C2RBAC: An Extended Capability-Role-Based Access Control with Context Awareness for Dynamic Environments

Mitsuhiro Mabuchi[1] and Koji Hasebe[2]
[1]*Toyota Motor Corporation, Japan*
[2]*Department of Computer Science, University of Tsukuba, Japan*
mitsuhiro_mabuchi@mail.toyota.co.jp, hasebe@cs.tsukuba.ac.jp

Keywords: Access Control, Context-aware, Delegation, Capability, RBAC

Abstract: Various working styles, such as remote work, have become more common instead of working in one office. Moreover, to accelerate the development of new technologies, collaborations among multiple companies are increasing. Thus, most development projects are operating in dynamic environments, for example, dynamically changing teams, working from anywhere and at any time. To ensure security in such dynamic environments while maintaining efficiency, flexible and scalable access control is necessary. We previously proposed capability-role-based access control (CRBAC) that allows users to create capabilities for delegating authority across various domains without an administrator's operation. However, in dynamic environments, a finer control is required based on where and when the authority is delegated or executed. In this paper, we propose an access control model called context-aware CRBAC (C2RBAC). This model is an extension of CRBAC obtained by introducing a mechanism of context-based restrictions on various operations regarding the delegation of authority by capabilities, such as time, place, and device. In this paper, we present a formal definition of C2RBAC and demonstrate its effectiveness using an example of collaborative development.

1 INTRODUCTION

Various working styles, such as remote work, have become more common instead of working in one office. In particular, because remote work is spreading rapidly due to the COVID-19 pandemic, the use of resources (servers and databases) from unexpected places, such as public spaces, is increasing. Moreover, to accelerate the development of new technologies, collaborations among multiple companies are increasing, for example, the collaboration of a company with data analysis skills and a company having big data.

Thus, most development projects are operated in dynamic environments because the team is assumed to be dynamically organized across companies, and the working style also changes dynamically, such as places, times, and devices depending on situations. Flexible and scalable access control is paramount to ensure security in such dynamic environments while maintaining efficiency. However, configuring access control by administrators may consume time and is inflexible if the team changes frequently. Manually assigning authority to users in other domains makes managing the system even more challenging. Furthermore, confidential data such as personal information must be protected from unexpected use by access control.

We previously proposed capability-role-based access control (CRBAC) (Hasebe et al., 2010) that is an extension of role-based access control (RBAC) (Sandhu et al., 1996) with capability-based access control (CBAC) (Snyder, 1981), (Levy, 1984). CRBAC allows users to create capabilities from their capabilities for delegating authority across domains without an administrator's operation. However, a finer control is required in dynamic environments depending on where and when the authority is delegated or executed. Because CRBAC cannot set any limitation by contexts associated with environments, users having access to a customer information database can obtain sensitive information from that database.

To realize such capability management, we propose an access control model called context-aware CRBAC (C2RBAC). This model is an extended CRBAC model with context-awareness. C2RBAC can impose context-based restrictions on various operations related to the delegation of authority by capabilities, such as time, place, and device. The context-based restrictions can prevent potential issues in dynamic environments, for example, an unexpected propagation of capability and the use of capability from an unexpected place or time. The contexts comprise surrounding environment information of users, such as place, time, and device, whereas the constraints are attributes of capabilities or roles, such as the number of uses and expiration date. The contexts can be evaluated with rules attached to capabilities whenever users use a capability. In this paper, we formally define the C2RBAC model as the basic C2RBAC0 model and describe model families C2RBAC1, C2RBAC2, and C2RBAC3. Furthermore, we demonstrate the effectiveness of C2RBAC using an example of collaborative development.

The rest of this paper is organized as follows. Section 2 presents related work and comparison results. Section 3 introduces the C2RBAC model. Section 4 demonstrates the application of C2RBAC considering a plausible case of collaborative development as a dynamic environment. Finally, Section 5 concludes this study and presents future work.

2 RELATED WORK

RBAC (Sandhu et al., 1996) is widely used as a model for hierarchical and efficient access control in an organization or a system. It assigns a preset role that is a set of permissions to appropriate users by authenticating their identities. The permission-based delegation model (Zhang et al., 2003) is an extension of RBAC to realize role delegation among users in a single domain. These enable fine-grained access control for a well-organized company or system.

