$, $1 \le k \le m$ *such that,*

- $\min_k\{\pi k \mid 1 \le i \le k, \pi i \in \{1, 2, \ldots, m\}\}$, and
- *for each* $t \in Sol(PE_x)$, *there is exactly one* $y \in \{\pi 1, \pi 2, \ldots, \pi k\}$ *such that* $(t \in Sol(PE'_y))$

We see that MS of a periodic expression PE_x of *PE* is the minimal subset of MDS_{PE} that collectively contains all the time instants of PE_x. Before presenting an example for MDS and MS, we first show some formal properties related to the computation of MDS and MS. We write ${}^iMDS_{PE}$ to mean MDS of the first *i* periodic expressions of *PE*, i.e., $\{PE_1, PE_2, \ldots, PE_i\}$

Property 1 (Bounds for size of MDS) [9**]**: *Given a set of periodic expressions* $PE = \{PE_1, PE_2, \ldots, PE_n\}$, *if* $MDS_{PE} = \{PE'_1, PE'_2, \ldots, PE'_m\}$ *and* $s_n = |MDS_{PE}|$ *then* $1 \le s_n \le (2^n - 1)$.

Property 2 (Bounds for size of MS) [9**]**: *Given a set of periodic expressions* $PE = \{PE_1, PE_2, \ldots, PE_n\}$ *and* $MDS_{PE} = \{PE'_1, PE'_2, \ldots, PE'_m\}$ *if* $p_n = |MS_{PE1}| + |MS_{PE2}| + \ldots + |MS_{PEn}|$, *then* $n \le p_n \le n2^{n-1}$.

Example 2.1: To simplify notation, we consider the *Daytime* of the days listed in a periodic expression. For example, if $PE = \{Sun\}$, we mean the interval (9am, 9pm) or daytime of a Sunday. Let $PE_A = \{Sun, Mon, Tue, Wed, Thu, Fri\}$, $PE_B = \{Sun, Tue\}$, $PE_C = \{Sun, Tue, Thu, Fri\}$, $PE_D = \{Sun, Mon, Tue, Wed, Sat\}$, $PE_E = \{Thu, Fri\}$. The following steps illustrate the computation of $MDS_{\{PEA, PEB, PEC, PED, PEE\}}$.

1. $MDS_{\{PEA, PEB\}} = \{\{Sun, Tue\}, \{Mon, Wed, Thu, Fri\}\}$
2. $MDS_{\{PEA, PEB, PEC\}} = \{\{Sun, Tues\}, \{Thu, Fri\}, \{Mon, Wed\}\}$
3. $MDS_{\{PEA, PEB, PEC, PED\}} = \{\{Sun, Tues\}, \{Thu, Fri\}, \{Mon, Wed\}, \{Sat\}\}$
4. $MDS_{\{PEA, PEB, PEC, PED, PEE\}} = \{\{Sun, Tues\}, \{Thu, Fri\}, \{Mon, Wed\}, \{Sat\}\}$

We get, $PE'_1 = \{Sun, Tue\}$, $PE'_2 = \{Thu, Fri\}$, $PE'_3 = \{Mon, Wed\}$, $PE'_4 = \{Sat\}$

Hence, we see that,

1. $MS_{PEA}(MDS_{\{PEA, PEB, PEC, PED, PEE\}}) = \{PE'_1, PE'_2, PE'_3\}$.
2. $MS_{PEB}(MDS_{\{PEA, PEB, PEC, PED, PEE\}}) = \{PE'_1\}$
3. $MS_{PEC}(MDS_{\{PEA, PEB, PEC, PED, PEE\}}) = \{PE'_1, PE'_2\}$.
4. $MS_{PED}(MDS_{\{PEA, PEB, PEC, PED, PEE\}}) = \{PE'_1, PE'_3, PE'_4\}$.
5. $MS_{PEE}(MDS_{\{PEA, PEB, PEC, PED, PEE\}}) = \{PE'_2\}$.

3. Time based Interoperation Framework

In this paper, we focus mainly on the algorithms to extend a local GTRBAC policy to facilitate inter-domain accesses. We assume that the negotiation phases have been successfully completed and the access request has been mapped to a set of local permissions. That is, the external domain needs to be allowed to acquire the mapped local permission set.

