Proceedings of the IFIP WG11.3 Workshop on Database Security,
Lake Tahoe, California, August 11-13, 1997

Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management

R. K. Thomas[1] *and R. S. Sandhu*[2]
[1]*Odyssey Research Associates*
Cornell Business and Technology Park,
33 Thornwood Drive, Suite 500
Ithaca, NY 14850-1250, USA. *rthomas@oracorp.com*

[2]*Laboratory for Information Security*
ISSE Department -- MS 4A4
George Mason University
Fairfax, VA 22030, USA. sandhu@isse.gmu.edu

Abstract

In this paper, we develop a new paradigm for access control and authorization management, called task-based authorization controls (TBAC). TBAC models access controls from a task-oriented perspective than the traditional subject-object one. Access mediation now involves authorizations at various points during the completion of tasks in accordance with some application logic. By taking a task-oriented view of access control and authorizations, TBAC lays the foundation for research into a new breed of "active" security models that are required for agent-based distributed computing and workflow management.

Keywords

Active security models, authorizations-step, composite authorizations, secure workflows

Title of book. Name of Eds.

1 INTRODUCTION

In this paper, we describe a new paradigm for access control and security models, called task-based authorization controls (TBAC). TBAC is well suited for distributed computing and information processing activities with multiple points of access, control, and decision making such as that found in workflow and distributed process and transaction management systems.

TBAC differs from traditional access controls and security models in many respects. Instead of having a system-centric view of security, TBAC approaches security modeling and enforcement at the application and enterprise level. Secondly TBAC lays the foundation for a new breed of what we call "**active**" security models. By active security models, we mean models that approach security modeling and enforcement from the perspective of activities or tasks, and as such, provide the abstractions and mechanisms for the active runtime management of security as tasks progress to completion. In an active approach to security management, permissions are constantly monitored and activated and deactivated in accordance with emerging context associated with progressing tasks (such as in workflows). Such a task-based approach to security represents a radical departure from classical passive security models such as those based on one or more variations of the subject-object view of security and access control. In a subject-object view of security, a subject is given access to objects in a system based on some permissions (rights) the subject possesses. However, such a subject-object view typically divorces access mediation from the larger context (such as the current state of tasks) in which a subject performs an operation on an object.

Our focus in this paper is on active security models for authorization management and access control in computerized information systems. An authorization is an approval act and manifests itself in the paper world as the act of signing a form. Typically, in the paper world, an authorization results in the enabling of one or more activities and related permissions. The person granting the authorization usually takes responsibility for the actions that are authorized by the authorization. Also, an authorization, as represented by a signature, has a lifetime associated with it during which it is considered valid. Once an authorization becomes invalid, organizations require that the associated permissions no longer be available. As paper-based systems become computerized, the related authorization procedures will have to become automated. Thus the TBAC approach described in this paper was motivated by this anticipated need to automate authorization and related access controls. In particular, the implementation of TBAC ideas will lead to systems that provide tighter just-in-time need-to-do permissions especially in application environments consisting of transactions and workflows (see (Georgakopoulos, 95) and (Rusinkiewicz, 94)). Also, the TBAC approach leads to access control models that are self-administering to a great extent, thereby reducing the overhead typically associated with fine-grained subject-object security administration.

As mentioned earlier, the most obvious application of TBAC is in secure workflow management. TBAC enables the granting, usage tracking, and revoking of permissions to be automated and co-ordinated with the progression of the various tasks. Without active authorization management, permissions will in most cases be "turned on" too early or too late and will probably remain "on" long after the workflow tasks have terminated. This opens up vulnerabilities in systems. Any attempt to minimize such vulnerabilities will require a security administrator to keep track of the progress of the tasks for all enacted workflow instances; an error-prone and impossible task! Thus what is needed is an approach where access control permissions are granted and revoked according to the validity of authorizations and one where this can be done without manual security administration. The authorizations themselves are of course processed strictly according to some application logic and policy. In the remaining sections of this paper we will describe how TBAC ideas can be used to accomplish this.