Numerous context-aware applications (Baldauf et al., 2007) have been proposed for adapting dynamic environments, such as the Internet of Things. In particular, some applications extended RBAC to define new access control models to realize fine-grained and well-organized access control in dynamic environments. Context-aware RBAC (Kulkarni and Tripathi, 2008) supports dynamic environments, such as place and time, by personalizing user access privileges based on the context information collected by the context management layer. Contex-role based access control (Park et al., 2006) based on RBAC was proposed by introducing context-role to adapt to dynamic environments. Moreover, all environment roles were created and managed by administrators; therefore, this model could not yield flexible and scalable access control in expected dynamic environments. Schefer-Wenzl et al. (Schefer-Wenzl and Strembeck, 2013) proposed a model that supports dynamic environments by extending process-related RBAC (Strembeck and Mendling, 2011) with context and setting restrictions by context when executing a certain task. Covington et al. (Covington et al., 2001) proposed a model that combines newly defined environment roles assigned to users based on context information with existing subject roles assigned to users based on user identity.

Another extension of RBAC, Rule-based RBAC (RB-RBAC) (Al-Kahtani and Sandhu, 2002), was proposed based on attribute information to address access control in dynamic environments. As a result of comparing a user's attribute with role allocation rules, PB-RBAC assigns a specific role to that user. In addition, when the attribute changes because of environmental changes, RB-RBAC performs the comparison of new attributes and rules to assign an appropriate role. In attribute-based access control (ABAC) (Jin et al., 2012), the attribute is defined as a function that takes an entity as input and outputs a value within a specific range. ABAC realizes fine-grained access control by adding attributes to all users, subjects, and objects. Our previous model, CRBAC (Hasebe et al., 2010), is an extension of RBAC with CBAC that enables flexible cross-domain access authority delegation. In CRBAC, users can create new capabilities from capabilities or roles assigned to them for delegating authority without any operation by the administrator.

We also proposed workflow-aware CRBAC (W-CRBAC) (Hasebe and Mabuchi, 2010) for applying the workflow system to CRBAC. W-CRBAC supports the workflow system to use a task transferable along with permissions by mapping a task to both a capability and a role.

We evaluate C2RBAC and other access control models described above in terms of the following eight items.

(1) Adding new members (or assigning authority to a user who has not been registered to a system) without an administrator's operation

(2) Delegating authority to users in the same domain without an administrator's operation

(3) Revoking delegated authority from users in the same domain without an administrator's operation

(4) Delegating authority to users in different domains without an administrator's operation

(5) Revoking delegated authority from users in different domains without an administrator's operation

(6) Preventing unexpected authority propagation

(7) Restricting authority management by a contextual information

(8) Traceability of authority propagation

	RBAC (Sandhu et al., 1996) PBDM (Zhang et al., 2003)	Context-aware RBACs (Kulkarni and Tripathi, 2008), (Park et al., 2006), (Schefer-Wenzl and Strembeck, 2013), (Covington et al., 2001)	RB-RBAC (Al-Kahtani and Sandhu, 2002) ABAC (Jin et al., 2012)	CRBAC (Hasebe et al., 2010) W-CRBAC (Hasebe and Mabuchi, 2010)	C2RBAC our new model
(1)				✓	✓
(2)	✓	✓	✓	✓	✓
(3)	✓	✓	✓	✓	✓
(4)				✓	✓
(5)				✓	✓
(6)	✓	✓	✓	partly	✓
(7)		✓	✓		✓
(8)	✓	✓	✓	✓	✓

Table 1: A comparison table of access control models which adapted to dynamic environments.

CRBAC, W-CRBAC, and C2RBAC can realize (1), (4), and (5) by exploiting the characteristics of CBAC. All models described above can realize (2), (3), and (8). However, CRBAC and W-CRBAC prevent capability propagation by constraints, such as the number of delegations and expiration date; they cannot perfectly realize (6) if there is a malicious user because of capability flexibility. Therefore, C2RBAC uses context restrictions to prevent them. Regarding (7), context-aware models, RB-RBAC, ABAC, and C2RBAC, can support dynamic environments. C2RBAC supports all items to achieve secure, flexible, and scalable access control for dynamic environments.

