CAAC - An Adaptive and Proactive Access Control Approach for Emergencies in Smart Infrastructures

KRISHNA K. VENKATASUBRAMANIAN[1], TRIDIB MUKHERJEE[2], and SANDEEP K. S. GUPTA[3]

[1]Worcester Polytechnic Institute, Worcester, Massachusetts

[2]Xerox Research Center India, Bangalore, India

[3]Arizona State University, Tempe, Arizona

The article presents an access control model called *Criticality Aware Access Control* (CAAC) for criticality (emergency) management in smart infrastructures. Criticalities are consequences of events which take a system (in our case, a smart infrastructure) into an unstable state. They require the execution of specific *response actions* in order to bring them under control. The principal aim of CAAC is to grant the right set of access privileges (to facilitate response action execution), at the right time, to the right set of subjects, for the right duration, in order to control the criticalities within the system. In this regard, the CAAC model uses a stochastic model called the *Action Generation Model* to determine the required response actions for the combination of criticalities present within the system. It then facilitates response actions by *adaptively* altering the privileges to specific subjects, in a *proactive* manner, without the need for any explicit access requests. In this article, we formalize the CAAC model and validate it based on two *design goals* - proactivity and adaptiveness. Finally, we present a case study demonstrating CAAC's operation on an oil-rig platform in order to aid in the response to health and fire related criticalities.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Access Control; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Access Control, Proactivity, Adaptivity

Additional Key Words and Phrases: Criticality Aware Access Control, Criticality, Window-of-Opportunity, Pervasive Computing, Smart Infrastructure, Cyber-Physical Systems

1. INTRODUCTION

Recent years have seen the development of smart infrastructures which consist of a large number of heterogeneous, massively distributed computing entities. Such

This work was done while the first and the second authors were with the IMPACT Lab, Arizona State University.

This research is supported in part by National Science Foundation grants CNS-0617671, CT-0831544 and MediServe Information Systems. A preliminary version of this article appeared in the Proceedings of Fourth IEEE International Conference on Pervasive Computing and Communications, Pisa, Italy, March 2006, pages 251-257.

Corresponding author's address: S. K. S. Gupta, IMPACT Lab (http://impact.asu.edu), School of Computing, Informatics, and Decision Systems Engineering Ira A. Fulton School of Engineering, Arizona State University, Tempe, Arizona, 85287; email: sandeep.gupta@asu.edu

cyber-physical systems [Gupta 2008] provide their users with an aware, intelligent, information rich environment for conducting their day-to-day activities. An important application of their monitoring capabilities is *emergency management* [Adelstein et al. 2005] [Venkatasubramanian et al. 2005]. Examples of emergencies include patient needing urgent medical attention, crisis such as building fire, and the computing infrastructure under attack from outside. Smart infrastructures can detect such emergencies and facilitate response by providing real-time information to the planners and relief workers, thereby improving the chances of saving lives and property.

Traditional wisdom dictates that in the event of emergencies, any security system in place should be disabled in order to allow relief workers to fully utilize the capabilities of the system for controlling the emergency [Mehrotra et al. 2004; Denning et al. 2008]. However, given the extent of sensitive information available within smart infrastructures, disabling security in the event of emergencies may potentially leave the system vulnerable to exploitation. Similarly, it is also possible that malicious elements may dupe the smart infrastructure into detecting a false emergency, disable the security system and collect sensitive information. We therefore contend that while utilizing smart infrastructures for emergency management care has to be taken that the privacy of the users is protected. We define *privacy preservation during emergency management* as the temporary provisioning of the necessary information and services for responding to the emergency, to specific subjects, until the emergency is contained. In the following section we present some of the principal concepts regarding emergencies and their management.

1.1 Emergencies or Criticalities: Concepts

Emergencies, also referred to as *criticalities* in this article, can be defined as adverse consequence of specific events called *critical events* on a system. Critical events are those whose occurrence moves a system into an abnormal/unstable state. Criticalities usually require timely *response actions* to be *controlled*, *i.e.*, their adverse effects negated. A criticality, which has not yet been controlled, is called an *active criticality*. In practical terms, controlling a criticality means minimizing the possibility of loss of lives, services, and property in the event of a criticality. Each criticality has a time duration associated with it, known as the *window-of-opportunity* (W_o), within which response actions have to be taken for controlling the criticality. A criticality is effectively controlled only if all the response actions for it are executed within its W_o. A criticality whose W_o has passed is said to be *expired*. *Criticality management* is defined as a process by which the criticalities can be controlled. As illustrated in Figure 1, criticality management has four phases: 1) **Detection** - is responsible for detecting criticalities in a timely manner; 2) **Response** - facilitates the actions that need to be taken to bring criticalities under control; 3) **Mitigation** - deals with long term recovery efforts; and 4) **Preparedness** - analyzes the criticalities of the past and get ready for the future ones. It is executed when the criticality has been controlled, or before the system is deployed in order to determine and improve the effectiveness of the requisite steps in the other three phases.

Criticalities usually occur in groups. For example, a building catching fire is a criticality and the presence of trapped people within this burning building is an additional criticality, both of which need to be controlled (*i.e.*, building saved

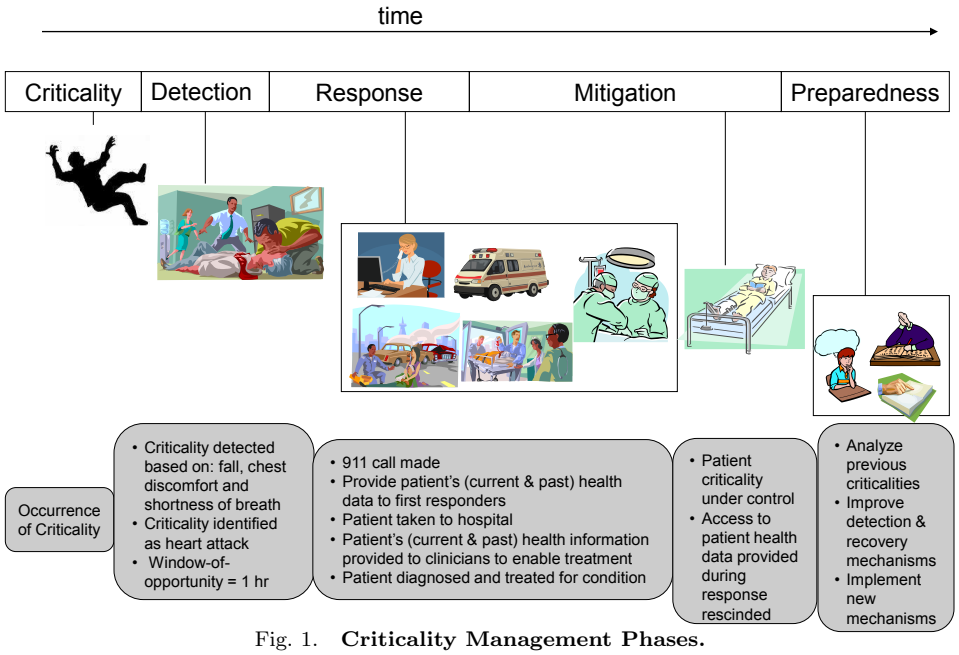


Fig. 1. **Criticality Management Phases.**

and people rescued). Handling multiple criticalities is considerably complex for various reasons: 1) the system has to not only keep track of the occurrence of new criticalities but also the expiration of existing criticalities (*i.e.*, when either the criticality is successfully responded to or the W_o is expired); 2) the occurrence of criticalities, and the response actions required to control them have a stochastic nature due to the probability of human error in executing them [Mukherjee et al. 2006] (it is therefore possible that responding to a criticality might lead to other criticalities within the system and the determination of response actions at any time have to take these into account); and 3) with multiple criticalities, we have to prioritize the control of one criticality over the others, such that, the probability that all the criticalities have been controlled is maximized. Therefore, depending upon the combination of criticalities present in the system, the response actions required to control them may vary. For example, if a person is facing an angina, the principal response action to perform is to give them the required medication. If however the person facing the angina is in a room engulfed in fire, the fire control might have to be prioritized in the larger interests.

