Criticality Aware Access Control Model for Pervasive Applications *

S. K. S. Gupta, T. Mukherjee and K. Venkatasubramanian
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287
http://impact.asu.edu

Abstract

Access control policies define the rules for accessing system resources. Traditionally, these are designed to take a reactive view for providing access, based on explicit user request. This methodology may not be sufficient in the case of critical events (emergencies) where automatic and timely access to resources may be required to facilitate corrective actions. This paper introduces the novel concept of **criticality** *which measures the level of responsiveness for taking such actions. The paper further incorporates criticality in an access control framework for facilitating the management of critical situations. Specific properties and requirements for such criticality aware access control are identified and a sample model is provided along with its verification.*

1 Introduction

An important aspect of Pervasive Computing is to develop intelligent environments which allow inhabitants to interact seamlessly with a smart, information-rich space [1]. The ability of such spaces to monitor and interact with its internal elements, makes them ideal candidates for hackers and tech-criminals to exploit. Access control is therefore an essential component of smart spaces to prevent unauthorized access to the information available in them.

Under normal circumstances, systems provide services in response to routine events. Access control in such circumstances can follow standard pre-defined set of policies, such as Role-Based Access Control (RBAC) [2]. In case of system emergencies, however, the access control requirements change radically. For example in a smart home environment, the basement door may be usually kept locked to prevent access to the children in the house. However, in case of emergencies, such as tornado warning, the basement should be automatically unlocked to allow unhindered access. Further, to be effective, the unlocking of basement has to be done within a certain duration, after the warning has been received. We refer to such system emergencies (e.g. tornado) as *criticality* and the causal events (e.g. tornado warning) as *critical events*.

This paper addresses access control for pervasive computing systems to aid them in handling critical events. The *idea* is to proactively monitor the system contextual information, to detect the occurrence of critical events, and to automatically provide alternate access privileges, appropriate for facilitating mitigative actions (e.g. unlocking the basement). This is unlike many of the traditional context aware access control policies (e.g. Context-Aware RBAC (CA-RBAC) model [3]), which by nature are reactive, and observe/evaluate the system context only on explicit access control requests from subjects. We contend that such reactive approaches are slower and less flexible compared to our proactive model.

The *contribution* of this paper is to identify the requirements and properties for criticality aware access policies and provide a sample model to realize it. Such criticality awareness enables the system to plan for emergencies and provides flexibility to prepare for appropriate mitigative procedures. We next describe the notion of criticality (Section 2) followed by the sample access control specification (Section 4) along with its verification (Section 5).

2 Criticality

Criticality is a measure of the required level of responsiveness for taking corrective actions to control the effects of critical events. To quantify this attribute, we define an application dependent parameter called *Window-of-Opportunity* (W_o), which is the maximum delay that can possibly be allowed to take corrective actions after the occurrence of a critical event [1]. In the tornado example, this value has been determined to be five minutes after the warning [4]. Similar values exist for other domains as well such

*Supported in part by MediServe Information Systems, Consortium for Embedded Systems and National Science Foundation grant ANI-0196156

[1] W_o can be determined in many ways including empirical analysis and system policies

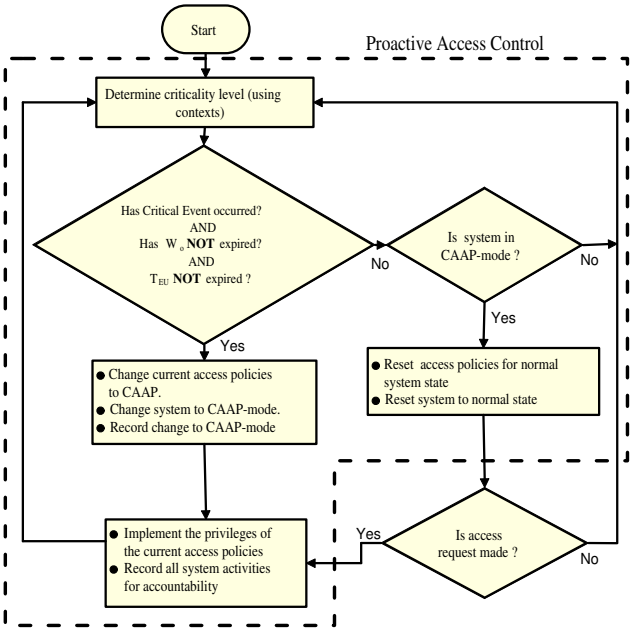

Figure 1. Criticality Handling in CAAC

as the *golden hour* for stroke patients. Standard access control policies, do not provide the necessary flexibility to take such timely actions, against the critical events. In the event of a tornado warning, the aware home, working with standard access control policies, will not unlock the basement without explicit request jeopardizing safety in certain cases. Therefore, it may be necessary for the system to enforce a new set of access policies to handle such criticalities.