3 C2RBAC

This section gives a formal definition of our proposed Context-Aware Capability Role Based Access Control (C2RBAC). C2RBAC is a family consisting of four models named C2RBAC0 to C2RBAC3. C2RBAC0 is a base model. C2RBAC1 and 2 are extensions of C2RBAC0 by introducing role hierarchy and restrictions, respectively. Here, context is formally represented as a set of variables ranging over states in the system, such as the current time, IP address of a user's terminal. Context can restrict on each operation of the delegation of authority, i.e. creation, transfer, execution of authority, and revocation of capablity.

Below, Section 3.1 defines the base model. Section 3.2 gives the definition of delegation of authority with capability in this model as a state transition of the system. Finally, Section 3.3 describes the extension of the base model. The application of the model defined above to an example will be shown in the next section.

3.1 C2RBAC0

The C2RBAC0 is the base model of the C2RBAC family. The formal definition is the following. (See also Figure 1 for a graphical presentation of the family of C2RBAC models. In this figure, the definition of the expression "rh_i", which refers to the role hierarchy, is described in the definition of C2RBAC2 in Section 3.3.)

Definition 1 (C2RBAC0)**.** The C2RBAC0 model has the following basic components. (Here, for a set X, the notation $\mathcal{P}(X)$ denotes the powerset of X.)

- Sub, Dom: the sets of *subjects* and *domains*, respectively.
- $Role_i$, Per_i, Ses_i, Cap_i: the sets of *roles*, *permissions*, *sessions*, and *capabilities* in the i-th domain for each $i \in Dom$, respectively. Here, Per_i may contain a special kind of permission, called *creation of capabilities*, denoted by *create*.
- $S = S_1 \times \cdots \times S_k$: the set of all possible states of the system. Here, a state is a snapshot of the system described by the values of the variables of the system at a given moment. In the following, a state of the entire system si called a global state, while a state of a part of the system (especially in a domain) is called a local state.
- $Cxt = Cxt_1 \times \cdots \times Cxt_n$: the set of all possible global states (or called *contexts*), where Cxt_i for each $i = 1, \ldots, n$ is the set of all possible local states in the i-th domain. That is, $Cxt_i = Cxt_i^1 \times \cdots \times Cxt_i^h$ where $Cxt_i^j = S'_j$ for all $j = 1, \ldots, h$ with $j' = 1, \ldots, k$. Here, various kinds of states can be regarded as contexts. For example, the time (from 0 to 23) and the user's location (L_1, L_2, and L_3) when a session is activated in Domain 1 are considered as contexts, the set Cxt_1 is $Cxt_1^1 \times Cxt_1^2$ with $Cxt_1^1 = \{0, 1, \ldots, 23\}$ and $Cxt_1^2 =$

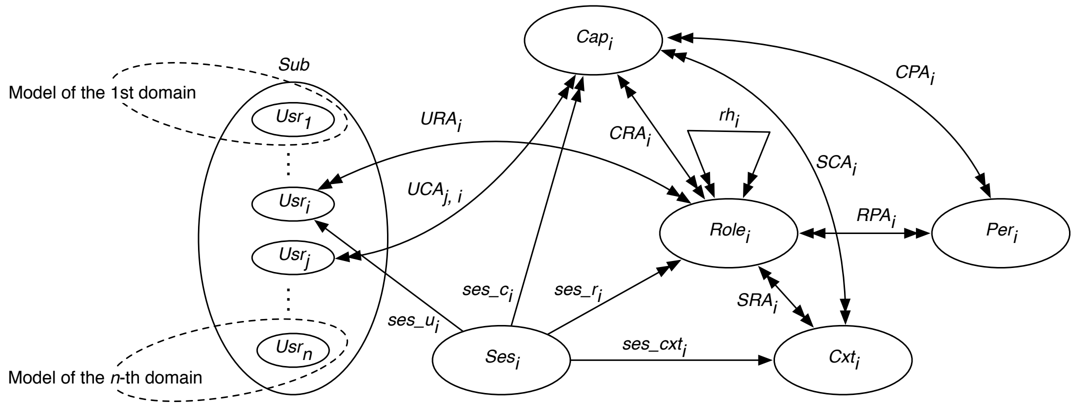

Figure 1: The family of C2RBAC models.

$\{L_1, L_2, L_3\}$. As a notational convention, to indicate a global or local state of the system (i.e., an element of set *Cxt* or Cxt_i), we often use σ or σ_i.