1.2 Access Control for Criticalities

One of the ways of securing sensitive information in smart infrastructures is by using access control models [Sampemane et al. 2002]. Access control models are typically used to authorize access to specific information and services for subjects in the system during day-to-day activities. We contend that for smart infrastructures the access control models can be easily applied for privacy-preserved emergency management, by using them to *facilitate response actions*.

In this article, we present **Criticality Aware Access Control (CAAC)**.

CAAC is an *adaptive* and *proactive* access control model designed to facilitate the control of multiple criticalities in smart infrastructures, while ensuring privacy preservation. Its principal aim is to provide *the right set of privileges, to the right set of subjects, at the right time, for the right duration*, to facilitate the control of all the active criticalities within the system. As its purpose suggests, CAAC is usually implemented in the response stage of the criticality management. We have identified two basic characteristics that CAAC needs to posses: 1) **Adaptiveness**: the ability to: i) determine the response actions required for handling the current set of criticalities within the smart infrastructure, and ii) change the privileges available to the subjects in order to maximize the chances of controlling all the criticalities within their W_o; and 2) **Proactivity**: the ability to determine the subjects for executing the response actions, and enable them to execute the required set of response actions (even those which are not allowed during normal operations) in an accountable manner, thus avoiding the need for any explicit access requests. Note that, once the criticalities present within the system have been brought under control, all privileges provided thus far are rescinded, as they are no longer needed. The same holds in the case where the W_o of one or more criticalities has expired.

It should be noted that there is a fundamental difference between criticality-awareness of CAAC and context-awareness that many traditional authorization models possess. Context-awareness takes into account the contextual information of the subject making the request while making its access decision. On the other hand, with criticality-awareness, the contextual information considered is for the whole system and all its components (not the subject making the request alone). This system context is evaluated continuously and in case of a deviation from the norm, appropriate privileges are provided for subjects to deal with it, even without an explicit request from the subject.

The *contributions* of the article are as follows: 1) formalization of the CAAC approach and its policies, 2) validation of CAAC's principal design goals of proactivity and adaptiveness, and 3) a detailed case study that illustrates the operation of CAAC on an oil-rig with medical and fire criticalities. In the rest of the article, the terms system and smart infrastructure are used interchangeably.

The article is organized as follows. Section 2 presents the CAAC and its characteristics, policy specifications and its implementation. Section 3 presents the validation of CAAC based on the design goals we identified. Section 4 presents a detailed case study to demonstrate CAAC's execution. Section 5 presents the related work followed by Section 6, which concludes the article.

2. CRITICALITY AWARE ACCESS CONTROL (CAAC)

This section presents CAAC model designed to facilitate response actions to criticalities within the system. We begin with the system model, and then move on to the design goals, the model primitives, and the response action facilitation scheme.

2.1 System Model

In this section we present some of our principal assumptions with respect to the operation of Criticality Aware Access Control. CAAC is assumed to be deployed in a smart infrastructure. It views the constituent entities of the infrastructure as belonging to one of the two groups - 1) *Objects:* which are both physical and virtual

entities which provide a variety of information and services, and 2) *Subjects:* which are inhabitants of the environment who access information and services provided by the objects. CAAC manages the access to the objects by the subjects within the environment. CAAC itself is deployed and managed by a trusted *administrator*.

All access control systems need an underlying authentication system to function. For this work, we assume that the smart-infrastructure has an authentication system which can reliably identify subjects, as in [Bhargav-Spantzel et al. 2006]. Note that the technology used by the authentication system is not the focus of this work; we simply assume that it has the ability to authenticate subjects reliably and provide this information to the access control system in place. Finally, we assume that all criticalities are detected reliably, *i.e.*, their types and properties are accurately known at the time of detection using techniques such as [Liu et al. 2004].

2.2 Design Goals

As mentioned in Section 1.2, principal characteristics that access control models for privacy preserving criticality management need to possess are *adaptiveness and proactivity*. To ensure this, we consider them as *design goals* CAAC needs to meet. In this regard, we present five criteria, which characterize the two design goals. They are: 1) *Correctness*: which ensures that the response actions are facilitated through access privileges only as a consequence to the occurrence of critical events within the system; 2) *Liveness*: which requires that any access privilege provided in response to critical events is only for a finite amount of time; 3) *Responsiveness*: which ensures that occurrence of criticality facilitates response actions, which requires provision of access privileges to the right set of subjects, and subject notification; 4) *Non-repudiability*: which mandates that all response actions taken within the system during criticality are recorded for accountability purposes; and 5) *Safety*: which ensures that only authorized change to resources and access control constructs can happen within the system.

We contend that the first three criteria are necessary for demonstrating CAAC's adaptiveness, while the last three demonstrate its proactivity. Intuitively, if CAAC is adaptive, it has to determine the privileges needed for managing a criticality (responsiveness), change the access privileges to subjects based on the changes in the critical states within the system (correctness), in a temporary manner (liveness). Similarly, if CAAC is proactive, it has to be able to legitimately authorize subjects with appropriate privileges in order to enable them to take response actions (safety), without any explicit access request (responsiveness), in an accountable manner (non-repudiation).

2.3 CAAC: Primitives

CAAC is implemented in the response stage of criticality management. It specifies the privileges that subjects get on various objects in the system under both normal situations and criticalities. In normal situations, the default privileges are provided, while during criticalities, alternate ones are issued. These privileges may allow subjects with alternate (greater or lesser than before) capabilities than they normally would have. The CAAC model uses the basic constructs of Role Based Access Control (RBAC) [Sandhu et al. 1996] for controlling access for subjects (S) to objects (O) in the system. We use RBAC as we want to demonstrate how crit-

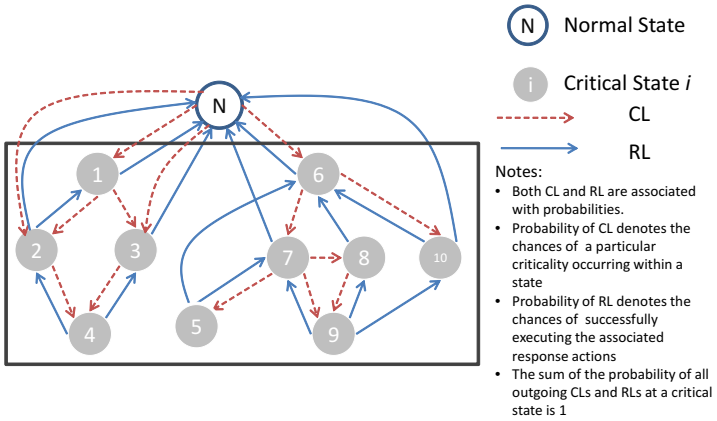


Fig. 2. **Example Action Generation Model, which illustrates the hierarchy of normal and critical states of a system**

icality awareness can be incorporated in an existing, widely used access control model. Subjects in the system have a role associated with them. Roles (R) are representation of subjects' responsibilities, and is assigned to them when they become part of the system. Examples include a doctor joining a hospital being assigned the roles of a *surgeon* and *doctor of patient X*. Even though a subject can have many roles, they can be activated only one at a time. The CAAC model keeps track of the current role the subject has taken and provides mechanisms for switching between the roles. To determine the actions that subjects can perform on specific objects, their roles are indexed into an Access Control List (ACL) maintained by objects in the system. ACLs are tables defined for each object in the system which maps roles to associated privileges. Privileges (PR) are authorizations which allow subjects to execute specific actions on specific objects within the system. For example, privilege for reading a file, using an equipment, or deleting a record.