We refer to such alternate access policies as Criticality-Aware Access Policies (CAAP) and during their enforcement, the system is said to be in the *CAAP-mode* as opposed to the normal mode (Figure 1 shows the control flow for this transition of the system from its normal mode to the CAAP-mode). In the tornado scenario, the smart home, in the CAAP-mode, unlocks the basement for easy access. However, changing access policies, during critical events, may introduce security concerns. These stem from the fact that the new set of policies may provide higher privileges to access resources. Further, if these higher privileges are provided for an extended duration, they may lead to misuse. Therefore, we contend that the duration of the CAAP-mode should be limited and end at a time, which is the *minimum* of: 1) the end of W_o, 2) the time instant when criticality is controlled (T_{EOC}), and 3) the time instant when all necessary actions to mitigate the criticality, have been taken (T_{EU}) [2].

In summary, managing system criticality has three main components: 1) ability to proactively monitor system context, 2) enforcement of an alternate set of rules for criticality management, and 3) limiting the duration of such alternate enforcements. Additionally we envision that the criticality aware access control should possess the following four properties: **Responsiveness**: The system should immediately respond to any critical event. **Correctness**: A change in policies should only be in response to a critical event. **Liveness**: The duration of policy changes should be finite and last only as long as needed. **Non-Repudiability**: All system activities performed in the CAAP-mode should be monitored for ensuring accountability.

[2]Limiting the CAAP-mode to T_{EU} is required because, once all possible corrective actions have been taken, continual provision of the CAAP is unnecessary, irrespective of the outcome

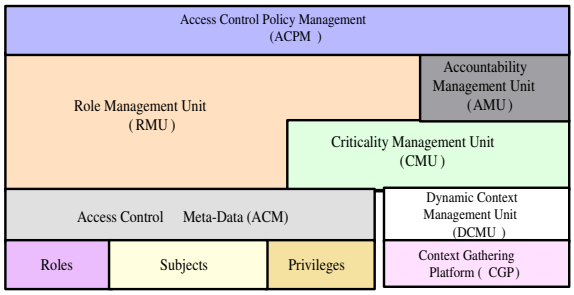

Figure 2. CAAC Model

In this paper, we present a sample mechanism for realizing criticality aware access control policies called Criticality Aware Access Control (CAAC). It has the ability to handle both critical and non-critical situations. The enclosed section in Figure 1 represents the proactive control required for criticality management. The CAAC model continually determines the criticality level of the system, and on observing a critical event, changes the access policies to CAAP and enters the CAAP-mode. If there is no criticality, then the system checks whether it is in the CAAP-mode, in which case, it returns the system to its normal state and enforces the appropriate policies when access is requested.

3 System Model

Criticality aware access extends the traditional context based access by incorporating proactive monitoring for critical events. To demonstrate criticality management, we present a centralized model for CAAC (Figure 2) which is designed to handle only one critical event at a time. Here, the *ACM* provides the meta-data abstracting the dependencies between the subjects, their roles and the corresponding privileges. The *CGP* collects raw context data while *DCMU* processes it and provides higher level contextual information from it. The context is further filtered by the CMU to handle criticality. Based on the information from CMU and ACM, the *RMU* then enforces appropriate access privileges.

Figure 3 shows the architecture for the CMU. The *CI* proactively monitors the system context information and detects the occurrence of criticality. The *CLD* component uses pre-defined window-of-opportunity values, de-

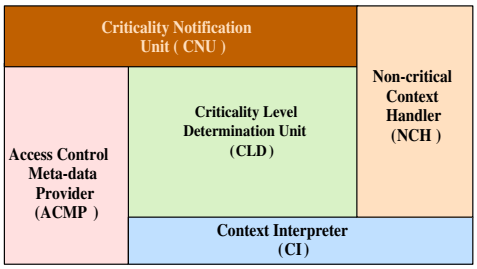

Figure 3. Criticality Management Unit (CMU)

pending upon the type of criticality detected[3]. The *CNU* then notifies other system components about the detected type and level of criticality and moves the system to the CAAP-mode[4]. The policies associated with the CAAP-mode are determined (by CNU) using the meta-data provided by *ACMP* which is an interface for ACM. *NCH*, on the other hand, enforces standard access control policies.

AMU records all events in the system for accountability, while *ACPM* uses the underlying infrastructure to implement the access control and administrative policies.

