A Usage Control Policy Specification with Petri Nets

Basel Katt and Michael Hafner
University of Innsbruck
Innsbruck, Austria
{basel.katt, m.hafner}@uibk.ac.at

Xinwen Zhang
Samsung Information Systems, America
San Jose, CA, USA
xinwen.z@samsung.com

Abstract

In this paper we propose a novel usage control policy specification based on Coloured Petri Nets formalism. Recently, usage control has been proposed in order to overcome the shortcomings of transitional access control that fails to meet new security requirements of today's highly dynamic and distributed systems. These new environments require for example (i) a continuity of control, (ii) fulfillment checks of obligatory tasks, during or after the usage end, (iii) an integration between functional behavior and security policy, and (iv) the management and control of concurrent and parallel usages by different subjects. Taking all these requirements into consideration, our usage control policy includes three main rule types: behavioral, security and concurrency rules. Security rules, can be further classified either into instant-, -ongoing, and post rules or into authorization and obligation rules. Instant rules must be checked before the execution of an action is granted, ongoing rules are checked during the execution of an action, and finally post rules are checked after the execution is finished. Therefore, post rules are only of type obligation. Coloured Petri nets are used because of their powerful modeling capabilities of distributed and concurrent systems and their efficiency for specification of systems by embodying the support of ML functional programming language.

1. Introduction

Recently, the need for collaboration and secure resource sharing between companies, institutions and/or citizens has increased in different application domains, including distributed eHealth networks, distributed DRM system, social networks, merging, outsourcing and offshoring environments, and others. These dynamic and distributed environments raise new questions that need to be considered and answered by security solutions. For example, behavioral rules and restrictions, obligatory tasks that need to be fulfilled by certain subjects, rules that must be fulfilled after resources are accessed (post rules) and their associated compensation actions, and finally, concurrency control that has been always considered from a functional perspective. While access control aims at restricting access to resources instantly, it fails to meet these new requirements.

Usage control has been proposed to overcome the drawbacks of access control. Usage control goes beyond the capabilities of access control to cover continuity of control, integration of security and functional aspects, obligations and post usage rules, and finally concurrency management. From the discussion above, we define *usage control* as *the ability to continuously control the usage of shared resources in a distributed environment. Continuously* means that the control must be ongoing and continuous for the whole period of a resource usage. *Control* means that permissions must be checked, obligatory tasks must be enforced, and environmental conditions must be considered. Furthermore, control can be applied before, during, or even after the resource usage. *Shared* indicates that resources can be accessed by different users at the same time. Finally, *distributed environment* allows the assumption that resources can be located on clients and/or servers.

The main approaches in the literature dealing with usage control are UCON model [13] and its specifications like in [17], [6], [11] and OSL [4]. UCON is a general model to capture the major security requirements of usage control. The policy specifications of UCON recognize the importance of a continuous and ongoing control, provide (pre- and on-)authorization and obligation rules, and supports (attribute) mutability. However, the current UCON model and its policy specifications lack post-obligation and concurrency support. Later, the authors of [5] considered the concurrent usages. They propose a static analysis method by creating the dependency graph between different controllers/users, however fail to define concurrency rules for users within one controller. In previous work [9] we proposed an extension of UCON specification to enhance its obligation expressiveness, however our approach lacked the ability to express periodic obligations, application behavior was not considered, and it was done in an ad-hoc manner that makes the resulted policy difficult to analyze. OSL, on the other hand, proposes an expressive policy language for usage control considering temporal, cardinal and permit rules. However, the continuity aspect of control beyond one action, the integration with the application behavior, mutability, and concurrency issues tend to be overlooked.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8394
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8394

To overcome the drawbacks we identify in the current approaches, we propose in this paper a usage control policy specification that embodies the following features. First, we emphasize the concept that controlling the usage of a resource must be continuous and ongoing at the action level and during the entire usage period. This requires the support of behavioral rules and restrictions to identify how resources can be accessed during the usage session. The second issue is *concurrency* control. When dealing with *stateful* usage control policies and shared resources, concurrency becomes a relevant issue. In real-world distributed and collaborative environments, resources are shared. This fact requires placing constraints on how concurrent usages of resources must be regulated. Finally, our usage control policy considers authorization, including context and environmental conditions, and obligations. Our novel contribution in this regard is the specification of post obligation rules and their compensation actions. For example, students registering for an online course are obliged to pay certain amount of fee within six months, otherwise, the registration is canceled. Paying the tution fees is the obligation that must be fulfilled by the subject, and canceling the registration is the compensation action in case of a violation. In general, we call actions that a reference monitor can execute *enforcement actions*. These actions updating attribute update, fulfillment check, and compensation actions.

