Access Control Using Active Rules

Raman Adaikkalavan\textsuperscript{1,*} and Sharma Chakravarthy\textsuperscript{2}

\textsuperscript{1} Computer and Information Sciences & Informatics, Indiana University South Bend
raman@cs.iusb.edu
\textsuperscript{2} Department of Computer Science Engineering, The University of Texas At Arlington
sharma@cse.uta.edu

Abstract. Access to only authorized resources is provided by access control mechanisms. Active or Event-Condition-Action rules make the underlying systems and applications active by detecting and reacting to changes. In this paper, we show how active rules can be used to enforce Role-Based Access Control (RBAC) standard. First, we analyze different components of active rules and their mappings for enforcing RBAC standard. Second, we discuss how RBAC standard is enforced using active rules. Finally, we discuss how active rules extend RBAC standard to cater to a large class of applications.

Keywords: ECA Rules, Role-Based Access Control, Event Constraints.

1 Introduction

Role-Based Access Control (RBAC), where object accesses are controlled by roles (or job functions) in an enterprise rather than a user or group, has shown to be a positive alternative to traditional access control mechanisms. RBAC does not provide a complete solution for all access control issues, but with its rich specification \cite{1}, it has shown to be cost effective \cite{2} and is used in several domains \cite{3}. ANSI RBAC Standard \cite{4} has four functional components: Core, Hierarchical, Static Separation of Duty and Dynamic Separation of Duty. Existing enforcement techniques are tied to a component, tightly integrated with the underlying system or they just support extensions. Constraints \cite{5,6} are critical in realizing RBAC over diverse domains, as they provide the flexibility for specifying fine-grained access control policies.

There is consensus in database and other communities on Active or Event-Condition-Action rules as being one of the most general formats for expressing rules in an event-driven application. Active rules and event processing systems \cite{7-18} have been employed successfully in diverse application domains for situation or change monitoring. Existing event specification languages and event detection systems provide well-defined point-based, interval-based and generalized event semantics.

In this paper, we show how the ANSI RBAC standard can be enforced via active rules using a layered approach. We also show how enterprises can move from one RBAC functional component to the other in a seamless manner. In other words, enterprises

\textsuperscript{*} This work was supported, in part by, IUSB Faculty Research Grant.
can move from Core RBAC to Hierarchical RBAC without rewriting events and rules. We then extend the RBAC specification with event-based constraints. We discuss the placement of simple and complex constraints at various granularities. We also show that these event constraints are not affected when the enterprises move from one RBAC component to the other. Finally, we discuss event detection graphs to implement events and rules.

Outline: Events and rules are discussed in Section 2. RBAC is discussed in Section 3. Enforcement using active rules is discussed in Section 4. Event constraints and advantages of our approach are discussed in Section 5. Event detection graph is discussed in Section 6. Section 7 has conclusions.

2 Active Rules

Below, we discuss events and active rules briefly.

2.1 Events

Snoop [9, 10, 14], event specification language, is used in this paper. The main motivation to use Snoop is that it supports expressive event specification using point-based [14], interval-based [10] and generalized semantics [7] in various event consumption modes [10, 14] and event detection modes [7]. LED uses event detection graphs to detect events using point-, interval-based and generalized semantics in various consumption and detection modes.

An event is “an occurrence of interest” in the real world that can be either simple (e.g., depositing cash) or complex (e.g., depositing cash, followed by withdrawal of cash). Simple events occur at a point in time (i.e., time of depositing), and complex events occur over an interval (i.e., starts at the time cash is deposited and ends when cash is withdrawn). Simple events are detected at a time point, whereas complex events can be detected either at the end of the interval (i.e., point-based semantics) [15, 16] or can be detected over the interval (i.e., interval-based semantics) [10, 11, 18]. Each event has a well-defined set of attributes based on the implicit and explicit parameters [9]. Implicit parameters contain system- and user-defined attributes such as event name and time of occurrence. Explicit parameters are collected from the event itself (e.g., stock price and stock value).

Below, we discuss event specification using the interval-based semantics, represented using $\mathcal{I}$. For more details about event semantics, please refer to [9, 10, 14].

Simple Events: Simple events are the basic building blocks in an event processing system and are derived from various application domains. E.g., data manipulation language and data definition language statements in a DBMS, function call invocation in Object-oriented systems, alarm clock, and increase in stock price.

Definition 1. An interval-based simple event $E$ occurs atomically at a point $[t]$ on the time line. It is detected over an interval $[t, t']$, where $[t]$ is the start time, $[t']$ is the end time and ($t = t'$) $^1$ It is defined as $\mathcal{I}(E, [t, t']) \triangleq \exists t = t' \ (E, [t, t'])$;