Perti Net-Based Workflow Access Control Model

CHEN Zhuo(陈卓) 1, 2,  LUO Ting(骆婷) 1,  SHI Lei(石磊) 1,  HONG Fan(洪帆) 1
1. Laboratory of Information Security, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P. R. China
2. College of Information Engineering Hubei Polytechnic University, Wuhan 430068, P. R. China

Abstract  Access control is an important protection mechanism for information systems. This paper shows how to make access control in workflow system. We give a workflow access control model (WACM) based on several current access control models. The model supports roles assignment and dynamic authorization. The paper defines the workflow using Petri net. It firstly gives the definition and description of the workflow, and then analyzes the architecture of the workflow access control model (WACM). Finally, an example of an e-commerce workflow access control model is discussed in detail.

Key words  workflow, task, access control, authorization, Petri net, access control matrix, e-commerce, SET protocol.

1 Introduction

Today, numerous researches and applications of workflow can be found in many domains. However, current workflow deployment practically focuses on workflow resources management. Many systems ignore the security issue. Taking into account the following aspects in security services: identification/authentication, authorization, confidentiality, integrity and non-denial/non-repudiation, access control is a specific mechanism providing authorization. The proposed mechanism focuses on confidentiality, but integrity of the data items and non-repudiation is also supported because only authorized subjects are allowed to manipulate the data in the system. Furthermore, the access control mechanism realizes the need-to-know-paradigm because subjects will only be the execution of an activity in the workflow. In this paper, we study the workflow authorization problem using access control mechanism. Several main models to effect access control administration are role-based access control (RBAC) [4, 5], workflow authorization model (WAM) [11, 10] and task-based authorization control (TBAC) [12]. WAM was presented by Atluri in 1996. In WAM, various tasks in a workflow are executed by several users or programs according to the organizational rules relevant to the processes represented by the workflow. Every task is defined in a time interval which controls the interval of the authorization. But the lower and the upper bounds of the time interval are fixed on difficulty in advance in actual workflow environment. In addition, WAM is a primitive model and is still cannot applicable in practice. TBAC models access controls from a task-oriented perspective than the traditional subject-object one. Access mediation 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.

In this paper, we develop a new workflow access control model (WACM) based on the above access control models. The main features of the model are: (1) synchronous authorization, and (2) dynamic role-based authorization. This paper uses Petri nets for the specification of workflows. Petri nets offer a solid mathematical basis and are well suited to represent discrete dynamic models like workflows. In this paper, a workflow specified by a Petri net will be called Petri net-based workflow.

The remainder of this paper is organized as follows. In Section 2, we discuss the definition and de-
scription of the workflow model. In Section 3, we study the architecture of WACM. In Section 4, we show an example of WACM in e-commerce workflow.

2 Workflow Model

2.1 What is workflow

Let \( S = \{s_1, s_2, \ldots\} \) denote the set of subjects, \( O = \{o_1, o_2, \ldots\} \) the set of objects, \( \Gamma = \{\gamma_1, \gamma_2, \ldots\} \) the set of object types, and \( R = \{r_1, r_2, \ldots\} \) the set of roles.

**Definition 1** Function \( F \) can be defined as: \( O \rightarrow \Gamma \). \( F(o_i) = \gamma_j \) means that \( o_i \) is object type \( \gamma_j \).

**Definition 2** A workflow \( WF \) can be defined as a directed graph whose nodes are the tasks \( t_1, t_2, \ldots, t_n \) in the workflow and edges are the task dependencies \( t_i \rightarrow t_j \), where \( t_i, t_j \in WF \) and \( \chi \) denotes the type of dependency.

Because of the large variety of coordination requirements that a workflow may need to support, various kinds of tasks dependencies have been proposed. Three basic types of task dependencies are identified: control-flow dependencies, value dependencies and external dependencies \(^{[10]}\).

**Definition 3** A task \( t_i \) is defined as \((OP_i, \Gamma_{IN_i}, \Gamma_{OUT_i})\), where \( OP_i \) is the set of operations to be performed in \( t_i \), \( \Gamma_{IN_i} \subseteq \Gamma \) is the set of object types allowed as inputs, \( \Gamma_{OUT_i} \subseteq \Gamma \) is the set of object types expected as outputs.