For the purpose of specifying the usage control policy we utilize Coloured Petri Nets. CPN embodies the Petri Nets' powerful modeling capabilities for concurrent system behaviors and the powerful specification support of the *ML* functional language. Furthermore, its mathematical foundation and the tool support enable and ease analysis and verification of the usage control policy. We define *Usage Control Coloured Petri Nets (UCPN)* that extends CPN with the concepts of *rule label function (R)* and *end arcs (EA)*. A rule label function associates each transition with appropreiate security and concurrency rules, which are defined based on ML functional programing language. End arcs, on the other hand, represent the behavior of a transition when its rules are violated.

Contributions: In this paper we propose a novel policy specification for usage control based on Colored Petri Nets. The main and unique features of our usage control policy are the support of (1) behavioral rules and restrictions (2) post obligation rules and their compensation actions, and (3) concurrency control for shared resources. Usage Control Coloured Petri Net (UCPN) is defined to specify our policy. It is a CPN augmented by a rule label function and end arcs. Behavioral rules are modeled as a CPN and security and concurrency rules are modeled as tuples using ML and mapped to different transitions of the behavioral rule's net using the rule label function.

Outline: In next section we give an overview of Coloured Petri Nets. Section 3 introduces our usage control policy and its specification. Related work is discussed in section 4, and finally, we conclude this paper and discuss the future work in Section 5.

2. Colored Petri Nets Overview

A Colored Petri Net (CP-net) [7] can be defined as a tuple *CPN = (Σ,P,T,A,N,C,G,E,I,SC)*, where Σ is a set of *color sets*, *P, T* and *A* are sets of *places/states, transitions* and *arcs*, respectively. A transition has incoming and outgoing arc(s). Incoming arcs indicate that a transition may remove one or more *tokens* from the corresponding input places while outgoing arcs indicate that the transition may add tokens to the output places. The exact number of tokens and their values are determined by the *arc expression*, defined by the function *E*. *N* is a *node* function that determines the source and destination of an arc. *C* is a *color* function that associates a color set *C(p)* or a type with each place *p*. *G* is a guard function that maps each transition *t* to a boolean expression *G(t)*. For a transition to be enabled, a *binding* of the variables that appear in the arc expressions must be found, and for this "binding element" the guard function must evaluate to true. This binding makes the arc expression of each input arc evaluates to a multi-set of token colors. *I* is an *initialization* function that maps each place to a multiset *I(p)*. The last element is *SC*, which is the *segmentation code* function of a transition, mapping a transition to a set of actions that are executed when the transition occurs.

A token element is a pair (p, c) such that $p \in P$ and $c \in C(p)$. For a color set $s \in \Sigma$, the base color sets of S are the color sets from which S was constructed using some structuring mechanisms such as cartesian products, records, or unions. The set of all token elements is denoted by TE. For $x, x_1, x_2 \in P \cup T$, $x\bullet = \{y \in P \cup T \mid \exists a \in A : N(a) = (x, y)\}$ is the *postset* of x; and $\bullet x = \{y \in P \cup T \mid \exists a \in A : N(a) = (y, x)\}$ is the *preset* of x. $A(x1, x2)$ is the set of arcs from x_1 to x_2, and the expression of (x_1, x_2) is $E(x_1, x_2) = \Sigma_{a \in A(x1,x2)} E(a)$.