There are basically two broad objectives guiding our research efforts in TBAC. The first is to model from an enterprise perspective, various authorization policies that are relevant to organizational tasks and workflows. We envision a set of user-friendly tools to help a security officer model and specify policies. Our second objective is to seek ways in which these modelled policies can be automatically enforced at runtime when the corresponding tasks are invoked. We limit the discussion in this paper to the core concepts in TBAC that form our conceptual framework. Various aspects of our research such as languages to model authorization policies, as well as, the runtime mapping of these policies to enforcement mechanisms are topics of ongoing investigation and will be reported in subsequent publications. We also do not address the TCB-style issues related to assurance, as our focus at this point is not on implementation.

Preliminary ideas for TBAC that recognized the need for active security were presented in (Thomas, 1993) and (Thomas, 1994). More recently, a workflow authorization model (WAM) was presented in (Atluri, 1996). WAM has the same general motivation as TBAC in that it tries to provide some notions of active security and just-in-time permissions. However, from a conceptual standpoint, TBAC is significantly different and more comprehensive than WAM. In WAM an authorization is a more primitive concept and represents the fact that a subject has a privilege on an object for a certain time interval. In TBAC an authorization (step) has much richer semantics as it models the equivalent of an authorization in the paper world. An authorization act in the paper world may result in the granting of several related permissions. Thus in TBAC an *authorization-step* is a convenient abstraction to model and manage a set of related permissions. TBAC also provides features such as usage tracking of permissions, lifecycle management, and the ability to put permissions temporarily on hold without invalidating them, as well as modeling sets of authorizations through composite authorizations.

2 BACKGROUND: FROM PASSIVE TO ACTIVE SECURITY

In this section we discuss how TBAC differs from the traditional (passive) subject-object view of access control.

2.1 The subject-object view of access control

In the subject-object view of access control, the basic entities are subjects, objects, and the rights subjects possess to gain access to the various objects. This can conceptually be represented in an access control matrix (Harrison, 1976). The horizontal and vertical projections of this matrix can be implemented in systems as capabilities or access control lists, respectively. From the standpoint of security models, the subject-object view of access control can be traced to the earlier security models such as the HRU model (Harrison, 1976) and its influence can be seen even in later work such as the typed access matrix model (TAM) (Sandhu, 1992).
A closer examination of the subject-object view of access control will reveal the following characteristics.

- The implicit assumption that there is a central pool of resources to which we need to provide access control.
- Access control information represents isolated units of security information.
- Access mediation is divorced from larger operation context.
- There is no memory of any evolving context associated with past accesses.
- There is no record of the usage of permissions.
- Existing permissions can be revoked but cannot be put on hold.
- Requires fine-grained security administration.

In summary, the subject-object paradigm of access control takes a very system-centric view of protecting a central pool of resources. It enforces a very simple access control discipline which can succinctly be stated as: *if a subject has requested an access operation to an object, and the subject possesses the permission for the operation, then grant the access*. Thus all the access decision function has to check is if the subject has the required permission. However this simplicity is precisely the limitation of subject-object access controls. No other contextual information about ongoing activities or tasks can be taken into account when evaluating an access request (some attempts at access control frameworks to overcome this limitation have been discussed in (Abrams, 1990) and (Abrams, 1991). Further, there is no record of the usage of individual permissions. So long as the permission exists (typically in a structure such as an access control list), the subject can issue an operation any number of times. We thus consider this to be a "passive" model of security. Next we discuss how TBAC forms an active approach to access control.

2.2 TBAC as an active security model for authorizations

We have coined the concept of active security models to characterize models that recognize the overall context in which security requests arise and take an active part in the management of security as it relates to the progress and emerging context within tasks (activities).

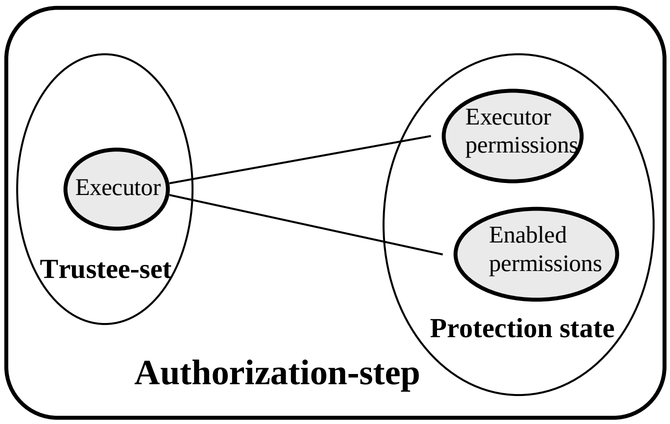

Figure 1. An authorization-step as an abstraction grouping trustees and permissions

Before we elaborate further on TBAC as an active model let us discuss some of the basic ideas in the TBAC approach.