Figure 3.1 illustrates the proposed interoperation framework. Assuming two domains interoperate, each domain first sends the access requirements to the other. Due to space limitation we do not address the trust negotiation issues that may be employed at this time. Once the requirements have been received, the requests are fulfilled by identifying or creating roles with the requested permissions. The external roles are mapped to exported roles as A-hierarchy relation – this semantically means, the external entity has to activate the specified exported roles in the other domain. At this time, the provider domain can establish activation

conditions to capture this mapping. The exported roles are themselves made *I*-seniors of other local roles that satisfy the requested accesses to ensure that the external entities do not activate other local roles. By using this *A* and *I* hierarchy structure, we prevent the

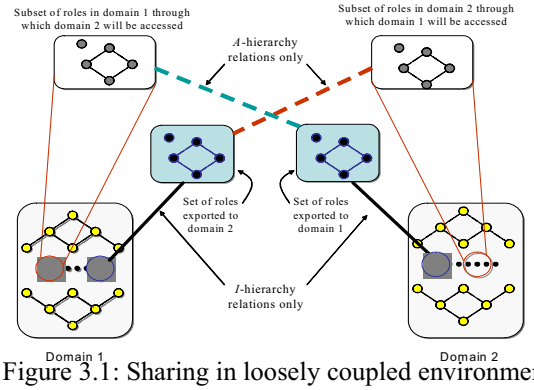

Figure 3.1: Sharing in loosely coupled environment

transitivity of the activation semantics that is usually the underlying problem in inter-domain accesses [5].

We use the conventions shown in Figure 3.2 for our discussion. A hexagon represents a user, a square represents a permission, a circle represents a role and a double circle represents a role through which some or all of the permissions requested by the requesting domain can be acquired.

Author names and affiliations are to be centered beneath the title and printed in Times 12-point, non-boldface type. Multiple authors may be shown in a two- or three-column format, with their affiliations italicized and centered below their respective names. Include e-mail addresses if possible. Author information should be followed by two 12-point blank lines.

3.1. Non-temporal Inter-domain Access

In general, for a requested permission set P_{req}, the following situations may arise in a local domain:

1. there is a set of roles, possibly hierarchically related, through which P_{req} can be exactly acquired.
2. there is a set of roles, possibly hierarchically related, for which P_{req} is a subset of permissions that can be acquired. That is, the set of roles include some other permissions as well.
3. there is a set of roles, possibly hierarchically related, through which only a sub-set of P_{req} can be acquired, as in cases 1 and 2.

In the first case, we need to simply identify the role set required to satisfy P_r and allow them to be assigned to the designated external entities. In the second case, we need to do some additional transformation, which may involve creating new roles, to get a role set that exactly provides P_{req} to the external entities. In the third case, we can split P_{req} into two subsets P_{req1} and P_{req2} such that P_{req1} relates to the first case and P_{req2} refers to the permission set that are not available through any existing role. We can then create a new role for P_{req2} and add that role to the set that satisfies

Function	Definition
$P_{as}(r)$	$\{p \mid p\_assigned(p, r, t)\}$
$P_{at}(r)$	$\{p \mid can\_be\_acquired(p, r, t)\}$
P_{req}	Set of permission requested by an external domain

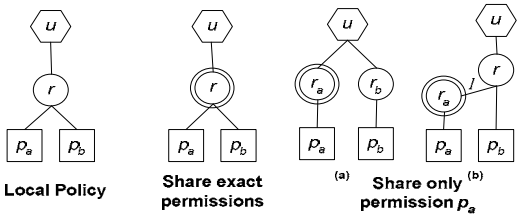

Figure 3.2: Sharing partial permissions

P_{req1} to generate the role set that satisfy P_{req}. In all the cases, any transformation need to ensure that the original permission acquisition semantics is not lost; *i.e.*, each local user still has the same set of accesses allowed in the transformed policy configuration.