In addition to the above components, the following functions and relations are defined for each $i \in Dom$:

- $usr : Dom \to \mathcal{P}(Sub)$, a function that determines the set of users in the i-th domain for each $i \in Dom$. We also use the notation Usr_i to denote $usr(i)$. Hereafter, we assume that $Sub = \bigcup_{i \in Dom} Usr_i$ with $Usr_i \cap Usr_j = \emptyset$ for any $i \neq j$.
- $ses\_u_i : Ses_i \to Usr_i$, a function mapping each session in the i-th domain to the user who establishes it.
- $ses\_r_i : Ses_i \to \mathcal{P}(Role_i)$, a function mapping each session in the i-th domain to a set of roles that is activated by this session.
- $ses\_c_i : Ses_i \to \mathcal{P}(Cap_i)$, a function mapping each session in the i-th domain to a set of capabilities.
- $ses\_cxt_i : Ses_i \to Cxt_i$, a function mapping each session in the i-th domain to the local state when the session is established.
- $URA_i \subseteq Usr_i \times Role_i$, a many-to-many user-to-role assignment relation.
- $RPA_i \subseteq Role_i \times Per_i$, a many-to-many role-to-permission assignment relation.
- $UCA_{j,i} : Usr_j \to \mathcal{P}(Cap_i)$, a function mapping each user in the j-th domain to a set of capabilities.
- $CRA_i \subseteq Cap_i \times Role_i$, a many-to-many capability-to-role assignment relation.
- $CPA_i \subseteq Cap_i \times Per_i$, a many-to-many capability-to-permission assignment relation.
- $SRA_i \subseteq Cxt_i \times Role_i$, a many-to-many context-to-role assignment relation.
- $SCA_i \subseteq Cxt_i \times Cap_i$, a many-to-many context-to-capability assignment relation.

These components satisfy the following conditions:

(C0-1) $ses\_r_i(s) \subseteq \{r \mid \langle ses\_u_i(s), r\rangle \in URA_i\}$, means any role activated by a session is one that is assigned to the user who establishes the session.

(C0-2) $ses\_r_i(s) \subseteq \{r \mid \langle ses\_cxt_i(s), r\rangle \in SRA_i\}$.

(C0-3) Session s has the permissions $\bigcup_{r \in ses\_r_i(s)}\{p | \langle r, p\rangle \in RPA_i\}$.

(C0-4) $ses\_c_i(s) \subseteq \{c \mid \langle ses\_u_i(s), c\rangle \in UCA_i\}$, means any capability activated by a session is owned by the user who establishes the session.

(C0-5) $ses\_c_i(s) \subseteq \{c \mid \langle ses\_cxt_i, c\rangle \in SCA_i\}$.

(C0-6) For any $r \in Role_i$ and $c \in Cap_i$, if $\langle c, r\rangle \in CRA_i$ then session s has the permissions that are determined by (C0-2) above, otherwise (i.e., $\langle c, r\rangle \notin CRA_i$) then s has the permissions $\bigcup_{c \in ses\_c_i(s)}\{p \mid \langle c, p\rangle \in CPA_i\}$.

The C2RBAC0 is a pure extension of the CRBAC model, which was introduced by our previous work based on the RBAC96 model. Here, the components $Contexts_i$, ses_cxt_i, SRA_i, and SCA_i are the extended parts from CRBAC, each of which plays the following roles: $Context_i$ denotes the set of all possible local states in the i-th domain; function ses_cxt_i determines the context when each session is established; relations SRA_i and SCA_i determine the set of roles and capabilities that can be activated in each state, respectively.

3.2 Delegation of authority

Similar to the CRBAC proposed in our previous study, various types of delegations are provided in C2RBAC. As described in (Hasebe et al., 2010), the delegation using capability transfer is modeled as the sequential composition of the following three basic operations.

(Step 1) *Creation*. A user creates a new capability.

(Step 2) *Assignment*. A user assigns authority to the capability.

(Step 3) *Transfer*. A user sends the capability created in the previous steps to another user.

For each of the first two steps, we can consider the following two cases. For Step 1, there are two cases in which the authority to create a new capability is based on the (1) role or (2) capability of the user who created the capability. For Step 2, there are two cases in which the authority to be delegated assigned to the created capability is either (3) a role or (4) a capability. According these cases, we classify delegations into four types D1–D4 as shown in Table 2.

Assign \ Create from	(1) Role	(2) Capability
(3) Role	D1	D3
(4) Capability	D2	D4

Table 2: Types of Delegations.