Initially, when CAAC is being setup the administrator of the system establishes the set roles, privileges and their default mappings (ACL). Under normal situations, CAAC behaves similar to a context aware access control model akin to schemes such as [Hu and Weaver 2003]. When the system experiences criticalities (C), the access control model becomes more proactive. It evaluates the criticalities in the system, identifies the response actions that need to be taken, and proactively enables them. The chosen subjects can now access the system with alternate set of privileges than normally available to them for criticality management.

2.4 Criticality Response Facilitation in CAAC

In this section, we present how CAAC facilitates the response actions in a smart infrastructure. It has two phases: i) *preparation*, which involves identifying the response actions; and ii) *execution*, which involves identifying the subjects to manage the active criticalities and temporarily providing them with the required privileges to execute the actions.

2.4.1 *Preparation.* This phase of CAAC relates to the *preparedness phase of the criticality management process*. The set of actions for all possible combinations of criticalities that can occur within the system are determined in this phase. For example, emergency procedure manuals are developed for managing common emergencies such as fires and earthquakes in buildings. The current techniques for determining the response actions during criticalities are qualitative in nature. Having quantitative models can allow them to be evaluated in a computer and their results available electronically in a format understood by CAAC.

Action Generation Model: We have developed a novel and effective way of determining the response actions for criticalities within the system called the *Action Generation Model* (AGM). This model is based on the criticality modeling framework defined in [Mukherjee et al. 2006]. It consists of two types of states: *normal* and *critical*. When a criticality occurs, the system transitions from the normal state to a critical one. The system's state changes as new criticalities occur or get controlled. The system reaches the normal state only when all active criticalities in the system are controlled. The transitions which take the system toward the normal state are called *Response Links* (RL) and those that increase the number of active criticalities in the system are called *Critical Links* (CL). Each CL and RL has a probability associated with it. Figure 2 shows an example AGM. The dashed lines are the CLs while the solid lines are the RLs. The sum of the probabilities of all the CLs and RLs originating from a given state is 1. The CL specifies the probability of a particular criticality occurring given the current state of the system. Similarly, RLs represent the execution of the associated response action, taking the system from a lower state toward the normal state, and the probability of successfully executing them (based on human error probabilities). Each RL and CL also has a time parameter associated with it, which determines the required time for taking the link. For CL this signifies the time to detect the occurrence of the criticality and for RL it is the time to take the response actions. From a given critical state, the system may have multiple RLs (each representing a different response action) which can potentially take it to the normal state.

The choice of a particular RL depends upon its *P*-value*, which is a combination of three factors: 1) the probability of successfully reaching the neighboring state from the current state, by taking a RL; 2) the probabilities of successfully reaching the normal state from the neighboring state, which is compiled as an aggregated value by considering all possible paths to the normal state from it; and 3) conformance to the window-of-opportunity of all the active criticalities in the system. For a RL to be chosen, it is desirable to have the maximum P*-value, since this represents the best response actions, given the current state the system is in. For example, if the system is in State 4 in Figure 2, then chosen RL is given by $\max(\{p(4,2) * P(2,N)\}, \{p(4,3) * P(3,N)\})$ provided the W_o is met for all the criticalities active in State 4. Here $p(i,j)$ is the probability of reaching State j from State i, and $P(k,N)$ is the aggregate probability of reaching normal state from State i through all possible paths to N, for example $P(2,N) = p(2,N) + p(2,1) * p(1,N) + p(2,1) * p(1,3) * p(3,N)$[1]. The aforemen-

[1]We assume that during a criticality response, the critical states from which we have moved up the hierarchy will not be reached again

tioned computation of P*-value of each RL and choosing the one with the maximum value, called the *optimal planning criterion*. However, using P*-value as the basis of identifying the next RL suffers from two problems: 1) the state space explosion in computing the P*-value, and 2) the computation of P*-value returns zero, if the W_o of any of the active criticalities expire, in which case, no RL is returned from the current state. To overcome this problem, we use two *heuristic planning criteria*, which are greedy in nature: 1) choosing the RL with *Maximum Probability* (MP) at the current state, and 2) choosing the RL at the current state whose actions take the *Minimum Time* (MT).

The AGM is executed in an off-line manner during the preparedness phase of the criticality management process. Using tools like AGM have an important advantage. They allow the identification and planning for situations where a given combination of criticalities cannot be controlled. Such situations can then be improved by designing better response actions, faster criticality detection mechanisms in order to maximize the probability of reaching the normal state from a critical state. If for some reason, it is not possible, we can use a mixture of optimal and heuristic planning criteria for different states of the AGM to try to improve the overall goal, minimizing the number of criticalities within the system.

Response Action Generation for CAAC: In order to automate the AGM execution process, a tool called *Criticality Response Evaluation Tool* (CRET) is used. The tool takes as input (in XML form) the set of states, the CLs, the RLs, and applies optimal or greedy strategies for determining the RLs of choice at different critical states [Mukherjee and Gupta 2009]. However, it lacks one capability, in that it does not allow the specification of *tasks*, i.e. sub-actions that constitute response action). As there may be multiple ways of responding to a criticality, the question now arises as to which one to choose as the task associated with a RL.

The process of choosing the response action to be associated with a RL has to be first and foremost *risk-averse*, taking into account the following factors: 1) probability of success, 2) knowledge of the number and capability of subjects who may execute the actions, and 3) the availability of resources to pursue the action. The relative priority associated with these factors is again system dependent. For example, if the criticality is a ventricular fibrillation on a subject with implanted pacemaker the possible task associated with responding to it could be: 1) command the pacemaker to shock, and 2) use external defibrillator. If we assume that both the actions require one subject and the necessary tools are available, and (1) has 90% success rate compared to (2) whose success rate is 10%, we associate the action 'command the pacemaker to shock' with RL whose probability is now 0.90.

We extend the CRET tool to allow the specification of response action with RLs. This is done in the form of a tuple with two elements $<ObjectID, Privileges>$, the object on which the action is to be executed and the privileges required on them. In our previous example, object in question is the *defibrillator* and the privilege is *use*. As the object-privilege tuple enables only a specific action, it can be chosen such that it follows the *principal of least-privilege* in facilitating the response action.

In general, each RL is associated with a task set $TS = \{a_1, a_2, ...a_k\}$, where k is the number of response sub-actions. The privilege associated with a RL will be provided to the chosen subjects (see the next section) even if they conflict with the

subjects' default privileges, since criticality response is of paramount importance for the long-term stability of the system.

2.4.2 *Execution.* Given the identified criticality response actions in the preparation phase, the execution phase identifies the subjects and provides them the proper access privileges to perform the actions.