Figure 3.3(a) illustrates a simple example where a user *u* is assigned to role *r* to which permissions p_a and p_b are assigned. Assume that $P_{req} = \{p_a, p_b\}$. This conforms to the first case discussed earlier. It is obvious that the request can be satisfied if the external entity is mapped to role *r*, indicated by a double circle. If $P_{req} = \{p_a\}$, it conforms to the second case. Here, role *r* cannot be assigned to the external entity as that would allow the external entity to acquire an extra permission p_b. In such a case, following transformations may be applied to satisfy the access requirements

1. *Split the original role*: The role *r* may be split as shown in Figure 3.3(c). New role r_a now exactly has P_{req}, and the other role now has the remaining permission p_b. Role r_a can now satisfy the access requirements. However, we note that the user who was originally assigned to the split role *r* now needs to be assigned to both r_a and r_b in order to maintain his original access capabilities.
2. *Augment local hierarchy*: Instead of replacing *r* with two new roles, we can simply create role r_a as an *I*-junior of role *r* (i.e., $r \geq_I r_a$) as shown in Figure 3.2(a). Here r_a can now satisfy the access request.

Note that the second solution is more efficient than the first one for several reasons. Firstly, the number of permission assignments (to r_a and r_b in Figure 3.2(c)) and de-assignments (from role *r* in Figure 3.2(c)) involved are less. Secondly, if there are many users assigned to the original role *r*, then the first solution

involves many more user-role assignments (to the new roles) and de-assignments (from the original role).

Figure 3.3 shows slightly more complex case, where multiple roles that satisfy the access request may be in the same hierarchy. It shows a GTRBAC policy consisting of two hierarchically related roles, r_1 and r_2, i.e., r_1 <f> r_2, <f> $\in \{I, A, IA\}$. Assume user u_1 is assigned to role r_1. Permission p_{1a} and p_{1b} are assigned to r_1 and p_{2a} and p_{2b} are assigned to r_2.

In Figure 3.3(a), $P_{req} = \{p_{1a}, p_{1b}, p_{2a}, p_{2b}\}$. We note that $P_{req} = P_{at}(r_1)$ when <f> = $\{I, IA\}$ – hence r_1 satisfies the request. However, if <f> = A then both r_1 and r_2 are needed to satisfy the request because $P_{at}(r_1) = P_{as}(r_1) = \{p_{1a}, p_{1b}\}$ and $P_{at}(r_2) = P_{as}(r_2) = \{p_{2a}, p_{2b}\}$. Next assume that, $P_{req} = \{p_{1a}, p_{2a}, p_{2b}\}$, as shown in

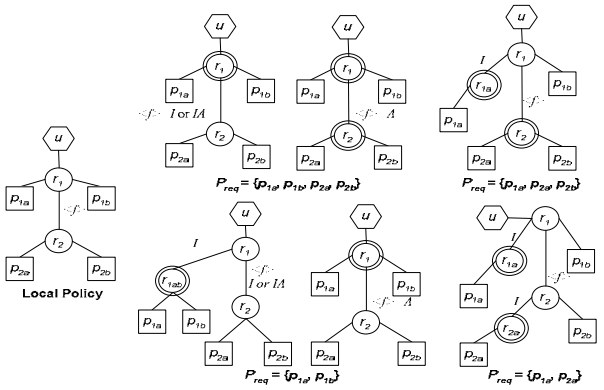

Figure 3.3 Interoperation through hierarchy

Figure 3.3(b). Note that the request cannot be satisfied exactly from the existing GTRBAC configuration. However, using an approach similar to Figure 3.2(d) we can satisfy the request. Here, we can create a new role r_{1a} which becomes an I-junior of r_1 and reassign p_{1a} to it. Now P_{req} can be exactly satisfied through r_{1a} and r_2. Note that <f> can be either I or IA. <f> can also be A as there is no relation between r_{1a} and r_2 in the transformed hierarchy. The same technique can be used in other cases where $P_{req}=\{p_{1a}, p_{1b}\}$ and $P_{req}=\{p_{1a}, p_{2a}\}$.