We define the delegation of authority as the state transitions in the model.

Definition 2 (Basic operations for delegation)**.** The basic operations for delegation (i.e., *creation*, *assignment*, and *transfer*) are defined by the following rules of state transitions for a model. Here, CH_i (called *creation history*) is a set of quadruple of the form $\langle u,\sigma,x,c\rangle$ with $u \in Usr_i$, $\sigma \in S$, $x \in Role_i \cup Cap_i$, and $c \in Cap_i$. Intuitively, an element $\langle u,\sigma,x,c\rangle$ in CH_i means that user u creates a capability c in state σ by executing his/her authority assigned to the role or capability x. Thus, CH_i indicates the history of creations of capabilities in the i-th domain.

Creation rule: Let σ be a state such that $\langle u,\sigma,x,c\rangle \in CH_i$.

(1) If $r \in Role_i$ with $\langle u,r\rangle \in URA_i$, $\langle r, create\rangle \in RPA_i$, and $\langle \sigma,r\rangle \in SRA_i$, then this rule can be applied to σ and defines new state σ' by replacing Cap_i and CH_i in σ with Cap'_i and CH'_i, respectively, such that:

- $Cap'_i := Cap_i \cup \{c'\}$ where $c' \notin Cap_i$;
- $CH'_i := CH_i \cup \{\langle u,\sigma,r,c'\rangle\}$.

(2) If $\langle u,c\rangle \in UCA_i$, $\langle c, create\rangle \in CPA_i$, and $\langle \sigma,c\rangle \in SCA_i$, then this rule can be applied to σ and defines new state σ' by replacing Cap_i and CH_i in σ with Cap'_i and CH'_i, respectively, such that:

- $Cap'_i := Cap_i \cup \{c'\}$ where $c' \notin Cap_i$;
- $CH'_i := CT_i \cup \{\langle u,\sigma,c,c'\rangle\}$.

The state transitions of these types are denoted by $\sigma \overset{cre(u,r,c')}{\rightsquigarrow} \sigma'$ (for Case (1)) and $\sigma \overset{cre(u,c,c')}{\rightsquigarrow} \sigma'$ (for Case (2)), respectively, and we can say "capability c' is created by user u from his role r (from his capability c, resp.)"

Assignment rule: Let σ be a state, $u \in Usr_i$, and c' ($\in Cap_i$) is created by u.

(3) If $R' \supseteq R \supseteq Role_i$ in σ such that $\langle u,r\rangle \in URA_i$ for any $r \in R$, then this rule can be applied to σ and defines new state σ' by replacing CRA_i in σ with $CRA'_i := CRA_i \cup \{\langle c',r\rangle \mid r \in R'\}$ and $SRA'_i := SRA_i \cup \{\langle \sigma,r\rangle \mid \sigma \in S, r \in R'\}$.

(4) If $P \supseteq Per_i$ in S, then this rule can be applied to σ and defines new state σ' by replacing CPA_i in S with $CPA'_i := CPA_i \cup \{\langle c',p\rangle \mid p \in P\}$ and $SCA'_i := SCA_i \cup \{\langle \sigma,c'\rangle \mid \sigma \in S\}$.

State transitions of these types are denoted by $\sigma \overset{asg(R',c')}{\rightsquigarrow} \sigma'$ (for Case (3)) and $\sigma \overset{asg(P',c')}{\rightsquigarrow} \sigma'$ (for Case (4)), respectively, and we can say "roles R' (permissions P', resp.) is assigned to capability c".

Transfer rule: Let σ be a state in which $u \in Usr_i$, $u' \in Usr_j$, $c' \in Cap_i$, and c' is created by u (where i and j may be the same domain number). Then this rule can be applied to σ and defines new state σ' by replacing $UCA_{i,j}$ in σ with $UCA'_{i,j} := UCA_{i,j} \cup \{\langle u',c'\rangle\}$. State transitions of this type are denoted by $\sigma \overset{trans(u,u',c')}{\rightsquigarrow} \sigma'$ and we can say "capability c' is transferred from user u to the other user u'". In particular, such transfer is called a *cross-domain transfer* if $i \neq j$.

By combining these basic operations, we define the delegations of types (D1) to (D4).