Subject Selection: We posit that subject selection in CAAC is primarily a function of criticality, as it is usually the nature of the problem which determines the number and type of people required to address it. Given the knowledge of the types of criticalities within the system during the preparation phase, subject selection for taking response actions can be primarily done in two ways: 1) *statically*, *i.e.*, the set of subjects required for controlling it can be pre-determined and stored in a static list depending upon the criticality; and 2) *dynamically*, *i.e.*, determining the set of subjects depending on the system context. For example, if the possible criticality is a fire, then the subjects required for controlling it are fire-fighters in the nearest fire station. As such, these fire-fighters can be statically selected. However, this approach does not work in situations where it is not possible to know the identity of the type of subjects to be chosen, beforehand. For example, in the event of a fire on floor X of a building, all the subjects on the floor need to be evacuated, and should therefore be given the required privileges to allow them to escape. To handle such situations, rules could be defined which specify contextual criteria (`location = Floor X`) for the subjects in order to be selected.

Enabling Response Actions: Once the action that needs to be taken at the current system state has been identified, and the subjects to execute the actions have been chosen, the actions need to be enabled, the subjects notified and then rescinded at a later time. This is done in three steps: 1) *provisioning alternate privileges* to the chosen subjects; 2) *informing selected subjects* of the new privileges; and 3) *rescinding alternate privileges* after the responses are performed or the windows-of-opportunity are expired. The alternate privileges are provided by assigning the selected subjects with a new temporary role (CAAC-Role) and adding a new entry to appropriate object's ACL. Since the underlying roles of the selected subjects have been modified, they cannot perform any of their tasks they can perform in normal situations, unless it is part of the response actions. Once the appropriate changes to subjects' role and objects' ACL have been made, CAAC informs the subjects of their new roles, the criticalities at hand and the response actions to perform. The system then maintains detailed records of the alternate privileges provided to subjects and the actions that were taken with these privileges.

The system is further designed to periodically check its state to determine the current set of response actions (based on the preparation phase). The privileges provided in the previous states are rescinded once they are no longer required to perform the response actions in a new state. Thus, any conflict between any privileges provided in the past and those provided for responding to current criticalities within the system is avoided. Such rescinding of privileges also takes place if the system has reached the normal state (no criticality) or that criticality is beyond control, in which case the system has to be audited based on its logs.

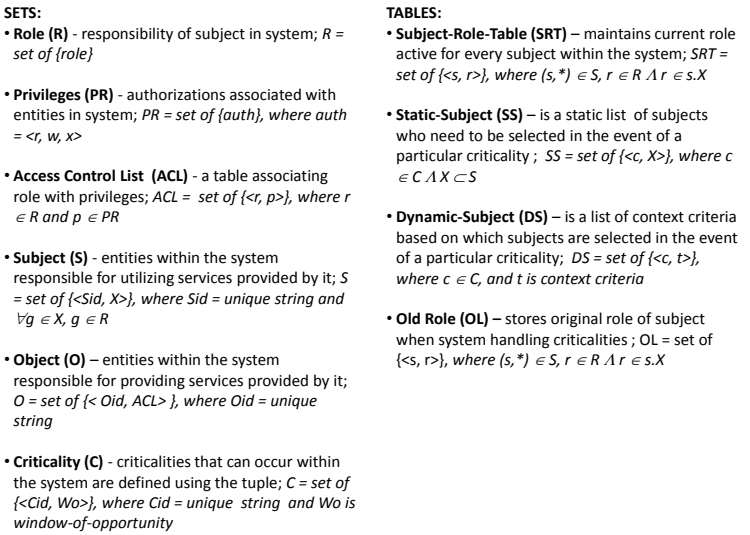


Fig. 3. **Principal Components of CAAC Policy Specifications.**

3. CAAC POLICY SPECIFICATION AND IMPLEMENTATION

Given the phases of CAAC in the previous section, this section presents the CAAC policy specification[2] and implementation (Figure 4) to facilitate the criticality response actions in the execution phase. Figure 3 shows the principal components used for the CAAC policy specification such as the notion of roles, subjects, objects, privileges, and access control lists as originally described in Section 2.3. The *Subject-Role-Table* (*SRT*) maps the subjects to specific roles. The *Static-Subject* (*SS*) and the *Dynamic-Subject* (*DS*) tables store statically and dynamically selected subjects (Section 2.4.2), respectively, to perform the response actions. The *Old-Role* (*OL*) table stores the original roles of the subjects who were given alternate roles during criticalities to perform the response actions. Given the principal components, the following subsections present the access control policies in CAAC.

3.1 Administrative Control Policies

These are the policies which are used to perform the basic functions of the CAAC model such as adding and removing subjects, associating and dissociating subjects with roles, updating ACLs and so on. Each of the policies can be executed only by the administrator of the smart infrastructure. We assume the presence of a system-dependent function $Auth()$ to authenticate subjects with administrative privileges ($Admin$). The symbols s, p, r, and o are used to index the subjects, privileges,

[2]The policies are described in the guarded command language form where a sequence of guards are followed by sequence of actions [Sampemane et al. 2002], represented as: $guard \longrightarrow command$. A policy rule is read as: if *guard* is true then execute *command*. The guard is usually a predicates which must hold before the command is executed. The guard may represent the contextual parameters such as occurrence of events or presence of specific conditions - time, user characteristics (location, designation). The command on the other hand usually specifies the action that need to be taken if the contextual cues in the guard are satisfied.

roles, and objects, respectively.

(1) **Add Subject**: Adds a subject (s) to the system ($AddSubject$ predicate is used for this purpose), along with the set of roles that the subject can take:
$AddSubject(s, roles, s_{admin}) \wedge Auth(s_{admin}, Admin) \wedge \forall r \in roles,\ r \in R \longrightarrow S = S \cup \{s, roles\}$.
Removing subjects ($RemoveSubject$) is a simple extension of this policy where the action $S = S \cup \{s, roles\}$ is replaced by $S = S - \{s, roles\}$.

(2) **Activate Role**: Activates the role of a subject ($ActRole$ predicate is used for this purpose) by storing it in a specific subject-role (SRT) table:
$ActRole(s, s_{admin}, r) \wedge (s, *) \in S \wedge Auth(s_{admin}, Admin) \wedge SubjectOwns(s, r) \longrightarrow SRT := SRT \cup \{s, r\}$.
Here the predicate $SubjectOwns(s, r)$ checks if r is one of the roles that s can activate. To deactivate the roles ($DeActRole$), the action $SRT := SRT \cup \{s, r\}$ is replaced by $SRT := SRT - \{s, r\}$:

(3) **Add ACL**: Adds a new ACL entry to an object ($AddACL$ predicate is used).
$AddACL(r, p, o, s_{admin}) \wedge Auth(s_{admin}, Admin) \wedge (o, *) \in O \wedge p \in PR \wedge if\ \nexists \{r, p\} \in o.ACL \longrightarrow o.ACL := o.ACL \cup \{r, p\}$.
To remove ACL ($RemoveACL$), the action $o.ACL := o.ACL \cup \{r, p\}$ is replaced by $o.ACL := o.ACL - \{r, p\}$.

3.2 Access Control Policy

This access control policy (ACP) is used to evaluate the access request of specific subjects and provide the requested privileges if the request holds true:
$ACP(s, o, p) \wedge (s, *) \in S \wedge (o, *) \in O \wedge p \in PR \wedge \{currentRole(s), p\} \in o.ACL \wedge$ (if($mode \neq critical$)then($sContext(s) == oContext(o)$) else true) $\longrightarrow allowAcc()$.
Here the functions $sContext(s)$ and $oContext(o)$ return the current context of the subject s and the context expected by the object o, respectively. The contexts of subjects and objects are evaluated only under normal situations and not in critical mode. Further, $currentRole(s)$ returns the present role of the subject s, and $allowAcc()$ is a Boolean function allowing the requested access.