One of the most fundamental abstractions in TBAC is that of an *authorization-step.* It represents a primitive authorization processing step and is the analog of a single act of granting a signature in the forms (paper) world. From the standpoint of modeling, it is an abstraction that groups trustees and various sets of permissions, as illustrated in Figure 1. In the paper world, a group of individuals may be potentially allowed to grant a certain type of signature. For example, all sales clerks may be allowed to sign sales orders. However, a single instance of a signature may be granted only by a single individual. For example, sales order 1208 is signed by sales clerk Tom. Similarly, in TBAC we associate an authorization-step with a group of trustees called the *trustee-set*. One member of the trustee-set will eventually grant the authorization-step when the authorization-step is instantiated. We call this trustee the *executor-trustee* of the step. The permissions required by the executor-trustee to invoke and grant the authorization-step make up a set of permissions called *executor-permissions*. Also, in the paper world, a signature also implies that certain permissions are granted (enabled). In a similar fashion, we model the set of permissions that are enabled by every authorization-step. These permissions comprise the *enabled-permissions* set. Collectively, we refer to the union of the executor-permissions and enabled-permissions as the *protection-state* of the authorization-step. Finally, the authority granted by a signature is good only for a limited period of time. Similarly, we associate a period of validity and a lifecycle with every authorization-step.

Classical subject-object access control $P \subseteq S \times O \times A$

TBAC view of access control $P \subseteq S \times O \times A \underbrace{\times\ U \times AS}_{\text{TBAC extensions}}$

Figure 2. Subject-object Versus TBAC views of access control

From the standpoint of access control models, Figure 2 illustrates how the TBAC view of access control differs from classical subject-object access controls. In the latter, a unit of access control or permission information can be seen as an element of the cross product of three domains (sets), namely the set of subjects, S, the set of objects, O, and the set of actions, A. In TBAC, access control involves information about two additional domains, namely, usage and validity counts, U, and authorization-steps, AS. These additional domains embed task-based contextual information.

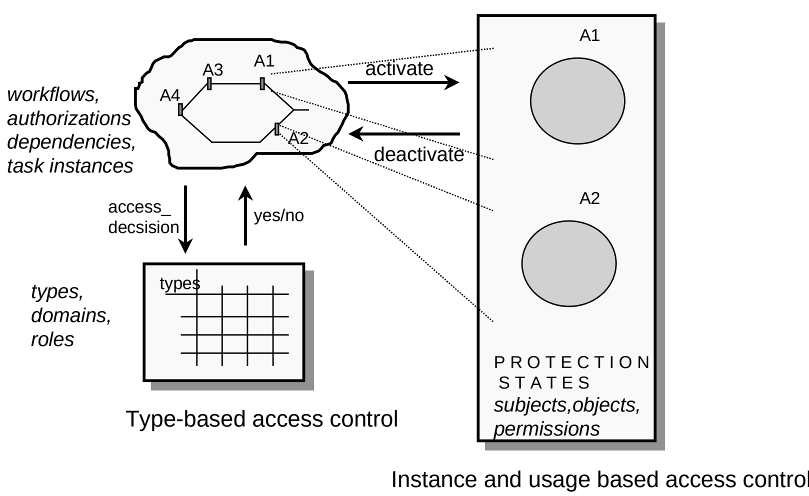

Figure 3. TBAC as an active security model

Figure 3 shows the concepts, features, and components that make TBAC an active security model. These include the following:

- the modeling of authorizations in tasks and workflows as well as the monitoring and management of authorization processing and life-cycles as tasks progress;
- the use of both type-based as well as instance and usage-based access control;
- the maintenance of separate protection states for each authorization-step;
- the dynamic runtime check-in and check-out of permissions from protection states as authorization-steps are processed.