Definition 3 (Delegation)**.** The delegations of types (D1) to (D4) are sequential compositions of basic operations *Creation*, *Assignment*, and *Transfer* as follows (where ";" is used to denote the sequential-composition operator):

(D1) $\sigma \overset{cre(u,c',r)}{\rightsquigarrow};\overset{asg(R',c')}{\rightsquigarrow};\overset{trans(u,u',c')}{\rightsquigarrow} \sigma'$.

(D2) $\sigma \overset{cre(u,c',r)}{\rightsquigarrow};\overset{asg(P',c')}{\rightsquigarrow};\overset{trans(u,u',c')}{\rightsquigarrow} \sigma'$.

(D3) $\sigma \overset{cre(u,c',c)}{\rightsquigarrow};\overset{asg(R',c')}{\rightsquigarrow};\overset{trans(u,u',c')}{\rightsquigarrow} \sigma'$.

(D4) $\sigma \overset{cre(u,c',c)}{\rightsquigarrow};\overset{asg(P',c')}{\rightsquigarrow};\overset{trans(u,u',c')}{\rightsquigarrow} \sigma'$.

For readability, we also use the following notations to denote delegations of these types:

(D1) $\sigma \overset{D1(u,u',c',r,R')}{\rightsquigarrow} \sigma'$.

(D2) $\sigma \overset{D2(u,u',c',r,P')}{\rightsquigarrow} \sigma'$.

(D3) $\sigma \overset{D3(u,u',c',c,R')}{\rightsquigarrow} \sigma'$.

(D4) $\sigma \overset{D4(u,u',c',c,P')}{\rightsquigarrow} \sigma'$.

3.3 Extended models

Based on the C2RBAC0 model, we develop a C2RBAC family similar to RBAC96 model (Sandhu et al., 1996). C2RBAC1 and C2RBAC2 are extensions of C2RBAC0 by introducing role hierarchy and constraints, respectively, that are essentially the same as for RBAC1 and 2. Furthermore, similar to the RBAC96 model, C2RBAC3 is obtained by the combination of C2RBAC1 and C2RBAC2.

Formally, C2RBAC1 is obtained from C2RBAC0 by adding the following component:

- rh_i, a partial order over $Role_i$, called *role hierarchy*. (The infix notation $r' \geq_i r$ is also used to denote role hierarchy and we can say "r' is senior role of r" or "r is junior role of r'" in the same sense as in the RBAC96 model.)

C2RBAC1 is as C2RBAC0 (Definition 1) where conditions (**C0-1**), (**C0-2**), (**C0-3**), and (**C0-6**) are respectively replaced with the following conditions: (**C1-1**), (**C1-2**), (**C1-3**), and (**C1-6**).

(C1-1) $ses\_r_i(s) \subseteq \{r \mid \exists r' \geq_i r(\langle ses\_u_i(s), r' \rangle \in URA_i)\}$, means any role activated by a session is one that is assigned to the user who establishes the session.

(C1-2) $ses\_r_i(s) \subseteq \{r \mid r' \geq_i r(\langle ses\_cxt_i(s), r' \rangle \in SRA_i)\}$.

(C1-3) Session s has the permissions $\bigcup_{r' \in ses\_r_i(s)} \{p \mid \exists r \geq_i r'(\langle r, p \rangle \in RPA_i)\}$.

(C1-6) For any $r, r' \in Role_i$ and $c \in Cap_i$, if $\langle c, r \rangle \in CRA_i$ with $r' \geq_i r$, then session s has the permissions that are determined by (C0-2) above for r', otherwise (i.e., $\langle c, r \rangle \notin CRA_i$) then s has the permissions $\bigcup_{c \in ses\_c_i(s)} \{p \mid \langle c, p \rangle \in CPA_i\}$.

The former two are related to the role hierarchy, that are the same as in the RBAC96, while the latter two guarantee the inheritance of permissions with respect to the role hierarchy.

On the other hand, C2RBAC2 is obtained by adding various constraints. Typical examples are as follows.

- Lifetime
- The number of activations
- The number of creations of new capabilities
- The inheritance of permissions related to role hierarchy
- The number of hops of capability transfer

Here we note that the difference between constraints and contexts: a state of the system is defined as an element in Cxt, while the attributes of capabilities are supposed to be not involved in Cxt.