**Definition 4** A task-instance \( t_{\text{inst}_i} \) is defined as: \((\text{OPER}_{t_i}, \text{IN}_{t_i}, \text{OUT}_{t_i})\) where \( \text{OPER}_{t_i} \) is the set of operations performed during the execution of \( t_i \), \( \text{IN}_{t_i} \) is the set of input objects to \( t_i \) such that \( \text{IN}_{t_i} = \{x \in O | F(x) \in \Gamma_{IN_i}\} \), \( \text{OUT}_{t_i} \) is the set of output objects from \( t_i \) such that \( \text{OUT}_{t_i} = \{x \in O | F(x) \in \Gamma_{OUT_i}\} \). Whenever a task is executed a task-instance will be generated.

2.2 Petri net workflow

Petri nets \(^{[6,11]}\) have been studied intensively in the community of computer science. A Petri net \( N \) is a triple \( N = (P, T, F) \). \( P = \{p_1, p_2, p_3, \ldots, p_n\} \) is the finite set of the places. \( T = \{t_1, t_2, t_3, \ldots, t_n\} \) is the finite set of transition with \( P \cap T = \Phi \). The flow relation \( F \) is defined by \( F \subseteq (P \times T) \cup (T \times P) \).

Let \( y \in P \cup T \), \( \cdot y \) is called the preset of \( y \) and is defined by

\[ y^\cdot = \{x \in P \cup T | (x, y) \in F\} \]

\( y \cdot \) is called the postset of \( y \) and is defined by

\[ y^\cdot = \{x \in P \cup T | (y, x) \in F\} \]

Fig. 1 shows an example of a Petri net. The net consists of the places \( p_1, \ldots, p_4 \) and the transition \( t_1, \ldots, t_5 \). The sets of \( P, T \), and \( F \) are defined as follows:

\[ P = \{p_1, p_2, p_3, p_4, p_5\} \]
\[ T = \{t_1, t_2, t_3, t_4\} \]
\[ F = \{(p_1, t_1), (t_1, p_2), (p_2, t_2), (t_2, p_3), (t_2, p_4), (t_4, p_5)\} \]

![Fig. 1 Example of a Petri net](image-url)

The graphical interpretation of a Petri net is a bipartite graph. Places can only be connected to transitions, and transitions can only be connected to places. Places are represented graphically as circles \( \bigcirc \), transitions as rectangles \( \blacksquare \). The graphical interpretation of an element \((x, y) \in F\) is an arrow from \( x \) to \( y \).

The preset and postset of a transition is a set of places. This set can be empty. The preset and postset of a place is a set of transitions. This set can be empty too. Presets and postsets from the example: \( \cdot p_1 = \Phi \), \( \cdot t_1 = \{p_1\} \) and \( t_1^\cdot = \{p_2\} \).

**Definition 5** A non empty set \( M \subseteq P \) is called a marking of a Petri net. A transition \( t \) is called activated under marking \( M \), if

1. \( t \subseteq M \) and
2. \( M \cap t^\cdot = \Phi \).

An activated transition can fire. The first marking of a Petri is called Start marking. Graphically a marking is represented by filled circles in all its places. These filled circles are also called tokens.

The behavior of Petri nets shall be illustrated following the example shown in Fig. 1. Let the start marking be \( \{p_1\} \). A token is put on place \( p_1 \). The transition \( t_1 \) is activated. If \( t_1 \) fires, the token moves from \( p_1 \) to \( p_2 \). The transition \( t_2 \) is activated now. If the token moves to \( p_4 \), the transition \( t_4 \) fires. If the token moves to \( p_3 \), the transition \( t_3 \) fires.

Therefore, with start marking \( \{p_1\} \) there are two possibilities for the Petri net to execute:

\[ p_1 \xrightarrow{t_1} p_2 \xrightarrow{t_2} p_3 \xrightarrow{t_3} p_5 \text{ and } p_1 \xrightarrow{t_1} p_2 \xrightarrow{t_2} p_4 \xrightarrow{t_4} p_5 \]