Every authorization-step maintains its own protection state. The initial value of a protection state is the set of permissions that are turned on (active) as a result of the authorization-step becoming valid. However, the contents of this set will keep changing as an authorization-step is processed and the relevant permissions are consumed. With each permission we associate a certain usage count. When a usage count has reached its limit, the associated permission is deactivated and the corresponding action is no longer allowed. Conceptually, we can think of an active permission as a check-in of the permission to the protection-state and a deactivation of a permission as a check out from the protection state. This constant and automated check-in and checkout of permissions as authorizations are being processed is one of the central features that make TBAC an active model. Further, the protection states of individual authorization-steps are unique and disjoint. What this means is that every permission in a protection state is uniquely mapped to an authorization-step instance and to the task or sub-task instance that is invoking the authorization. This ability to associate contextual information with permissions is absent in typical subject-object style access control models.

The distinction between type-based and instance and usage-based access control is also a significant feature of the TBAC model. Type-based access control is used to encapsulate access control restrictions as reflected by broad policy and applied to types. Instance and usage-based access control on the other hand, is used to model and manage the details of access control and protection states (permissions) of individual authorization instances including keeping track of the usage of permissions.

In summary, TBAC differs from traditional passive subject-object models in many respects by associating the dimension of tasks with access control. First, there is a notion of protection states, which represent active permissions that are maintained for each authorization step. The protection state of each authorization step is unique and disjoint from the protection states of other steps. Each authorization-step corresponds to some activity or task within the broader context of a workflow. Traditional subject-object models have no notion of access control for processes or tasks. Second, TBAC recognizes the notion of a life-cycle and associated processing steps for authorizations. Third, TBAC dynamically manages permissions as authorizations progress to completion. This again differs from subject-object models where the primitive units of access control information contain no context or application logic. Also, TBAC understands the notion of "usage" associated with permissions. Thus an active permission resulting from an authorization does not imply a license for an unlimited number of accesses with that permission. Rather, authorizations have strict usage, validity, and expiration characteristics that may be tracked at runtime. In a typical subject-object access control model, a permission associated with a subject-object pair implies nothing more than the fact that the subject has the permission for the object. There is no recognition or monitoring of the usage of that permission. Finally, TBAC can form the basis of self-administering security models as security administration can be coupled and automated with task activation and termination events.

3 A FAMILY OF TBAC MODELS

Rather than formulating one simple monolithic model of TBAC we have chosen to formulate a family of models. Before discussing the models, we first lay out a framework to guide us in designing the family of models.

3.1 Framework

Our framework consists of formulating a simple model of TBAC called $TBAC_0$ and using this as a basis to build other models.

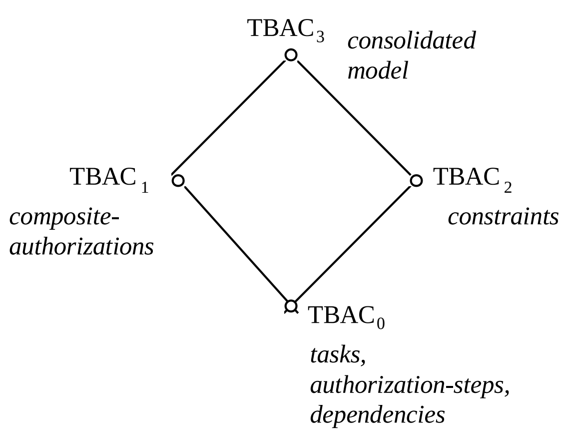

Figure 4. A framework for a hierarchy of TBAC models

Figure 4 shows our framework. $TBAC_0$ is a base model and is thus at the bottom of the lattice. It provides some basic facilities to model tasks, authorization-steps, and dependencies relating various authorization-steps. $TBAC_0$ is a very general and flexible model and is thus the minimum requirement for any system incorporating task-based authorizations. The advanced models $TBAC_1$ and $TBAC_2$ include (inherit) $TBAC_0$ but add more features. $TBAC_1$ incorporates the notion of composite authorizations (discussed shortly) whereas $TBAC_2$ adds constraints. Finally, $TBAC_3$ is the consolidated model that includes $TBAC_1$ and $TBAC_2$ and by transitivity $TBAC_0$.

Formulating such a family of models has many benefits. Researchers and developers can compare their system implementation of TBAC concepts with this family of models. Also, a family of models gives developers various choices in choosing conformance points for their implementations and can thus serve as a guide and evolution path for additional features.

3.2 The model $TBAC_0$

We will now describe the model $TBAC_0$ in more detail. We describe the various attributes or components that make up every authorization-step, followed by its life-cycle, and lastly the dependencies that are used to model authorization policies.