4 CASE STUDY

We demonstrated the effectiveness of C2RBAC for the eight items described in Section 2 using an example of collaborative development between four companies, Companies A, B, C, and D. The example scenario consists of the following five steps. The graphical presentation is shown in Figure 2, where the solid and dashed arrows represent the flows of the delegation of authority and access to resources, respectively.

Step 1. To start collaborative development with Company B ($Co.B$), the administrator in Company A ($Co.A$) assigns the *developer* role to the Manager and Alice in $Co.A$.

Step 2. Alice creates a capability (called c_1) by executing the authority of the *developer* role and delegates it to Bob, a temporary member in Alice's team. This capability is limited in the execution of authority by the following constraints and contexts.

- Constraints: expiration date, and prohibition of capability creation and delegation.
- Contexts: Restriction of the available devices.

This delegation of authority allows Bob to access the resources required by him to work in terms of the devices authorized by this capability. Moreover, Bob cannot create new capabilities and delegate them to others. To realize the delegation of authority in this step, the base model needs to provide items (1) and (2) presented in Table 1 in Section 2.

Step 3. Alice creates another capability (called c_2) from her *developer* role, and delegates it to Carol in $Co.B$. Here, there is no constraints on c_2 for delegation of authority, because $Co.B$ is taking responsibility to develop and manage the web application and analyze data and outsources some tasks to Company C and D ($Co.C$ and $Co.D$, respectively). On the other hand, c_2 is assigned permissions to access the resources with the following constraints.

- Constraints: Expiration date.

To realize the delegation of authority in this step, the base model needs to provide item (4) in Table 1.

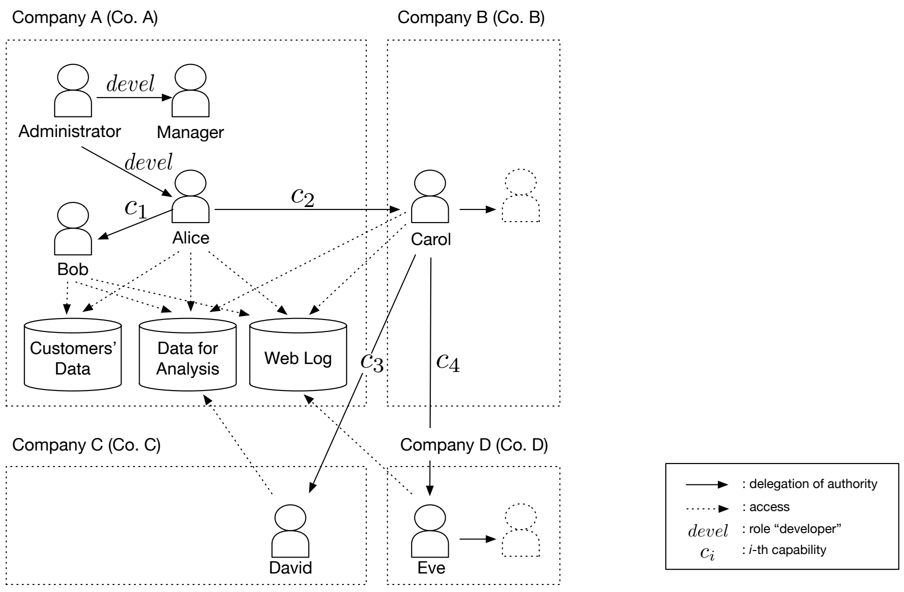

Figure 2: A scenario of collaborative development with multiple companies in dynamic environments.

Step 4. Carol creates a capability (called c_3) from c_2, and delegates it to David, a data analyst in Company C (*Co.C*). Access rights to the storage are assigned to c_3 for data analysis. Here, c_3 inherits constraints of c_2, and additional constraints are imposed, resulting in the following constraints and contexts.

- Constraints: expiration date, and prohibition of capability creation and delegation.
- Contexts: limitation of a source IP address.

To realize the delegation of authority in this step, the base model needs to support item (7) in Table 1.

Step 5. Carol creates another capability (called c_4) from c_2 to access resources related to web application development and delegates it to Eve, an engineer in charge of the web application development in Company D (*Co.D*). The capability c_4 also inherits the constraints of c_2. To continuously operate the web application, Carol allows Eve to create new capabilities and delegate them for a limited number of times as a constraint to only engineers who belong in *Co.D* using contexts. As a result, c_4 is assigned the following constraints and contexts.