3.3 Criticality Control Policies

These policies are used for enabling the CAAC model to control the criticalities that exist within the system. There are three main policies which accomplish this task, which we describe below:

(1) **Alternate Privileges**: This policy (predicate $AltPriv$) provides the alternate privileges required to enable criticality management. It utilizes the task set TS generated by the $getTS()$ function for the current state the system is in, as described in the Section 2. Each $a \in TS$ is an action of the form $< o, p >$, where $o \in O$ and $p \in PR$. It also needs a list of chosen subjects Sub who will be granted the alternate privileges. The original privileges of the subjects are stored in the table OL, which will be used to reset the subject's role:
$AltPriv(Sub, TS) \wedge \forall s \in Sub,\ (s, *) \in S \wedge \forall a \in TS,\ (a.o, *) \in O,\ a.p \in PR \longrightarrow \forall a \in TS,\ AddACL(CAAC\text{-}role, a.p, a.o, s_{admin}),\ \forall s \in Sub,\ OL := OL \cup (s, currentRole(s)),\ ActRole(s, s_{admin}, CAAC\text{-}role)$.

(2) **Inform Subjects**: It is used to inform the chosen subjects (Sub) they have alternate privileges (predicate $InformSub$). A system specific function $Inform$

```
CAAC_EXEC()
1.  stateChange := false
2.  selSubject := ϕ
3.  Mode := Normal
4.  while(true)
5.      t := checkSystem State()
6.      if(t ≠ currentState)
7.          currentState := t
8.          currentCrit = findCrit(t)
9.          RescPriv()
10.         if(t ≠ Normal)
11.             mode := CAAC
12.         else
13.             mode := Normal
14.         endif
15.     endif
16.     if (mode == CAAC)
17.         TS = getTS(currentState)
18.     foreach (e ∈ SS)
19.         if (e.c == currentCrit)
20.             selSubject := selSubject ∪ {e.X}
21.         endif
22.     endforeach
23.     foreach (d ∈ DS)
24.         if (d.c == currentCrit)
25.             foreach (s ∈ S)
26.                 if (uContext(s) == d.t)
27.                     selSubject := selSubject ∪ {s}
28.                 endif
29.             endforeach
30.         endif
31.     endforeach
32.     if (AltPriv(selSubjects,TS))
33.         InformSub(selSubjects,TS)
34.         RecordActions()
35.     endif
36.  endif
37.  Wait t_p
38. endwhile
```

Fig. 4. **CAAC Execution Model.**

is used to perform the required action for this purpose:
$InformSub(Sub, TS) \wedge \forall s \in Sub, (s, *) \in S \wedge \forall a \in TS, (a.o, *) \in O, a.p \in PR \longrightarrow \forall a \in TS, Inform(Sub, a.p, a.o, s_{admin})$.

(3) **Rescind Privileges**: This policy (predicate $RescPriv$) rescinds the alternate privileges, provided earlier, in the event of a change in the number of criticalities or elimination of all criticalities from the system:
$RescPriv() \longrightarrow \forall e \in OL, ActRole(e.s, s_{admin}, e.r)$.

3.4 Implementation

The overall execution model of CAAC is described in Figure 4, which gives the pseudo-code for the entire process. The model runs in an infinite loop monitoring the system for change in the system state. The system dependent $checkSystemState()$ function is used for this purpose. If a change is detected, we check if the system is now moved away from its normal state. If so, the system is moved to the CAAC mode. In the CAAC mode, given the current state of the system, the function $getTS()$ returns the TS by simply mapping the state to the result of CRET execution. Once TS is known, the set of subjects necessary to carry out the actions are selected based on the SS and DS tables. Each of the selected subject is provided with the privileges to execute the actions in TS using the $AltPriv$ policy, and then informed using the $InformSub$ policy. All actions taken when the system is in the critical state is recorded using the $RecordActions()$ function. The function is system dependent and we therefore refrain from defining it here. The recording is required to ensure accountability to the CAAC model to ensure that any malicious activity performed using the alternate privileges for criticality control are detected. Once the privileges have been provided, the system waits for t_p duration of time, and repeats the whole process all over again. If the system moves from one critical state to another, then any alternate privileges are rescinded using the $RescPriv()$ function.

CAAC is designed to be proactive and adaptive in nature and is expected to

Table I. **Criticalities and Properties.**

ID	Criticality	W_o	Task-Set (TS)	Exec. Time
c1	Heart Attack	5 min	$\{< Defib, < -, -, True >>,$ $< Health\ Data, < True, True, - >>\}$	1 min
c2	Fire	20 min	$\{< Fire\ Exit, < -, -, True >>\}$	2 min
c3	Unstable Angina	60 min	$\{< Health\ Data, < True, True, - >> \}$	1 min
c4	Fire Assistance	12 min	$\{< Control\ Room\ Door, < -, -, True >>\}$	2 min

"$-$" means the corresponding privilege is not applicable to the object in **TS** (*e.g.*, read and write privileges are not applicable for defibrillator, $Defib$).

provide right access control privileges to right set of subjects at the right time for the right duration, in order to control active criticalities within the system. In Section 2 we defined a set of criteria which demonstrate CAAC's adherence to its design goals of proactivity and adaptiveness. In the following section, we present semi-formal proofs to illustrate the CAAC policy specification's adherence to the design goals.

4. VALIDATION

We assume for the proofs the access policies are implemented and enforced correctly.

THEOREM 4.1. **Responsiveness:** *When a critical event occurs - 1) the subject is immediately notified, and 2) its access privileges are changed.*

PROOF. The proofs of the claims above are as follows: 1) When there is a criticality, the subjects are notified in Line 33 of Figure 4. 2) The role of the subject being notified is updated in Line 32 of Figure 4 using the policy $AltPriv$. □

THEOREM 4.2. **Correctness:** *Subjects get alternate set of privileges if and only if there is at least one un-controlled criticality in the system.*

PROOF. If there is at least one un-controlled criticality in the system, the *mode* variable is set to $CAAC$ in lines 10 - 15 of Figure 4. This results in the execution of Lines 16-34 of Figure 4, thus providing alternate set of privileges to subjects. If a subject is allowed to execute actions which are enabled by alternate privileges (line 32 of Figure 4), the *mode* has to be set to $CAAC$. As this can happen only if the *currentState* of the system is in critical state, the result follows. □

THEOREM 4.3. **Liveness:** *The maximum duration for which subjects are assigned alternate privileges is limited by - the time instant when the number of active criticalities in the system changes.*

PROOF. From Theorem 4.2, it follows that subjects receive alternate privileges if and only if there is at least one criticality in the system. The alternate set of privileges for subjects are rescinded in Line 9 of Figure 4 and new privileges are provided by executing Line 32 of Figure 4. Both these actions can be executed only if there is a change in the current state of the system (Line 6, Figure 4), which can happen only if there is a change in the number of active criticalities (occurrence of new one or control of existing one) in the system. □

THEOREM 4.4. **Non-Repudiation:** *Malicious use of alternate privileges when system is experiencing criticalities is non-repudiable and limited to a finite duration.*

PROOF. Line 34 of Figure 4 ensures that whenever a role is changed and a request for resource is successful, it is recorded along with the appropriate start and end times enforcing non-repudiation of any malicious activity by a subject due to the new privileges. As we assume that all the access control policies execute correctly, the records are accurately updated. From Theorems 4.2 and 4.3, it follows that subjects are granted alternate privileges only in the presence of criticalities and the maximum time for which the alternate privileges are granted is limited by the time when a change in the number of criticalities occurs, thereby limiting potential malicious activity to a finite amount of time. □