3.2.1 Components of an authorization-step

Every authorization-step has to specify a variety of attributes. We now describe briefly each of these attributes (components) in turn.

- *Step-name*: this is the name of the authorization-step.
- *Processing-state*: The current processing state indicates how far the authorization-step has progressed in its life-cycle (discussed shortly).
- *Protection-state*: The protection-state defines all potential active permissions that can be checked-in by the authorization-step. The current value of the protection-state, at any given time, gives a snapshot of the active permissions at the time. Associated with every permission is a validity-and-usage specification. The validity-and-usage-specification specifies the validity and usage aspects of the permissions associated with an authorization-step. It will thus specify how the usage of the permissions will relate to the authorization remaining valid (or becoming invalid).
- *Trustee-set*: This contains relevant information about the set of trustees that can potentially grant/invoke the authorization-step such as their user-identities and roles.
- *Executor-trustee*: This records the member of the trustee-set that eventually grants the authorization-step.
- *Task-handle*: This stores relevant information such as the task and the event identifiers of the task from which the authorization-step is invoked.

3.2.2 Processing states and life-cycle of authorizations

As mentioned earlier, an authorization is not static; rather it has a lifetime and a life-cycle associated with it. In order to better understand the execution aspects of authorizations, it is useful to consider the various processing states that every instance of an authorization-step goes through during its life-cycle.

A simple view of this life-cycle is to consider every authorization-step instance as going through five states, namely dormant, invoked, valid, invalid, and hold, as shown in Figure 5. An authorization is dormant when it has not been invoked (requested) by any task. Once invoked, an authorization-step comes into existence, and will be processed. If this processing is successful, the authorization-step enters the valid state. Otherwise, it becomes invalid. In the valid state, all associated permissions with the authorization are turned on (activated) and thus available for consumption. From the valid state, an authorization-step will undergo further processing and eventually reach the end of its lifetime and enter the invalid state. Also, a valid authorization-step may be put on hold temporarily. When this happens, all permissions associated with the authorization-step are inactive and cannot be used to gain any access until this hold is released and the validity reinstated. Eventually, when an authorization becomes invalid, it ceases to exist, and is deleted from the system.

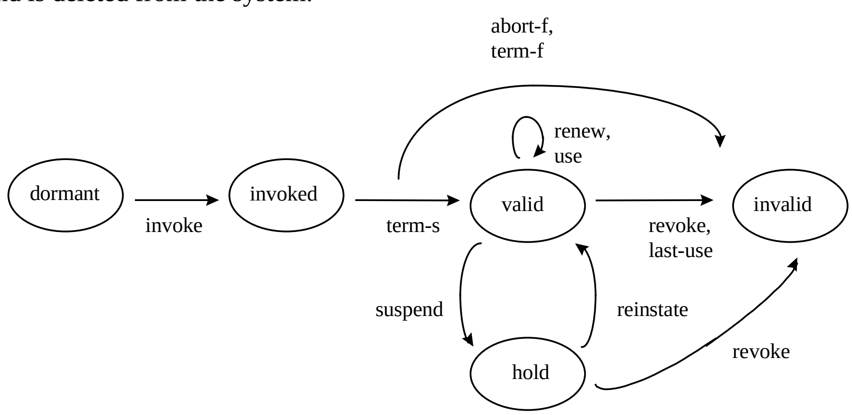

Figure 5. Basic processing states for an authorization-step

However, to get a more detailed description of what happens to an authorization during its lifetime, one can derive a more elaborate state diagram such as that shown in

Figure 6. This more elaborate state diagram recognizes the dimension of usage of permissions. A permission that is in the protection state of an authorization-step is consumed if any action that is enabled by the authorization-step requires the permission. Every action or request thus decrements the usage count of the

permission. Once the usage limit is reached an action will no longer succeed as TBAC ensures that the required permission is no longer available.