Next, we present an algorithm *FindRoleSet* that finds the set of roles that satisfies a requested permission set P_{req}. The algorithm does necessary transformation to generate the exactly matching role set for cases 1 and 2. The algorithm assumes that the roles of the hierarchy are visited in a breadth first search, assuming that there is single senior-most role. In general, it is not necessary that there is one single senior-most role. In such cases, we can easily create a senior-most role with the original senior-most roles acting as the juniors or simply assume an initial order among the senior-most role to start the algorithm. The algorithm collects those roles that collectively satisfy the maximum subset of the requested permission set. Once the roles are collected, we can now create a new role that is an I-senior of all of them. This role can now be an exported role. If there is one single role that satisfies the entire requested permission set and it has only I-juniors then, that role can itself be an exported role. These steps make sure that the external entities

Algorithm *FindRoleSet*(P_{req})
Input P_{req} // *permission set requested to be shared*
Output R_{req} // *a set of role which need to acquire* P_{req}
Initialize a set $R_{req} = \emptyset$
R is set of local roles // *Assume roles sorted using BFS*
H is set of role hierarchy relationships
PR is set of (p, r) which permission p is assigned to role r
$R' = R$ // R' *is the set of role of the original configuration*
FOR each role $r \in R'$ **DO**
  // *All uthorized permissions are needed*
  IF $(P_{at}(r) \subseteq P_{req}) \wedge (P_{at}(r) \neq \emptyset)$ **THEN**
    $P_{req} = P_{req} - P_{at}(r)$
    $R_{req} = R_{req} \cup \{r\}$
  // *only some authorized permissions are needed*
  ELSE IF $P_{at}(r) \cap P_{req} \neq \emptyset$ **THEN**
    // *all assigned permissions are needed*
    IF $(P_{as}(r) \subseteq P_{req}) \wedge (P_{as}(r) \neq \emptyset)$ **THEN**
      $P_{req} = P_{req} - P_{as}(r)$
      IF r has junior role with only A-Hierarchy OR no junior role **THEN**
        $R_{export} = R_{export} \cup \{r\}$
        $P_{req} = P_{req} - P_{as}(r)$
      ELSE // *r has junior role with I or IA Hierarchy*
        $R = R \cup \{r_{req}\}$ // *Create new role* r_{req}
        $H = H \cup r \geq_I r_{req}$ // *Create* r_{req} *Ijunior to r*
        FOR each $p \in P_{as}(r)$ **DO**
          $PR = PR \cup \{(p, r_{req})\} - \{(p, r)\}$
        END FOR
        $R_{req} = R_{req} \cup \{r_{req}\}$
      END IF
    // *only some assigned permissions are needed*
    ELSE IF $P_{as}(r) \cap P_{req} \neq \emptyset$ **THEN**
      $R = R \cup \{r_{req}\}$
      $H = H \cup (r \geq_I r_{req})$
      FOR each $p \in P_{as}(r) \cap P_{req}$ **DO**
        $PR = PR \cup \{(p, r_{req})\} - \{(p, r)\}$
      END FOR
      $R_{req} = R_{req} \cup \{r_{req}\}$
      $P_{req} = P_{req} - P_{as}(r)$
    END IF // *no assigned permissions are needed*
  END IF // *case of indirectly assigned permission will be checked in the next role r*
END FOR
// *Refers to Case 3 – some permissions are not available through existing roles*
IF $P_{req} \neq \emptyset$ **THEN**
  $R = R \cup \{r_{new}\}$
  Assign P_{req} to r_{new}
  $R_{req} = R_{req} \cup \{ r_{new} \}$
RETURN R_{req}

Figure 3.4: Algorithm *FindRoleSet()*

are do not have authorization to activate other (sub) roles because of the *I* or *IA* hierarchies that may exist between the exported roles and the internal roles. The modified GTRBAC configuration is essentially a-equivalent to the original configuration. This is because the transformation involves splitting or roles, reassignments of users and permissions, or restructuring of the hierarchy to ensure that the local users are still authorized to the same permissions that they were authorized for before the new policy configuration is created. Due to space limitation, we omit the formal proof.