$TRAN(cpn)$ is a function that maps each CPN net or subnet to a set of all transitions contained in that net. $A : X \rightarrow A_s$ maps each node, $x \in P \times T \cup T \times P$, to its surrounding arcs. $Var(t)$ is the set of variables of a transition t. $Type(v) \in \Sigma$ denotes the type of the variable v. A *binding element* (t, b) is a pair consisting of a transition t and a binding b of data values to its variables such that $G(t) < b >$ evaluates to `true`. $expr < b >$ in general denotes the value obtained by evaluating the expression *expr* in the binding *b*. By *B(t)* we denote the set of all bindings for a transition *t*. The Binding element is written in the form $(t, < v_1 = c_1, v_2 = c_2, ..., v_n = c_n >)$, where $v_1, v_2, ..., v_n \in Var(t)$ are the variables of t and $c_1, c_2, ..., c_n$ are the data values such that $c_i \in Type(v_i)$ for $1 \leq i \leq n$. For a binding element (t, b) and a variable v

of t, $b(v)$ denotes the value assigned to v in the binding b. $B(t)$ denotes the set of all binding elements is denoted BE.

$M(p)$ denotes the marking of a place p in the marking M. M_0 is the *initial marking*. If a binding element (t, b) is *enabled* in a marking M_1, denoted $M_1[(t, b)\rangle$, then (t, b) may *occur* in M_1 yielding some marking M_2. This is written as $M_1[(t, b)\rangle M_2$. Accordingly, a *finite occurrence sequence* is a sequence consisting of markings M_i and binding elements (t_i, b_i) denoted $M_1[(t_1, b_1)\rangle M_2 ... M_{n-1}[(t_{n-1}, b_{n-1})\rangle M_n$ and satisfying $M_i[(t_i, b_i)\rangle M_{i+1}$ for $1 \leq i < n$. M_1 is called *start marking*, M_{n+1} is called end marking, and n is called the length of the occurrence sequence. If the length is infinite we call the occurrence sequence *infinite occurrence sequence*. The set of all finite occurrence sequences is denoted by *OSF*, while the set of all infinite occurrence sequences is denoted by *OSI*, and finally $OS = OSF \cup OSI$ is the set of all occurrence sequences. $time : OSF \rightarrow N$: defines the duration of a finite occurrence sequence $\sigma \in OSF$. A *reachable marking* is a marking which can be obtained by an occurrence sequence starting in the initial marking. $[M_0\rangle$ denotes the set of reachable markings. Finally, The sets of all markings and steps is denoted by $\mathbb{M}$ and $\mathbb{Y}$, respectively [1].

Let $X \subseteq BE$ be a set of binding elements and $\sigma \in OSF$ is a finite occurrence sequence, we can also consider an infinite one, of the from: $\sigma = M_1[Y_1\rangle M_2 \ldots M_n[Y_n\rangle M_{n+1}$. For each $i \in N_+$, $EN_{X,i}(\sigma)$ is the number of elements from X which are enabled in the marking M_i and $OC_{X,i}(\sigma)$ is the number of elements from X which occur in the step Y_i. Furthermore, $EN_X(\sigma) = \sum_{i \in N_+} EN_{X,i}(\sigma)$ and $OC_X(\sigma) = \sum_{i \in N_+} OC_{X,i}(\sigma)$ are the total number of enabling and occurrences in σ, respectively.

3. Usage Control Policy Specification

3.1. Usage Control Policy Elements

Subjects: are active principals that can perform actions on resources, which are identified by an ID and a set of attributes. We define for this reason a new type (coloset) called *SUB*. A general definition of the SUB type is a product of an ID, of a type string, and a list of attributes. Each subject is represented as one token of type SUB. We assume also a *Subjects* place that contains all subject tokens in the system. Additionally, we define the type *TSUB* which represents subject tokens with time stamps, it is used for the temporal related mechanisms.

Objects: represent resources that must be protected. They are identified by a unique ID and attributes. We define a new colset called *OBJ*. Similar to subjects, OBJ is also defined as the product of ID and a set of attributes. Each resource in the system is represented as on token of type OBJ, furthermore, we assume the *Objects* place that contains all objects.

1. We only coonsider steps that consist of one binding elements.

Contexts: represent environmental entities that play a role in authorization decisions. For example, the entity that provides the IP address of an accessing machine is a context entity. We define a new coloset called *CON*. Each context entity in the system is represented as a token of a type CON. The *Contexts* place, on the other hand, contains tokens of all context entities.