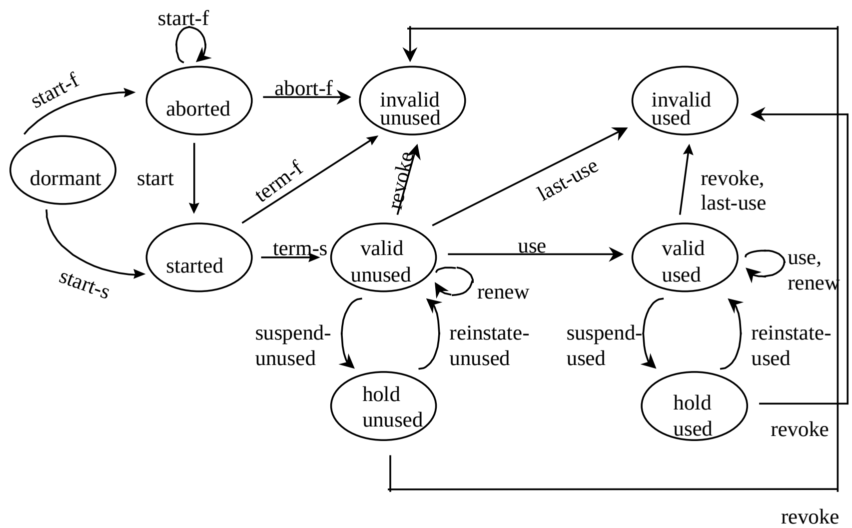

Figure 6. Detailed processing states of an authorization-step

Figure 6 is a direct refinement of Figure 5. The aborted and started states of
Figure 6 are a refinement of the invoked state of Figure 5. Similarly, the valid, hold and invalid states of Figure 5 are each refined into a pair of corresponding used and unused states in
Figure 6.
We describe each of the processing states below.

- ***Dormant***: An authorization-step is in this state if it has not been invoked by any task. Equivalently, the dormant state can be viewed as one where the authorization-step does not as yet exist. In particular, the protection state of the authorization-step is empty.
- ***Started***: Once an authorization-step has been successfully invoked, it enters this state where processing begins.
- ***Aborted***: The aborted state is in many ways similar to dormant except that a failed attempt to start the authorization-step was made in this case.
- ***Valid-unused***: Once an authorization-step has been started subsequent successful processing will transition it into the valid-unused state.
- ***Valid-used***: If an authorization was in a valid-unused state, and it is subsequently used or consumed, then it enters the valid-used state. Depending on policy, an authorization may be used multiple times before it enters the invalid state.

- ***Invalid-unused***: This state is entered if certain conditions for an authorization to be valid are not met upon termination or if the authorization had entered the valid-unused state and was subsequently revoked.
- ***Invalid-used***: This state is entered either as a result of a last-use transition from the valid-unused state or as a result of a revoke or last-use event (transition) from the valid-used state.
- ***Hold-unused:*** In this state the unused authorization is temporarily suspended. All associated permissions will thus be inactive.
- ***Hold-used:*** The authorization is temporarily suspended. All associated permissions will thus be inactive.

We now informally state some properties about authorization-steps.
Property 1. ***Executor Assignment***. For every authorization-step, as, the executor-trustee (ET) component is null until *as* transitions into the "started" state.
Property 2. ***Non-replaceable Executor.*** Once an executor trustee is assigned to an authorization-step, it is fixed for the entire lifetime of the step.
Property 3. Disjoint Protection States. The protection states associated with various authorization-steps are disjoint. Thus every authorization-step instance has a unique protection state. Thus given a set of authorization-steps a_1, a_2, …a_k, and their respective protection states, p_1, p_2, …p_k, the intersection of two or more of these states will be empty.

3.2.3 Basic dependencies to construct authorization policies

In the previous sections, we discussed authorization-steps. However, in any application or workflow logic, authorization steps do not stand in isolation. Rather, they are often related and dependent on each other due to policy implications. We now discuss various dependencies and constructs that relate authorization-steps to each other and constrain their execution and behavior. These dependencies can thus be used to formulate enterprise-oriented authorization policies.

We specify dependencies in terms of existential, temporal, and concurrency relationships that hold between events (or states resulting from the occurrence of events).

We list the dependency types and their meanings (interpretations) below.

1. $A1^{\textbf{\textit{state1}}} \rightarrow A2^{\textbf{\textit{state2}}}$: if A1 transitions into state1, then A2 **must** also transition into state2.
2. $A1^{\textbf{\textit{state1}}} < A2^{\textbf{\textit{state2}}}$: if both A1 and A2 transition into states state1 and state2 respectively, then A1's transition must occur **before** A2's

3. $A1^{\textit{\textbf{state1}}}$ # $A2^{\textit{\textbf{state2}}}$: A1 **cannot** be in state 1 concurrently when A2 is in state2.