3.2. Time-based Inter-domain Access

In real world applications, we also encounter temporal requirements with respect to the when and how long the inter-domain accesses need to be allowed. For instance, an external auditor may need to be given access to sensitive financial information only during certain time of the day. Such requirements may be needed to ensure proper availability of resources, supporting personnel to provide assistance (e.g., collaborating internal and external auditors), etc., when needed. In subscription based systems, time factor related to access may be cost related. For instance, a

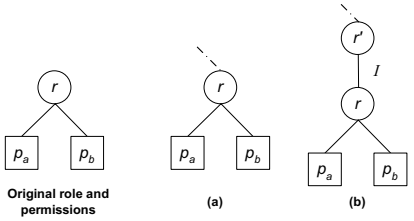

Figure 3.5 An example where *tc* matches *t*

subscriber may choose to access stock information only between 7PM- 9PM to ensure maximum use of information he gets at a lower subscription rate for only two hours per week. Furthermore, although the domain that requests permission may not have time constraints, it is possible that the provider domain will have. As done in the earlier example, an exported role that is made *I*-senior of the set of roles that satisfy the request is created. We follow the similar semantics in the case when the inter-domain accesses need to also satisfy the timing constraints. We assume that all the local roles have temporal constraints on its role enabling. If the external entity does not have its temporal requirements, the access is bounded by the temporal constraints of the local roles that satisfy the requested permission. In that case, we can simply use the algorithm presented earlier. However, there may be specific temporal requirements identified for the external entity when it accesses its local resources though cost-based negotiation.

Figure 3.5 shows a simple case, where role *r* matches exactly with the temporal as well as requested permission requirements of the external domain (the local role's temporal constraint is not shown). In such a case the external entity is simply mapped to role *r* (indicated by the dotted line) as shown in Figure 3.5(a). However, as discussed earlier, in a general case, we prefer to create a role *r*' that is exported because if role *r* has *A* or *IA* juniors then the activation semantics is propagated and the external entity may be able to activate other roles in the hierarchy.

In general, the time-based availability of a set of permissions can be captured by using role enabling time constraint [8]. If the temporal requirement from the external domain has an exactly matching enabling time constraint of a local role, then we can simply map the external role to that role without changing the original configuration, provided that requested permission set is also exactly satisfied by the local role. However, creating a separate role to be exported role is

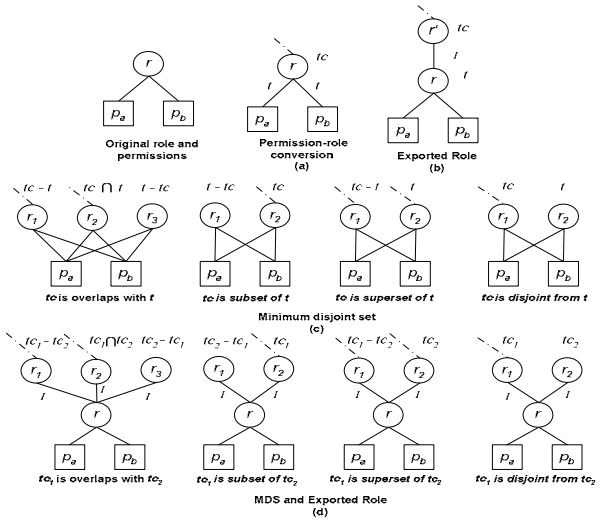

Figure 3.6: The four techniques to handle the time-based inter-domain access

more preferable as discussed in section 3.1. If there is no local role that can exactly match the temporal and permission requirements then the existing policy has to be augmented to capture those requirements to determine roles that can be exported. Four techniques that can be used to modify the local configuration based on the requested temporal constraint are explored: (a) converting temporal enabling time constraint to temporal constraints on permission-role assignment [9] (b) creating a special *exported role* (c)