- Constraints: expiration date, and limit on the number of creations and delegations.
- Contexts: restriction of available devices, and restriction of delegation range.

Owing to the constraints and contexts, Eve can delegate capabilities only to the members in the same company, and they perform their operations using only the permitted devices. Therefore, the base model needs to realize items (2), (4), (6), and (7) in Table 1.

Capabilities can be revoked by anyone who has an upper-level capability in a hierarchy of capability; for example, Alice can revoke all capabilities such as c_1, c_2, c_3, and c_4 in this case study. If a capability is revoked, all lower capabilities are also revoked simultaneously. For example, if Alice revokes c_3, c_4 is also revoked. Furthermore, anyone who has an upper-level capability can confirm a history of creation and the delegation of all lower-level ones. These features satisfy items (3), (5), and (8) in Table 1. Consequently, we have shown that C2RBAC can support all items in Table 1.

The above scenario can be formally described by the state transitions shown in Section 3.2 as follows.

Step 1. $D_1(Admin, Manager, devel, admin, \{devel\})$ and $D_1(Admin, Alice, devel, admin, \{devel\})$.

Step 2. $D_2(Alice, Bob, c_1, devel, \{devel\})$.

Step 3. $D_2(Alice, Carol, c_2, devel, \{May(Data, access), May(Web, access)\})$.

Step 4. $D_4(Carol, David, c_3, c_2, \{May(Data, access)\})$.

Step 5. $D_4(Carol, Eve, c_4, c_2, \{May(Web, access)\})$.

5 CONCLUSIONS AND FUTURE WORK

We proposed a new access control model, C2RBAC, based on CRBAC with a mechanism of context-aware control. To realize secure, flexible, and scalable access control for dynamic environments, C2RBAC provides context-aware capability management, such as restricting delegation range, while maintaining the flexibility and scalability of CRBAC. Similar to CRBAC, we described model families C2RBAC1, C2RBAC2, and C2RBAC3. In addition, we demonstrated the effectiveness of C2RBAC by comparing it with other models and using an example of collaborative development.

For future work, we will define protocols of context-aware capability management to feasibly implement C2RBAC and develop a formal method for security verification. In particular, we are interested in a method for detecting unintended capability propagation by searching all possible delegation processes using a model checking approach. We are also interested in implementing a prototype of C2RBAC.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers whose comments/suggestions helped improve this manuscript.

REFERENCES

Al-Kahtani, M. A. and Sandhu, R. (2002). A model for attribute-based user-role assignment. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, pages 353–362. IEEE.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

Covington, M. J., Long, W., Srinivasan, S., Dev, A. K., Ahamad, M., and Abowd, G. D. (2001). Securing context-aware applications using environment roles. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20.

Hasebe, K. and Mabuchi, M. (2010). Capability-role-based delegation in workflow systems. In *IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing, EUC 2010, Hong Kong, China, 11-13 December 2010*, pages 711–717. IEEE Computer Society.

Hasebe, K., Mabuchi, M., and Matsushita, A. (2010). Capability-based delegation model in rbac. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, pages 109–118.

Jin, X., Krishnan, R., and Sandhu, R. (2012). A unified attribute-based access control model covering dac, mac and rbac. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 41–55. Springer.

Kulkarni, D. and Tripathi, A. (2008). Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 113–122.

Levy, H. M. (1984). *Capability-Based Computer Systems*. Butterworth-Heinemann, USA.

Park, S.-H., Han, Y.-J., and Chung, T.-M. (2006). Context-role based access control for context-aware application. In *International Conference on High Performance Computing and Communications*, pages 572–580. Springer.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2):38–47.

Schefer-Wenzl, S. and Strembeck, M. (2013). Modelling context-aware rbac models for mobile business processes. *International Journal of Wireless and Mobile Computing 3*, 6(5):448–462.

Snyder, L. (1981). Formal models of capability-based protection systems. *IEEE Trans. Comput.*, 30(3):172–181.

Strembeck, M. and Mendling, J. (2011). Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5):456–483.

Zhang, X., Oh, S., and Sandhu, R. S. (2003). PBDM: a flexible delegation model in RBAC. In Ferrari, E. and Ferraiolo, D. F., editors, *8th ACM Symposium on Access Control Models and Technologies, SACMAT 2003, Villa Gallia, Como, Italy, June 2-3, 2003, Proceedings*, pages 149–157. ACM.