4. $A1^{\textit{\textbf{state1}}}$ ||| $A2^{\textit{\textbf{state2}}}$: A1 **must** concurrently be in state1 when A2 is in state2.

The first two dependency types $\rightarrow$ and < express existential and temporal predicates and as such are best interpreted as predicates between transition events that lead to changes in the processing states of authorization-steps. They were originally proposed by Klein in (Klein, 1991) to capture the semantics of database transaction protocols. The other dependencies express concurrency properties.

Having discussed $TBAC_0$, we now highlight the ideas in the models $TBAC_1$ and $TBAC_2$. Due to space constraints, our discussion here is brief.

3.3 The model $TBAC_1$ to support composite authorizations

The model $TBAC_1$ supports the notion of composite authorizations. A *composite authorization* is an abstraction that encapsulates two or more authorization-steps. This is convenient when an authorization-step is too fine-grained a unit to express authorization requirements at a high (abstract) level.

For example, consider the authorization to transfer funds from one bank account to another. Such an action typically requires two authorizations. The first authorization is for withdrawal of funds from the source account and the second to deposit funds into the target account. However, it is useful for modeling purposes to think of a more composite abstraction called "authorize-transfer" that consists of the individual authorization-steps.

Thus a composite-authorization consists of a set of component authorization-steps. These component authorization-steps can be related to other steps within the *same* composite-authorization through various dependencies. In other words, the authorization-steps of a composite-authorization are not visible externally to other authorization-steps outside the composite-authorization. The motivation for this restriction comes from a desire to follow sound software-engineering principles, especially those related to encapsulation and information hiding. Thus to the external world, a composite-authorization is a single abstraction.

Collectively, the above properties and restrictions impose different semantics during the lifetime of a composite-authorization. In particular, we have to reexamine the notions of when we consider a composite-authorization to be started, valid, and invalid. We approach these issues by associating a *critical-set* of component authorization-steps with every composite-authorization. The critical-set is a subset of the total number of component authorization-steps. We consider a composite-authorization to have started when any member of the critical-set has reached the started state. To be considered valid, all steps in the critical-set have to reach their respective valid states. On the other hand, a composite-authorization is considered invalid as soon as any step in the critical-set becomes invalid.

In addition to the validity associated with the critical-set, a composite-authorization may declare other non-critical-sets of authorization-steps to capture additional states of validity. However, these other sets can become valid only when the critical-set itself is valid and can remain valid only as long as the critical-set remains valid. Collectively, the critical-set along with the various non-critical sets, define progressive states (checkpoints) of validity. The specification of a critical-set within a composite-authorization should thus be done with careful thought given to some minimal notion of validity that ensures consistency with authorization policies for the enterprise.

3.4 The model $TBAC_2$ and constraints

As mentioned earlier, $TBAC_2$ supports more advanced notions of constraints. Thus $TBAC_2$ would be more suitable for an organization that finds $TBAC_0$ to be too open-ended or not having tight enough controls.

We classify constraints as static or dynamic constraints. Static constraints are those that can be defined and enforced when authorization-steps are specified. Dynamic constraints on the other hand, are those that can be evaluated only at runtime as authorization-steps are being processed.

In $TBAC_2$ the basic structure of an authorization-step has two components in addition to $TBAC_o$. We describe these below.

- Start-condition (SC). This component can be used to specify a rich set of constraints that govern whether an authorization-step can transition into the started state.
- Scope (SP). This component controls the visibility of an authorization-step with respect to other authorization-steps when formulating and enforcing authorization policies. Thus scope can be used to control if an authorization-step is visible to an entire workflow, a task, or other finer units such as sub-tasks.

We are currently investigating other static constraints for authorization-steps such as those on the processing states, protection states, trustee-sets, and executor permissions.

The most obvious examples of dynamic constraints are those involving dynamic separation of duties/roles and coincidence of roles. By keeping track of the executor trustees of invoked authorizations and combining the notions of dependencies and scope, the $TBAC_2$ model can be used to provide a much more powerful and general approach to specifying separation of duties requirements than transaction control expressions (proposed in (Sandhu, 1988)). Further, by utilizing the notion of scope, TBAC can specify that a dynamic separation of duties requirements hold across the scope of a sub-task, task, or other coarser units such as an entire workflow.