creating roles with temporal constraints corresponding to the MDS of the external domain's requirement and that of the provider domain [9] and (d) combining MDS technique with a new *exported role*. Figure 3.6 shows the examples of each technique. The temporal constraint *tc* represents the requirement of the requesting domain and *t* represents role enabling time constraint on a local role that matches the requested permission set. The role with a connected dotted line is mapped with the role from the external domain. In the first technique, the temporal constraint *t* on role enabling time is converted to temporal constraints on permission-role assignment constraints – each assignment uses the same temporal constraints expression *t*. The role enabling time constraint of the local role is then set to *tc*. It is easy to see that this technique is applicable only when $t \subset tc$. This is because, if $t \subset tc$ does not hold then there will be some instants in *t* that are not in *tc*, in which the users assigned to the local role may not be able to acquire the permissions assigned to the local role as it will be disabled at that time. The second technique is to create an exported role *r* with enabling time constraint *tc*. This technique is suitable if there are multiple temporal constraints (*e.g.,* presence of both periodicity and duration time constraints). Furthermore, the local policy will not have to be modified. The third technique is based on MDS computation and is shown in Figure 3.6(c). Here the local role is temporally split into roles corresponding to the MDS of the *tc* and *t*. In this case permission-role assignment has to be restructured and can become very complex. Also this technique is only applicable for periodicity time constraint [9]. The fourth technique combines the *MDS* computation and the exported role techniques. Exported roles are created with enabling time constraints corresponding to the temporal constraints of the *MDS* of *tc* and *t* and an *I*-hierarchy is created from these roles to the local role with *t*. This technique will reduce both of the number of roles to be created and the number of permission-role re-assignments. The local role and its assignments need not be modified. A single exported role can be created as *I*-senior of all the temporal exported roles created to facilitate mapping to a single role. This approach is appropriate for more complex scenarios where multiple local roles or temporal constraints are involved. Note that in Figure 3.6 we only illustrate a simple case of one *tc* and one *t*. We propose *FindRoleSetWithTime* algorithm, shown in Figure 3.7 that use the *exported role* technique for duration time constraint and non-temporal requirements. For periodicity time constraints, it uses the fourth technique. We assume that the requirements of an external domain is represented as $DOM\_REQ_i = \{(d_i, \{(P_{ij}, RT_{ij})\}) \mid 1 \le j \le n\}$ where d_i represents a requesting external domain, P_{ij} represents the requested permission set and $RT_{ij} = \{(r, tc)\}$, where *r* is a role in domain d_i through which a user needs to acquire P_{ij} in time instants in *tc*. RT_{ij} indicates that the same permissions may need to be acquired by users in the d_i at different times through different roles. Given the requirement, the algorithm creates an inter-domain mapping to provide the requested permissions

```
Algorithm: FindRoleSetWithTime ()
Input: DOM_REQ // contains the requirements of each domain
Output: POLICY_REL // contains the inter-domain policy
Initialize each set of POLICY_REL, PE_REQ and D_REQ = ∅
FOR each DOM_REQ_i of DOM_REQ DO
  FOR each (P_ij, RT_ij) of DOM_REQ_i DO
    FOR each (r, tc) of RT_ij DO
      IF tc is a periodicity time constraint THEN
        // Create list of requirement with periodic constraint
        PE = PE ∪ {tc}
        PE_REQ = PE_REQ ∪{(r, tc)}
      ELSE IF tc is a duration time constraint THEN
        // Create requirement list with duration constraints
        D_REQ = D_REQ ∪{(r, tc)}
      ELSE
        // Create list of requirement no constraint
        D_REQ = D_REQ ∪{(r, ∞ )}
      END
    END FOR
    // identify local roles for acquiring P_ij
    Local_R = identifyRoles(P_ij)
    Initialize set EXPORTED_R = ∅
    MDS_PE = computeMDS(PE)
    FOR each (r_i, PE_i) of PE_REQ DO
    // r_i is a role from the external domain which needs P_ij
    // during tc = PE_i
    MS_PEi = computeMS(PE_i, MDS(PE))
      FOR each PE_i' of MS_PEi DO
      IF no exported_r ∈ EXPORTED_R which has enabling
        time constraint = PE_i' THEN
        R = R ∪ {exported_r}
        Set exported_r enabling time constraint = PE_i'
      END IF
      FOR each r in Local_R DO
        H = H ∪ exported_r ≥_I r
      END FOR
      POLICY_REL = POLICY_REL ∪ (exported_r, r_i, d_i)
    END FOR
  END FOR
  // For requirement with duration or non-temporal constraint
  FOR each (r_i, D_i) of REQ DO
    R = R ∪ {exported_r}
    IF D_i ≠ ∞ THEN
      Set exported_r enabling time constraint = D_i
    END IF
    POLICY_REL = POLICY_REL ∪ (exported_r, r_i, d_i)
  END FOR
END FOR
RETURN POLICY_REL
```

Figure 3.7: Algorithm *FindRoleSetWithTime* ()

according to the specified time constraints. Here, the algorithm considers all requesting domains DOM_REQ_i, $1 \leq i \leq m$. Using the notion of *a-equivalence* between two configurations of a GTRBAC system, we can easily show that the modified GTRBAC policy is a-equivalent to the original. Due to space limitation, we omit the formal proof.