4 Access Control Specifications

Access control specifications is presented in terms of model and policies. In our model we assume that all the system components are time-synchronized and there is a reliable context gathering platform which provides accurate data and is not vulnerable to attacks by adversaries. We have two types of system policies: Access Control Policies and Administrative Policies. We present our access control model and associated policies below.

4.1 Access Control Model

The principal model builds upon the context aware role based access control model given in [5] and extends it to incorporate criticality. Our system supports two types of roles: *system role* (R_{sys}) and *space role* (R_{space}). The former is assigned to subjects when they become part of a system. For example a person joining a hospital as a surgeon may get a system-role of a *doctor*. The latter is derived from the subject's system role, and the current contextual information and is used for access control.

In our model, access to resources is given based on access control lists (ACLs) maintained for each resource in the system. The ACL is a list of all possible space roles that subjects can be assigned in the system and their respective privileges. To provide access, a subject's system role and context information is mapped on to a space role using an application dependent function (we refer to this mapping function as $currentrole$). The obtained space role is then used to index the resource's ACL to retrieve the subject privileges. During a critical event, the system promotes the space role of certain subjects (as part of the CAAP-mode) to provide them with higher privileges.

4.2 Access Control Policies

Access Control Policies define the rules for controlling the access to different resources within the system. All the decisions take into account the roles, context and criticality for making the access control decisions. In the following specification the symbols S, r_{sys} and r_{space} denote the set of resources in the system, the system role ($r_{sys} \in R_{sys}$), and the space role ($r_{space} \in R_{space}$) respectively.

Access Control Predicate: In this predicate (ACP), when a subject u makes a request to a resource s for a specific method m (thus making $request(u, s, m)$ true), it presents a set of credentials (C). The credentials are of following types: $typeof(u, r_{sys})$ which ensures that u has the system-role r_{sys}, and $exports(s, m)$ which ensures that s provides the access privileges m. Given these credentials, the predicate validates the access control by first computing the subject's space-role (using $currentrole$ function). It then uses the $promoterole$ function to decide if the role needs to be promoted. Role promotion happens only if the request is made in the $CAAP-mode$. The access is provided only if m matches the space-role returned by $promoterole$ in the ACL of s[5]. This predicate extends the access control predicate in [5], to include $promoterole$ function which handles criticality by executing the CAAP.

ACP:
$request(u, s, m) \wedge typeof(u, r_{sys}) \in C \wedge (s \in S) \wedge exports(s, m)$
$\wedge (promoterole(currentrole(r_{sys}, context), u), m) \in ACL_s$

Promote and Demote Role: The function $promoterole$ (Alg 1) is used to promote the space-role of a subject with respect to a resource in case of an access request for controlling a critical event. When invoked, this function checks the level of criticality [6] using the function $Criticality$. If access is requested in a normal state ($Criticality() = 0$), it simply returns r_{space}. If a critical event has occurred, it does the following: 1) computes the window-of-opportunity of criticality using the function $calculatetime$ which takes the level of criticality as input, 2) implements CAAP, by computing the promoted space-role r_{pSpace}, based on the level of criticality using $calculaterole$, and 3) updates a Promoted Role Table (PRT) with the following tuple <*subject's id u, promoted role* r_{pSpace}, *start time of role-promotion* $currentTime()$, *stop time of role-promotion* W_o>. The

[3]The value of Window-of-Opportunity may not be static and may vary with mitigative actions (a stroke patient's condition progressively stabilizes as she is treated thereby increasing the Window- of-Opportunity

[4]It implements the logic given in Figure 1

[5]The assumption here is that the privileges associated with a higher role encompasses all the lower privileges.

[6]Level of Criticality (CL) is an application specific positive number, e.g. if the critical event is a tornado occurrence, then the CL could be its intensity

PRT is used to account for all subjects whose roles have been promoted. The presence of such a table allows for easy auditing and role accountability.

Alg 1:PROMOTE ROLE:
Function Name : $promoterole$, *Return Value* : Promoted Role
Attributes : $u \in$ Set of Subjects, $r_{space} \in R_{space}$
1. if (Criticality() $\neq$ 0)
2. $W_o = calculatetime(Criticality())$
/* Compute new space-role */
3. $r_{pSpace} = calculaterole(Criticality(), u)$
/* Update PRT */
4. $PRT = PRT \cup \{(u, r_{pSpace}, currentTime(), W_o)\}$
5. return r_{pSpace}
6. else
7. return r_{space}
8. end if

We also define another function called $demoterole$ (Alg 2) which demotes the space-role for a promoted subject when the critical event has been controlled. If the current time returned by $currentTime()$ is within W_o it updates the <stop time of the role-promotion> element of the corresponding PRT entry to the current time.