4 CONCLUSIONS AND SUMMARY

We have described an active approach and a family of models for authorization management, collectively called task-based authorization controls (TBAC). Our approach differs from passive subject-object models in many respects. Permissions are controlled and managed in such a way that they are turned-on only in a just-in-time fashion and synchronized with the processing of authorizations in progressing tasks. An authorization-step is a fundamental abstraction in TBAC and is used to group and manage a set of related permissions. To enable this, TBAC supports the notion of a lifecycle for an authorization-step. Further, TBAC keeps track of the usage and consumption of permissions, thereby preventing the abuse of permissions through unnecessarily and malicious operations. TBAC provides for the modeling of enterprise-oriented authorization policies using dependencies that relate authorizations according to some enterprise policy. To demonstrate our ideas we are currently building a prototype which will be reported in future publications.

We are currently investigating several issues. The consolidated model $TBAC_3$ needs further examination. In particular, the interaction of composite-authorizations from $TBAC_1$ and constraints from $TBAC_2$ requires further study. We are also looking at formulating higher level modeling constructs for authorizations that can be composed from the four types of dependencies mentioned in the paper. For example, it might be useful to have a construct to express atomicity semantics on the validity of a set of authorizations. Also of interest is a framework to cohesively model and understand various constraints. From the standpoint of building end user tools, we are collaborating with the Universities of Pittsburgh and Salerno and exploring various aspects of visual languages (Chang, 1991) including various visual metaphors and related policy grammars to express authorization policies. The mapping of policy sentences to dependencies and various security rules that will be automatically incorporated into workflow task definitions is also under investigation. Also under investigation are issues related to the delegation and revocation of authorizations and their related permissions.

Acknowledgment

This research was partly funded under DARPA contract F30602-95-C-0285. We are grateful to Teresa Lunt and Gary Koob for their support and encouragement.

5 REFERENCES

Abrams, M., Eggers, K., LaPadula, L., and Olson, I. A Generalized Framework for Access Control: An Informal Description, Proceedings of the 13th NIST-NCSC National Computer Security framework, 1990, pages 135–143.

Abrams, M., Heaney, J., King, O., LaPadula, L., Lazear, M., and Olson, I. Generalized Framework for Access Control: Toward prototyping the Orgcon Policy, Proceedings of the 14th NIST-NCSC National Computer Security framework, 1991, pages 257–266.

Atluri V., and Huang, W. An Authorization Model for Workflows, Proceedings of the Fourth European Symposium on Research in Computer Security, Rome, Italy, September pages 25–27, 1996.

Bell, D.E. and LaPadula, L.J. Secure Computer Systems: Unified exposition and multics interpretation. EDS-TR-75–306, Mitre Corporation, Bedford, MA, March 1976.

Chang. S.K. et. al. Visual-Language System for User Interfaces, IEEE Software, March, 1995.

Georgakopoulos, D., Hornick, M. and Sheth, A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel databases, Vol. 3, pages 119–153, 1995.

Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. Protection in Operating Systems. Communications of the ACM, 19(8), pages 461–471, 1976.

Klein, J. Advanced Rule Driven Transaction Management. Proceedings of the IEEE Compcon Conference, 1991.

LaPadula, L.J. and Williams, J.G. Towards a Universal Integrity Model. Proceedings of the IEEE Computer Security Foundations Workshop, New Hampshire, IEEE Press, 1991.

Rusinkiewicz, M. and Sheth, A. Specification and Execution of Transactional Workflows, In Modern Database Systems: The Object Model, Interoperability, and beyond, W. Kim, Ed., Addison-Wesley / ACM Press, 1994.

Sandhu, R.S. Transaction Control Expressions for Separation of Duties, Proceedings of the Fourth Computer Security Applications Conference, pages 282–286, 1988.

Sandhu, R.S. The Typed Access Control Model, Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1992, pages 122–136.

Thomas, R.K. and Sandhu, R.S. Towards a Task-based Paradigm for Flexible and Adaptable Access Control in Distributed Applications. Proceedings of the Second New Security Paradigms Workshop, Little Compton, Rhode Island, IEEE Press, 1993.

Thomas, R.K. and Sandhu, R.S. Conceptual Foundations for A Model of Task-based Authorizations. Proceedings of the IEEE Computer Security Foundations Workshop, New Hampshire, IEEE Press, 1994.