THEOREM 4.5. **Safety:** *Only authorized access is allowed to 1) objects and 2) access control constructs within the model.*

PROOF. 1) Access to resources is allowed, only if ACP evaluates to true. Now, as in both normal and critical situation subjects access to an objects is only by invoking the ACP, authorized access is ensured. 2) Access to modify constructs of the access control model can be done by executing any of the CAAC administrative policies. Each of these predicates can be executed only by CAAC using the administrator role (U_{admin}), thus ensuring authorized access to the access control model. □

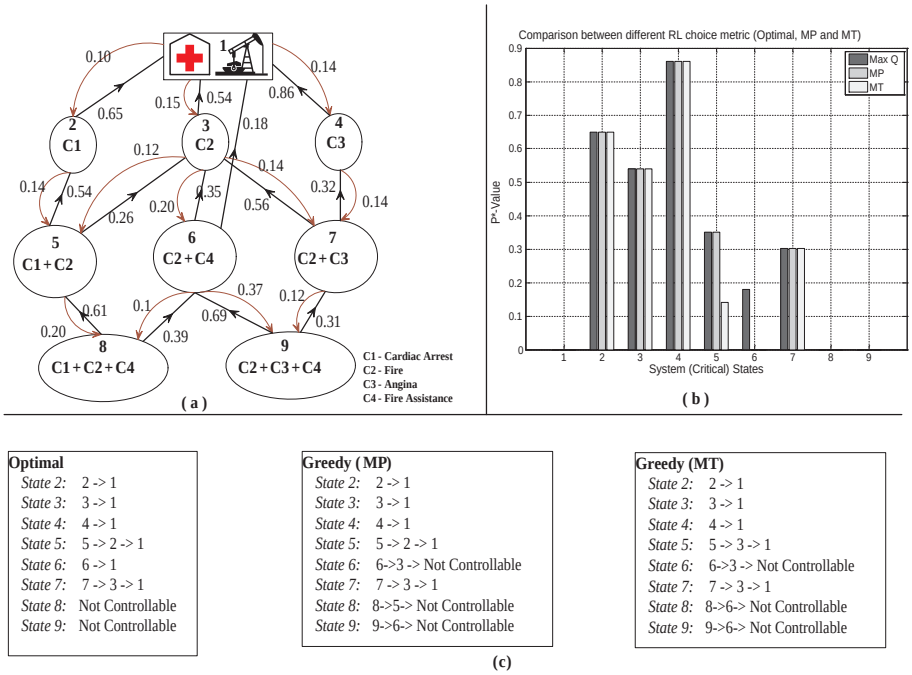
We claim from the above theorems that CAAC satisfies all the five criteria set forth in Section 2.2 and hence meets the design goals of proactivity and adaptiveness.

5. CASE STUDY: MEDICAL AND FIRE EMERGENCY ON AN OFFSHORE OIL RIG

In this section, we give an example of how CAAC can be used for criticality response. The goal is to show the ability of CAAC to handle any conceivable combination of criticalities in a system. Consider an emergency situation on an oil rig which are prone to emergencies situations. The recent blow-out of Deepwater Horizon rig caused massive oil spill, several casualties and eventual sinking of the rig itself [Oil Spill 2010]. We look at a combination of two types of criticalities, in such situations, for this example: 1) health-related and 2) fire accident. We assume that the oil rig is a smart infrastructure where each individual subject can interact seamlessly with their environment to obtain services. The smart-environment also keeps track of the subjects within the rig. The entire rig is managed by a Rig Management Environment (RME) that provides the common interface for subjects to interact with. The following subsection discusses the preparation and execution phase of CAAC in this example.

5.1 Preparation

Before the rig is deployed, in order to handle criticalities within this environment, planners and engineers of the oil-rig will have to execute CRET in order to determine the actions which need to be taken during specific emergencies. Four possible criticalities within the oil rig are considered for this example: **c1**: a worker on the rig with a chronic hypertension having a heart attack in the control room; **c2**: fire alarm in the control room of the rig; **c3**: a worker on the rig with a chronic hypertension having unstable angina in the control room; and **c4**: people trapped in control room needing immediate assistance. The Table I provides details for each criticality and its important characteristics. Figure 5(a) shows the AGM that would be generated by CRET using the stochastic model described in Section 2.4.1.



Optimal

State 2:	2 -> 1
State 3:	3 -> 1
State 4:	4 -> 1
State 5:	5 -> 2 -> 1
State 6:	6 -> 1
State 7:	7 -> 3 -> 1
State 8:	Not Controllable
State 9:	Not Controllable

Greedy (MP)

State 2:	2 -> 1
State 3:	3 -> 1
State 4:	4 -> 1
State 5:	5 -> 2 -> 1
State 6:	6->3 -> Not Controllable
State 7:	7 -> 3 -> 1
State 8:	8->5-> Not Controllable
State 9:	9->6-> Not Controllable

Greedy (MT)

State 2:	2 -> 1
State 3:	3 -> 1
State 4:	4 -> 1
State 5:	5 -> 3 -> 1
State 6:	6->3 -> Not Controllable
State 7:	7 -> 3 -> 1
State 8:	8->6-> Not Controllable
State 9:	9->6-> Not Controllable

(c)

Fig. 5. **CAAC Example: a) AGM for Medical and Fire Criticalities in Oil Rig, b) Criticality Response P*-Values at each critical state, c) Path to Normal State for each Critical State.**

The probabilities of the CLs and RLs have been obtained from various studies on occurrences of emergencies and response errors for medical ailments [Chan et al. 2008], [Pope et al. 2000], [Khot et al. 2003], and [Hendrix et al. 2004] and oil rig management [DiMattia et al. 2005]. The CRET is then executed to determine the response actions for each critical state in the system using the optimal planning criterion for choosing RL at each state (based on P*-value).

Figure 5(b) shows comparative results, in terms of P*-value, on probability of successfully controlling all the active criticalities within the system. It can be seen that as the number of criticalities in the system increases, the probability of reaching the normal state decreases. Further, as the number of states in this example are limited, the optimal solution and the heuristics yield similar P*-values. The execution of CRET gives us the next link from each critical state in order to have the maximum probability of reaching the normal state. These collections of links can be viewed as a path to the normal state. Figure 5(c) shows the paths yielded by the optimal and greedy criteria. Note that, even if the P*-value may be identical for the optimal and greedy approaches, the actual path and therefore the response actions that need to be taken may be different as can be seen for State 6 in Figure 5(c). For some critical states, the optimal planning criterion returns a zero P*-value. The heuristic planning criteria provides some set of actions in all the cases, but there is no guarantee that all criticalities in the system will be controlled, as in the case of States 8 and 9 for both MT and MP. Given the path, the planners