Actions: Beside these basic elements, actions (ACT) are essential elements. Actions in a usage control system can be classified into the following types: (i) *subject actions* ($SAct \subseteq ACT$) are actions that a subject can execute/request on a resources and they are included in the system behavior. Subject actions are represented as transitions in the policy view. (ii) *enforcement actions* ($EAct \subseteq ACT$) are those that a reference monitor or an enfrcement engine can execute, like updating subject/object attributes or logging access information. Due to the fact enforcement mechanisms are modeled as a CP-net, these actions as represented as ML functions and positioned as segmentation code of a transition. Finally (iii) *obliged action* ($OAct \subseteq ACT$) are actions that a subject must execute (obligatory tasks). Each obliged action is represented by a unique ID and is used to define obligation rules.

3.2. Behavioral Rules

One of the main features of our usage control policy, which distinguishs it from traditional access control, is the continuity of control. This means that making authorization decisions, following obligatory tasks, executing compensation and enforcement actions, and managing concurrent and parallel usages must be done and controlled continuously during the time period, in which resources are being used. Therefore, our usage control policy is session based.

Definition 1 (Usage Session). *A usage session of a resource or a set of resources is defined as a finite occurrence sequence $\sigma = M_0[Y1...Yn\rangle M_n$, where M_0 is the initial marking of the usage and the M_n is the finial marking in which the usage is finished.*

Furthermore, to ensure a correct and safe behavior of the system, behavioral rules must be defined. These rules indicate how (security related) actions are composed and their relationship. For example an action must be executed after another action, or in parallel with other action(s). For this purpose, we utilize Coloured Petri Nets by defying a CP-net containing all actions and their relationships[2]. We call the resulting CP-net that embodies the behavioral rules and conditions a *Safe Behavior*.

2. Encoding the informal behavioral requirements in the form of CP-net requires a Petri Net modeling skills. The modeling issue itself is out of scope of this paper but is assumed to be able to define the behavioral rules' CP-net.

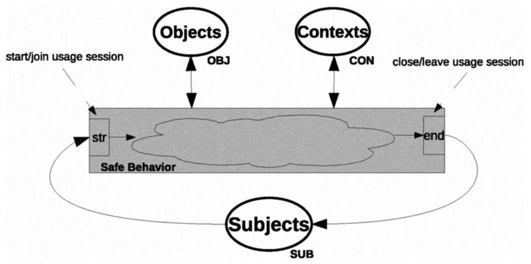

Figure 1: Usage Pattern.

Definition 2 (Safe Behavior). *A safe behavior is defined as a Coloured Petri Net that embodies the behavioral rules and conditions. Transitions in a safe behavior represent subject actions; and places, which are of SUB type, indicate the state of each subject using a resource. Finally, a safe behavior contains one source and one sink transitions, str and end, respectively, where* $\bullet str = \{\} \wedge end\bullet = \{\}$. *We denote the finite set of all safe behaviors with* $(SB \subseteq CPN)$.

str and end transition/actions are the transitions that are used by subjects to initiate/Join a usage session or end/leave a usage session. By considering a usage session from a global perspective, and augmenting the behavioral CPN with *Subjects*, *Objects*, and *Contexts* places, the concept of a *Usage Pattern* emerges.

Definition 3 (Usage Pattern). *A usage pattern is defined as a tuple* $UP = (sb, Pl) \subset UP$, *where* $sb \in SB$ *is a safe behavior that represents the behavioral rules.* Pl' *is a sets of pre-defined places representing the basic elements of a usage control system:* $Pl' = \{Subjects, Objects, Contexts\}$. *The basic elements' places are connected to the safe behavior according to the following rules:*

(i) $\forall t \in TRAN(sb) : \{Objects, Contexts\} \subset \bullet t \wedge \{Objects, Contexts\} \subset \bullet t$.

(ii) $\bullet str = \{Subjects\} \wedge end\bullet = \{Subjects\}$.

(iii) $\exists\ o \in OBJ, c \in CON, s \in SUB : [\forall a \in A(Contexts) \Rightarrow (E(a) = c)] \wedge [\forall a \in A(Objects) \Rightarrow (E(a) = o)] \wedge [\forall a \in A \setminus \{A(Objects) \cup A(Contexts)\} \Rightarrow (E(a) = s)]$.

Figure 1 shows how to construct the usage pattern from the safe behavior and the basic elements' places. $Objects$ and $Contexts$ places are connected to all transitions of the safe behavior by double headed arcs. $Subjects$ place is connected to the source and end transitions of the safe behavior. Finally, we define variables *s, o,* and *c*, of SUB, OBJ, and CON types, respectively. Variables o and c are used as an arc expression for all arcs that are connected to *Objects* and *Contexts* places, and the variable s is used as an arc expression for the rest of the arcs.