Alg 2: DEMOTE ROLE:
Function Name: $demoterole$, *Return Value* : Demoted Role
Attributes: $u \in$ Set of Subjects
1. for (each resource) do
/* Demote role */
2. $currentrole(r_{sys}, context)$, where $typeof(u, r_{sys})$
/* Update PRT */
3. $\forall i \in PRT$, if $currentTime() < W_o$ such that $W_o \in i$
4. $PRT = PRT - \{(u, r_{space}, t_{start}, W_o)\}$
5. $PRT = PRT \cup \{(u, r_{space}, t_{start}, currentTime())\}$
6. end for

Notification: Criticality management is mainly handled using the $notifyCritical$ function (Alg 3). This function primarily implements the logical flow of criticality handling presented in Figure 1. It has following five aspects: 1) To continuously monitor the system for critical events (using $Criticality$ which returns 0 when there is no criticality) and start the CAAP when a critical event occurs. 2) Identifying the appropriate subjects who can deal with a critical event using the function $findUser$. 3) Notifying the subjects identified to handle criticality. The $notify$ function is used for this purpose. It takes as input the result of $findUser$, the current criticality level and the associated W_o. If the notification has already been sent to the subject for the same criticality level, it returns false otherwise it returns true. 4) Promoting the roles of the subjects (using $promoterole$) to provide them necessary privileges, on resources, for handling the critical event. 5) Demoting the subject roles when the effects of critical events are handled or go beyond control, using the $demoterole$ function and returning the system to normal state. Specifications for the functions $Criticality$, $findUser$, $notify$, $calculaterole$, $calculatetime$, and $start\_timer$ are application dependent and therefore are abstracted out.

Alg 3: NOTIFICATION:
Function Name : $notifyCritical$
1. while (TRUE)
2. while (Criticality() = 0)
3. if ($state = CAAP\text{-}mode$)
/* Revert to the normal state, when criticality is over (T_{EOC})*/
4. $state = normal$
5. $demoterole(u)$
6. end if
7. end while
8. if ($state = CAAP\text{-}mode$)
9. if (timer W_o expired)
/* Revert to the normal state, when window of opportunity expires */
10. $state = normal$
11. $currentrole(r_{sys}, context)$, $\forall$ resource
12. else
13 if (all actions taken)
/* Revert to the normal state, when all actions have been taken (T_{EU}) */
14. $demoterole(u)$
15. $state = normal$
16. end if
17. end if
18. end if
/* Criticality is observed, enter the CAAP-mode */
19. $state = CAAP\text{-}mode$
20. $CL = Criticality()$
21. $W_o = calculatetime(CL)$
22. if $\neg(notify(findUser(), CL, W_o))$
23. $start\_timer(W_o)$
24. $promoterole(currentrole(r_{sys}, context), u)$, $\forall$ resource
25. end if
26.end while

4.3 Administrative Policies

For adding/removing subjects, roles and managing the smart spaces within the system, we use the policies given in [5]. We however incorporate an additional policy for maintaining accountability (Alg 4) within the system. The function basically, returns the details of the PRT and a Current-Role-Table (CRT table is used to store the current space role of a user) for a particular user. The presence of role accountability allows the administrator ($SysAdm$) to find out which roles were promoted, when they were promoted and for what resources.

Alg 4: ACCOUNTABILITY:
Function Name : $roleaccountability$
Attributes : $u \in$ Set of Subjects, $u_a \in$ Set of Subject
1. if ($typeof(u_a, SysAdm) \in C$)
/* Access PRT */
2. $\forall a \in PRT, if\, u \in a$ obtain the tuple $(u, r_{space}, t_{start}, t_{end})$
/* Access CRT */
3. $\forall a \in CRT, if\, u \in a$ obtain the tuple (u, r_{sys}, r_{space})
4. end if

4.4 Example

We now illustrate how the CNU specification can be enforced in the aware-home example given in Section 1. When a tornado warning is issued, the aware-home's CI detects this and calculates the level of criticality using the function $Criticality$. This information is then used, by the CLD, to calculate the associated W_o (5 minutes) using

$calculatetime$. The CNU then uses the criticality level and associated W_o to automate the transition to the CAAP-mode (unlock basement) using the functions $notifyCritical$ and $promoterole$. Once the criticality level has receded (tornado has passed), the CNU executes $demoterole$ to switch back to the standard access mode (ensure everyone is out of the basement & lock it). In the standard mode, all access requests are evaluated by the NCH using *ACP* predicate.