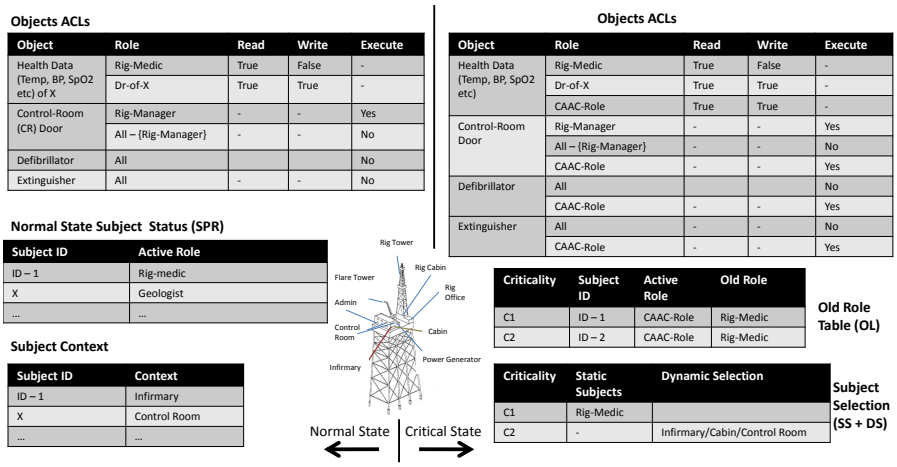


Fig. 6. **CAAC Example - Access Control Structures.**

can easily identify the set of actions that need to be executed, and the order in which they have to be executed, to ensure that the criticalities present within the system can be controlled. For example, using the optimal criteria, the path from State 5 goes to State 2 and then to the normal state. Therefore, in order to have a high probability of reaching the normal state, one needs to first take actions to respond to the fire and then respond to the heart-attack. Apart from determining the task set for a specific criticality, the planners also determine other requirements for specific criticalities such as subjects that need to take response actions. As mentioned previously, subjects can be selected based on their capabilities (statically) or based on their contexts (in this example we consider only the location context for simplicity).

5.2 Execution

Given the response actions that need to be taken at each critical state in the system, we now illustrate how CAAC functions. Figure 6 shows the access control constructs used by CAAC for performing access control in this scenario. The tables on the left side of the rig show elements in normal state, while the ones on the right show the values during criticalities. Consider the scenario where a hypertensive crew member (Geologist, ID-X) in the control room has had a heart attack ($c1$). The CAAC model which is routinely evaluating the system state (every t_p time units) notices that the system is not in the normal state any more but in the critical state (State 2 in the AGM). It determines that the path to reach the normal state is by responding to the criticality $c1$ directly. The task-set for this has two actions - enable defibrillation, and provide access to X's health information. CAAC then checks the SS and DS tables to identify the best subjects to take the criticality. This is the *rig-medic*. The CAAC then changes the role of the rig-medic to a temporary *CAAC-role*, updates the ACL of objects - defibrillator, and health data of X with the new role which has the appropriate privileges (see Figure 6). It then informs the rig medic about the changes, who can take the required action.

If before the arrival of the medic, a fire breaks out in the control room ($c2$), then the system is now in State 5 of the AGM. In this state, based on the prior execution

Table II. **Comparison of Different Access Control Classes with CAAC (Here R - RBAC, C - CA-RBAC, B - CBAC, U - UCON, O - OS, P - PBM S - PS, and A - COAC).**

Properties	R	C	B	U	O	P	S	A	CAAC
Proactive	-	-	-	-	-	-	-	Yes	Yes
Adaptive	-	-	-	-	Yes	-	Yes	Yes	Yes
Alternate Privileges	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Single Criticality	-	-	-	-	-	-	-	Yes	Yes
Multiple Criticality	-	-	-	-	-	-	-	-	Yes

of the AGM by CRET, leads CAAC to prioritize fire control over medical emergency. CAAC then determines that all the subjects within the range of the control room - *i.e.*, those in infirmary, cabins and control room be given the privileges to control the fire (Technician,ID-3; Rig-Manager,ID-4; and Rig-Medic,ID-1 in our example). Further, the task set for criticality $c2$ states that the object fire extinguisher be provided access to. CAAC provides the privileges to all the people in the three locations with access to the fire-extinguisher object, by changing their active role to CAAC-role (the old roles are stored in the OL table) and adding an entry to fire extinguisher object's ACL (see Figure 6). It then informs all the chosen subjects about the changes, who can take the required action.

As the state of the system changed before providing privileges for fire fighting, it rescinds the privileges of the rig-medic for handling $c1$, in order to enable them to fight the fire which is far more dangerous to many more people. Once the fire is controlled the system moves back to State 2 in the AGM, and the rig-medic is provided the privileges for accessing X's data and using the defibrillator as he sees fit. The privileges for fire-fighting are rescinded, and the old roles of subjects involved are returned (except rig-medic who is now the chosen subject for $c1$). Once the heart-attack is controlled as well, the system is in the normal state and the role of the rig-medic is also changed to the default value. If suppose the X's heart-attack cannot be controlled within its W_o, then the privileges of rig-medic are rescinded by changing their roles to the original values. All the actions taken during the criticality response are recorded for determining the effectiveness of CAAC. The example thus demonstrates how CAAC can be used for managing criticalities within the system by providing the right set of privileges, to the right set of subjects, at the right time for the right duration.

6. RELATED WORK

Much work has been done with respect to access control for smart infrastructures and other systems. This section discusses some of the prominent classes of access control and their utility for privacy preserved criticality management. Table II presents a summary of the capabilities of CAAC and traditional access control models.

One of the most influential access control models for enabling authorized information release is the *Role Based Access Control* (RBAC) [Sandhu et al. 1996]. By de-coupling the process of directly associating privileges with a subject, RBAC provide an effective and easy way of managing security and enforcement of complex access control policies within the system. The concept of RBAC has been generalized in [Moyer and Ahamad 2001] by incorporating subject roles, object roles and

environment roles. As most systems have dynamic requirements, RBAC was further extended by including different types of context information in the access control decision making process, leading to the development of *Context Aware-RBAC (CA-RBAC)* [Covington et al. 2001] [Joshi et al. 2005]. Over the years, newer paradigms of access control have been proposed such as *Context Based Access Control* (CBAC) [Corradi et al. 2004] which divert from the role-based approach, by associating privileges directly with context information for each subject and avoids the notion of roles increasing the simplicity. *Usage Control (UCON)* [Wang et al. 2006] is another alternative which combines the notions of access control, trust management and digital rights management to provide finer grained access control to a subjects who may not be known to the system. None of these schemes were designed for privacy preserved criticality management. They are reactive in nature and wait for subjects to ask for specific privileges, which may introduce arbitrary delays in response. Further, none of them consider the stochastic nature of criticalities, the need for urgent response actions and human error involved in executing them. In [Povey 2000], the author presents the notion of *Optimistic Security* (OS), which lets subjects exceed their default privileges but in a semi-constrained manner in that: 1) it records all actions taken during the time when subjects exceed their privileges, 2) it allows subjects to execute only those actions which can be rolled back, and 3) provided a supervisor agrees to it. OS, like CAAC framework, allows subjects to perform actions which require privileges exceeding their default values. However, it does not control the duration for which alternate privileges are provided and requires human intervention; thus making it unsuitable for criticality control.

Much work has also been done towards developing *Policy Based smart-space Management (PBM)* schemes. Such schemes allow for influencing the behavior of smart-spaces without hard coding the behavior into them [Bettini et al. 2002] [Kagal et al. 2003], [Sloman and Lupu 2002]. In case of an event, and a pre-defined set of system conditions take a particular action. The PBM schemes are adaptive and can be easily used to provide alternate privileges to subjects when needed. However, their lack of awareness of the criticalities, the associated stochastic characteristics along with their event based triggering of change are too simplistic to suite the needs of criticality response, especially in the event of multiple criticalities. A novel policy based model which possesses some of the adaptiveness of CAAC, is called the *Policy Spaces (PS)* [Ardagna et al. 2008]. PS divides the policies into groups which provide mandatory access, mandatory access denial and planned exceptions. The idea is that for specific situations, policies provide access which are not normally allowed, just as in CAAC. However, the principal drawback of both PS and PBM is that they are reactive in nature awaiting access requests from subjects before they allow or disallow any actions, which may waste valuable time.