4. Related Work

Several research efforts have been devoted to the topic of policy composition in multi-domain environment [2, 3, 5]. The problem of secure interoperation has been addressed in literature in the context of multi-level security (MLS) model. [2, 3, 5]. Dawson et. al. [3] have discussed a mediator based framework for establishing secure interoperation among heterogeneous systems with lattice based access control policies. MLS based approach is static in nature and is very restrictive in nature. RBAC based secure interoperation has being recently pursued as a practical approach to solve the multi-domain problem [11]. Shafiq *et al.* [11] allows policy integration between multiple RBAC policies using an Integer programming approach. The main purpose is to maximize inter-domain information exchange for enterprise-wide system and collaboration environment it has a conflict resolution and constraint relaxing using integer programming to get a solution based on some optimal criteria RBAC policy. However, such an approach is suitable more for tightly coupled environments. Furthermore, time-based secure interoperation has not been addressed by earlier models. Several work such as [1] address trust negotiation and trust management issues that can complement the proposed framework for generating a holistic solution for multi-domain environments.

5. Conclusions

In this paper, we have presented our preliminary work on temporal RBAC based policy mapping between two loosely coupled interacting domains for sharing information and resources. The algorithms presented transforms local policies in such a way that the modified policies still respects the original authorizations for the local users while separating roles that can now be provided to the external entities to allow them access the requested resources and information. The work is being extended in several directions. Firstly, tools and techniques are needed to administer the overall evolving policies. As policies change the mappings between two domains may change and hence restructuring it may be needed to maintain inter-domain accesses. The proposed framework needs to be integrated with trust negotiation mechanisms to facilitate the overall system implementation. The presented work considers only two domains, and needs to be extended to generate scalable techniques for any number of domains.

References

[1] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System Version 2." Internet RFC 2704, September 1999.

[2] P.A. Bonatti, M. L. Sapino, V.S. Subrahmanian, "Merging Heterogeneous Security Orderings," *ESORICS 1996*, pp. 183-197.

[3] S. Dawson, S. Qian, and P. Samarati, "Providing Security and Interoperation of Heterogeneous Systems," International Journal of Distributed and Parallel Databases. 8(1), pp 119-145.

[4] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands "Models for Coalition-based Access Control," *Seventh ACM Symposium on Access Control Models and Technologies*, , June 2002, pp. 97 – 106.

[5] L. Gong and X. Qian, "Computational Issues in Secure Interoperation", *IEEE Transaction on Software and Engineering*, Vol. 22, No. 1, January 1996.

[6] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford, "Digital Government Security Infrastructure Design Challenges," *IEEE Computer*, Vol. 34, No. 2, February 2001, pp 66-72.

[7] J. B. D. Joshi, R. Bhatti, E. Bertino, A. Ghafoor, "An Access Control Language for Multidomain Environments," *IEEE Internet Computing,* Nov-Dec 2004, pp 40-50.

[8] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor. Generalized Temporal Role Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering*, 17 (1), Jan, 2005, pp 4-23.

[9] J. B. D. Joshi, E. Bertino, A. Ghafoor, "Analysis of Expressiveness and Design Issues for a Temporal Role Based Access Control Model*," IEEE Transactions on Dependable and Secure Computing* (*Accepted*).

[10] L. Pearlman, V. Welch, Ian Foster, Carl Kesselman, S. Tuecke, "A Community Authorization Service for Group Collaboration," *2002 IEEE Workshop on Policies for Distributed Systems and Networks*.

[11] B. Shafiq, J. B. D. Joshi, E. Bertino, A. Ghafoor, "Secure Interoperation in a Multi-Domain Environment Employing RBAC Policies, *IEEE Transactions on Knowledge & Data Engg.* (*Accepted*).

[12] E. Bertino et. al. X -TNL: An XML-based Language for Trust Negotiations. *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*. 2004.