5 Verification

In this section we present semi-formal proofs for verifying the policies specified above. We assume all access control policies execute correctly, all the administrative entities are trusted and the policies and system log cannot be accessed in an unauthorized manner. The following proofs verify the four properties of criticality awareness.

Theorem 5.1 Correctness: *The system can enter the CAAP-mode if and only if there is a critical event.*

Proof if part: If there is a criticality, function $promoterole$ is called (in line 24 of Alg 3) and line 3 of $promoterole$ (Alg 1) will be executed. $only - if$ part: If the role of a subject is promoted, it means that line 4 of $promoterole$ has been reached earlier and this can happen only in case of a critical event. As $promoterole$ implements CAAP, the result follows. ∎

Theorem 5.2 Liveness: *For a single critical event, a subject's role is promoted for a maximum of W_o time (i.e $max(T_{CAAP}) = W_o$).*

Proof From Theorem 5.1 and the assumption that roles are not promoted in normal system state, it follows that when a subject's role is promoted, the W_o timer has been started for the critical event for which the role has been promoted. The promoted role of a subject is demoted in the following cases: 1) Line 11 of Alg 3: If the W_o expires. Here $T_{CAAP} = W_o$. 2) Line 5 of Alg 3: If there is no criticality, but the system state is in the CAAP-mode. This can only happen if a critical event has been controlled before its W_o ($T_{EOC} < W_o$). Therefore, $T_{CAAP} = T_{EOC}$. 3) Line 14 of Alg 3: If the W_o has not expired, but all actions for critical event handling have been taken i.e. $T_{EU} < W_o$. Then $T_{CAAP} = T_{EU}$. Therefore for a single critical event, the subject's role is promoted for a maximum of W_o. ∎

Theorem 5.3 Responsiveness: *When a critical event occurs - 1) the subject is immediately notified, 2) if required the subject's access privileges are elevated (role promotion), and 3) any role promotion is active until either the criticality is controlled or it cannot be controlled any more.*

Proof The proofs of the claims above are as follows: 1) When there is a criticality, the subjects are notified in line 22 of Alg 3. 2) If the subject being notified already has required privileges, its role is not promoted as the call for the function $calculaterole$ in line 3 of Alg 1 does not return elevated space-role (by definition). Otherwise, the function $calculaterole$ returns an elevated space-role based on the level of criticality, thus promoting the subject's role. 3) From Theorem 5.1 and the assumption that roles are not promoted in normal state, we know that role promotion is done when there is a critical event and from Theorem 5.2 it follows that role is promoted until either the criticality is controlled or W_o expires. ∎

Theorem 5.4 Non-Repudiation: *Malicious use of promoted role after the occurrence of a critical event is non-repudiable and limited to a finite amount of time.*

Proof Line 4 in Alg 1 and line 5 Alg 2 ensure that whenever a role is promoted it is recorded in PRT along with the appropriate start and end times enforcing non-repudiation of any malicious activity by a subject due to role promotion. As we assume that all the access control policies execute correctly, the PRT table is accurately updated. Further, as the PRT is assumed to be secured from any unauthorized access, and the administrator is a trusted entity, line 2 of Alg 4 can be used for ensuring non-repudiation. From Theorem 5.1 and 5.2, it follows that the maximum time the role can be promoted, in the CAAP-mode, is W_o, thereby limiting potential malicious activity to a finite amount of time. ∎

6 Conclusions

In this paper we presented a novel access control framework for smart spaces, called criticality aware access control, which helps mitigate system emergencies incorporating a novel concept of *criticality*. We further presented a sample criticality aware access control model (CAAC) to illustrate the working of our framework and provided semi-formal proofs to verify its properties. Future work includes its formal verification and prototype development.

Acknowledgments

We are grateful to John Quintero, Guofeng Deng, and the anonymous reviewers for their comments which helped improve the paper, and, Bruce and Zach Mortensen of MediServe Information Systems for their encouragement.

References

[1] F. Adelstein, S. K. S. Gupta, G. G. Richard and L. Schwiebert "Fundamentals of Mobile and Pervasive Computing". *McGraw Hill*, 2005

[2] R. Sandhu, E. Coyne, H. Feinstein and C. Youman. "Role Based Access Control Models". *In IEEE Computer*, Feb, 1996.pp 38-47

[3] A. Kumar, N. Karnik and G. Chafle. "Context Sensitivity in Role-based Access Control". *In ACM SIGOPS OS Review 36(3)*, July, 2002

[4] Working Group on Natural Disaster Information Systems "Effective Disaster Warning" November, 2000

[5] G. Sampemane, P. Naldurg and R. Campbell. "Access control for Active Spaces". *In Proc. of ACSAC*, 2002