The notion of altering access control privileges to enable criticality management for smart-spaces was first introduced in our preliminary work which we refer to as *Criticality-Oriented Access Control (COAC)* in [Gupta et al. 2006]. However, the scheme was limited in scope as it only addressed systems with single criticality. It did not provide a mechanism for determining the response actions or dealing with the stochastic nature of criticalities.

7. CONCLUSIONS

In this article, we presented the *Criticality Aware Access Control* (CAAC) approach for proactive and adaptive access control during emergencies. In this regard, CAAC facilitates response actions to specifically chosen subjects in system by providing them with access privilege for executing response actions, without their asking for it. It uses a stochastic model called Action Generation Model (AGM) to determine response actions, in an off-line manner, given a combination of criticalities are present within the system. Detailed policy specifications and implementation details for CAAC have been also been provided. Further, we have validated it based on its principal properties of proactivity and adaptiveness, and also provided a detailed example, in the context of an oil rig, of how CAAC functions.

Acknowledgements

We would like to thank the anonymous reviewers and the editors of the ACM TAAS Special Issue on Adaptive Security Systems, Drs. Yang Xiang and Wanlei Zhou, who helped improve this paper immensely.

REFERENCES

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

Ardagna, C. A., di Vimercati, S. D. C., Grandison, T., Jajodia, S., and Samarati, P. 2008. Regulating Exceptions in Healthcare using Policy Spaces. In *Proceedings of the Twenty-Second Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. 254–267.

Bettini, C., Jajodia, S., Wang, X., and Wijesekera, D. 2002. Obligation Monitoring in Policy Management. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks*. Springer-Verlag, 2–12.

Bhargav-Spantzel, A., Squicciarini, A., and Bertino, E. 2006. Privacy Preserving Multi-factor Authentication with Biometrics. In *Proceedings of the Second ACM Workshop on Digital Identity Management*. ACM, 63–72.

Chan, P. S., Krumholz, H. M., Nichol, G., and Nallamothu, B. K. 2008. Delayed Time to Defibrillation after In-Hospital Cardiac Arrest. *The New England Journal of Medicine 358,* 1 (January), 9–17.

Corradi, A., Montanari, R., and Tibaldi, D. 2004. Context-based Access Control Management in Ubiquitous Environments. In *Proceedings of the Third International Symposium on Network Computing and Applications*. IEEE, 253–260.

Covington, M. J., Long, W., and Srinivasan, S. 2001. Secure Context-Aware Applications Using Environmental Roles. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technology*. ACM, 10–20.

Denning, T., Fu, K., and Kohno, T. 2008. Absence makes the heart grow fonder: new directions for implantable medical device security. In *HOTSEC'08: Proceedings of the 3rd conference on Hot topics in security*. USENIX Association, Berkeley, CA, USA, 1–7.

DiMattia, G. D., Faisal, I. K., and Amyotte, P. R. 2005. Determination of human error probabilities for offshore platform musters. *Journal of Loss Prevention in the Process Industries 18*, 488–501.

Gupta, S. K. S. 2008. Towards Formal Framework for Modeling and Evaluation of High-Confidence Criticality-Aware Software for Distributed CPS: A White Paper. In *National Workshop for research on High Confidence Transportation Cyber Physical Systems: Automotive, Avaition, and Rail*.

Gupta, S. K. S., Mukherjee, T., and Venkatasubramanian, K. 2006. Criticality Aware Access Control Model for Pervasive Applications. In *Proceedings of the Fourth Conference on Pervasive Computing*. IEEE, 251–257.

Hendrix, K., Mayhan, S., Lackland, D., and Egan, B. 2004. Prevalence, Treatment, and Control of Chest Pain Syndromes and Associated Risk Factors in Hypertensive Patients. *American Journal of Hypertension 18,* 8 (May), 1026–1032.

Hu, J. and Weaver, A. C. 2003. A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *Proceedings of the First Workshop on Pervasive Security, Privacy and Trust*. ICST.

Joshi, J. B. D., Bertino, E., and Ghafoor, A. 2005. Analysis of Expressiveness and Design Issues for a Temporal Role Based Access Control Model. *Transactions on Dependable and Secure Computing 2,* 2 (April-June), 157–175.

Kagal, L., Finin, T., and Joshi, A. 2003. A Policy Language for A Pervasive Computing Environment. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks*. Springer-Verlag, 63–74.

Khot, U. N., Khot, M. B., Bajzer, C. T., Sapp, S. K., Ohman, E. M., Brener, S. J., Ellis, S. G., Lincodd, A. M., and Topol, E. J. 2003. Prevalence of Conventional Risk Factors in Patients With Coronary Heart Disease. *The Journal of American medical Association 290,* 7 (August), 898–904.

Liu, H., Motoda, H., and Yu, L. 2004. A Selective Sampling Approach to Active Feature Selection. *Artificial Intelligence 159,* 1-2 (November), 49–74.

Mehrotra, S., Butts, C., Kalashnikov, D., Venkatasubramanian, N., Rao, R., Chockalingam, G., Eguchi, R., Adams, B., and Huyck, C. 2004. Project RESCUE: Challenges in Responding to the Unexpected. In *Proceedings of the Sixteenth Annual Symposium on Electronic Imaging Science and Technology*. SPIE, 179–192.

Moyer, M. J. and Ahamad, M. 2001. Generalized Role Based Access Control. In *Proceedings of the Twenty-First International Conference Distributed Computing System*. IEEE, 391–398.

Mukherjee, T. and Gupta, S. K. S. 2009. CRET: A Crisis Response Evaluation Tool to improve Crisis Preparednesss. In *Proceedings of the International Conference on Technologies for Homeland Security*. IEEE.

Mukherjee, T., Venkatasubramanian, K., and Gupta, S. K. S. 2006. Performance Modeling of Critical Event Management for Ubiquitous Computing Applications. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM/IEEE, 12–19.

Oil Spill. 2010. Gulf of Mexico Oil Spill (2010). NY Times, `http://topics.nytimes.com/top/reference/timestopics/subjects/o/oil_spills/index.html`.

Pope, J. H., Aufderheide, T. P., Ruthazer, R., Woolard, R. H., Feldman, J. A., Beshansky, J. R., Griffith, J. L., and Selker, H. P. 2000. Missed Diagnoses of Acute Cardiac Ischemia in the Emergency Department. *The New England Journal of Medicine 342,* 16 (April), 1163–1170.

Povey, D. 2000. Optimistic Security: A New Access Control Paradigm. In *Proceedings of the Workshop on New Security Paradigms*. ACM, 40–45.

Sampemane, G., Naldurg, P., and Campbell, R. H. 2002. Access control for Active Spaces. In *Proceedings of the Eighteenth Annual Computer Security Applications Conference*. IEEE, 343–352.

Sandhu, R., Coyne, E. J., Feinstein, H. L., and Youman, C. E. 1996. Role Based Access Control Models. *IEEE Computer 29,* 2 (February), 38–47.

Sloman, M. and Lupu, E. 2002. Security and Management Policy Specification. *IEEE Network 16,* 2 (March/April), 10–19.

Venkatasubramanian, K., Deng, G., Mukherjee, T., Quintero, J., Annamalai, V., and Gupta, S. K. S. 2005. Ayushman: A Wireless Sensor Network Based Health Monitoring Infrastructure and Testbed. In *Proceedings of the Intlernational Conference on Distributed Computing in Sensor Systems*. IEEE, 406–407.

Wang, H., Zhang, Y., and Cao, J. 2006. Ubiquitous Computing Environments and its Usage Access Control. In *Proceedings of the First International Conference on Scalable Information Systems*. ACM, 6.