3.3. Authorization and Obligation Rules

An authorization defines the right to use a resource and an obligation rule determines the tasks/actions that a subject must perform when a resource is used. These rules can be furthere classified into instant, temporal and cardinal for auhtorizatin and instant, temporal, and periodic for obligations. Finally we can define a concurrecny rule that defines who and how many subjects are allowed to execute an action at the same time.

3.3.1. Single Instant Access Right.

Definition 4. *A single instant access right is defined as 7-tuple ir=(s,o,c,sa,ea1,ea2,authCheck)*$\in$ *IR. This right indicates that a subject* $s \in S$ *is allowed to perform an action* $sa \in SA$ *on an object* $o \in O$ *in the context defined by* $c \in C$, *where ea1 and ea2 are enforcement actions that must be executed by the reference monitor if the action is allowed or denied, respectively. The function* $authCheck : Attr(S) \times Attr(O) \times Attr(C) \rightarrow Bool$ *indicates an enforcement action (a predicate) that takes as parameters subject, object and context attributes and returns a Boolean value indicating whether the subject is authorized or not. It defines the constraints or conditions that must be fulfilled to authorize the subject.*

3.3.2. Single Temporal Access Right.

Definition 5. *A single temporal access right is defined as a 4-tuple tr=(s,sa,d,ea)* $\in$ *TR. It indicates the maximum time period,* $d \in N$, *allowed for executing a subject action, sa, by a subject s. An enforcement action* $ea \in EAct$ *must be executed in case the execution is revoked.*

3.3.3. Single Cardinal Access Right.

Definition 6. *A single cardinal access right is defined as a tuple cr=(CSet,t,n,ea)*$\in$ *CR. It indicates the maximum number of times,* $n \in N$, *allowed for executing* $t \in T$ *by subjects belonging to the cardinal set* $CSet \subseteq SUB$. *An enforcement action ea must be executed if the action is denied.*

3.3.4. Single Instant Obligation.

Definition 7. *A single instant obligation is defined as 6-tuple iobl=(s,sa,oa,ea1,ea2,oblCheck)* $\in$ *IO. It indicates that a subject* $s \in S$ *is allowed to perform an action* $sa \in SA$ *if the obliged action* $oa \in OA$ *has been fulfilled by s, where ea1 and ea2 are optional enforcement actions that must be executed by the reference monitor if the action is allowed or denied, respectively. The function* $oblCheck : S \times OA \rightarrow Bool$ *indicates an enforcement action (a predicate) that takes an obliged action and a subject as input parameters and returns a boolean value indicating whether the subject has fulfilled the obliged action.*

3.3.5. Single Temporal Obligation.

Definition 8. *A single temporal obligation is defined as 6-tuple* $tobl = (s, sa, oa, d, ea, oblCheck) \in TO$. *A temporal obligation indicates that upon executing the subject action, sa, the obliged action oa must be fulfilled by the subject s within a period of* $d \in N$ *time units.* $ea \in EAct$ *is the compensation action in case of violation. OblCheck is the fulfillment predicate.*

3.3.6. Single Periodic Obligation.

Definition 9. *A single periodic obligation is defined as 7-tuple* $pobl = (s, sa, oa, d, n, oblCheck, ea) \in PO$. *A periodic obligation indicates that the obliged action must be fulfilled periodically,* n *times within a period of* d *time units. s, sa, oa, ea represent the subject, object, subject action, obliged action, and enforcement compensation action, respectively.*

3.4. Concurrecny

A concurrency rule expresses the way multiple users are allowed to use a resource at the same time, which can be categorized into *cardinal* and *type*. *Cardinal* indicates how many subjects are allowed to execute an action on a resource at the same time, while *type* indicates whether the concurrent execution is true or interleaved.

Definition 10. *A concurrency rule is defined as a triple* $conc = (sa, Par\_Set, n) \in CONC$, *where* $sa \in SA$ *is a subject action,* $Par\_Set \subseteq S$ *is a set of subjects that are allowed to execute the action sa in a true concurrent way, and* $n \in N$ *is the maximum number of subject that are allowed to execute the subject at the same time. We call* Par_Set *the true concurrent set.*

Based on the time instance in which security rules must be applied or checked, we classify rules into three categories: *Instant Rules (IRu)* , *Execution Rules (ERu)*, and *Post Rules (PRu)*, Figure 2 shows the different types of security rules applied on usage control actions. Instant rules are applied instantly when the action request is received and the action is only allowed if the instant rule is fulfilled. Execution rules are rules that are applied during the exection of an action and Post rules are rules that applied after the execution of an action is finished. It can be noticed that post rules do not restrict the access of resources, however they require that some actions/tasks (obligations) must be executed afterwords. In order to enforce post obligation rules we have to impose compensation actions upon the violation of these rules. For example, the student is allowed to *register* at the university if he has been accepted, but he must pay the fees within six months, otherwise her registration will be canceled. The rule that checks that the student is accepted is an instant rule to authorize the student to register. After registering, the student must pay the fees within six months (the post obligation task), otherwise her registration is canceled (the compensation action).

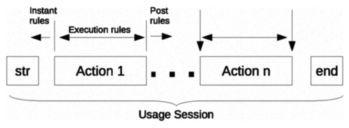

Figure 2: Different types of security rules.

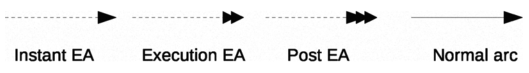

Figure 3: Different types of end arcs.

Based on this discussion we can conclude that instant right, instant obligation and cardinal right are of type IRu, temporal right is of type ERu, and temporal and periodic obligation rules are of type PRu. Thus we can write $IRu = \{IR \cup CR \cup IO\}$, $ERu = \{TR\}$, and $PRu = \{TO \cup PO\}$. The set of all security rules is denoted as $Ru = \{IRu \cup ERu \cup PRu\}$. Furthermore, we define for each secuirty rule a predicate $fulfill : Ru \rightarrow Bool$ that indicates whether the rule is fulfilled or not.

3.5. Usage Control Coloured Petri Nets (UCPN)

Based on the different rules discussed before, a usage control policy can be defined as a new type of Coloured Petri Net called Usage Control Coloured Petri Nets (UCPN). First we introduce UCPN and then show how this Petri Net can be used to construct a usage control policy.

Definition 11 (UCPN)**.** *A Usage Control CPN is defined as a tuple* $UCPN = (\Sigma, P, T, AA, N, C, G, E, R, I)$, *where*

- Σ, *P and T, N, C, G, E and I are colour sets, places transitions, a node function,a colour function,a guard function,an arc expression function and an initialization function, respectively. They represent the same elements of a standard CPN [7];*
- $AA = A \cup EA$ *is a finite set of arcs, where A is the finite set of "normal" arcs as defined in [7] and EA is the finite set of end arcs such that* $[A \cap EA = \emptyset] \wedge [\forall t \in T.\forall a \in EA : (\nexists p \in P : N(a) = (p, t))]$ *(for any transition,* end arcs *are always output arcs). Similar to the rule types, end arcs also have three types, instant*(IEA), *execution*(EEA), *and post* (PEA) *end arcs:* $EA = IEA \cup EEA \cup PEA$, *and they are drawn as dashed arcs with one, two or three heads, respectively(cf. Figure 3); and*
- $R : T \rightarrow P(Ru \cup CNC)$ *is a rule label function that maps each transition with a set of rules where* $P(Ru \cup CNC)$ *denotes the powerset of* $Ru \cup CNC$.

Similar to the function that maps each node to its surrounding arcs A (cf. section 2), we define also the functions: $EA : P \cup T \rightarrow EA_s$, $IEA : P \cup T \rightarrow IEA_s$, $EEA : P \cup T \rightarrow EEA_s$, $PEA : P \cup T \rightarrow PEA_s$, that maps each node to the surrounding- end arcs, -instant end arc, -execution end arcs, and -post end arcs, respectively.

Informally, semantics of end arcs indicate that the occurrence of transitions is three step process: First, instant rules are checked before the occurrence is allowed, upon a violation of any of these rules, *instant end arcs* are the only output arcs that will be "activated". Second, during the occurrence of a transition, execution rules are checked and any violation of these any of these rules will revoke the execution and *execution end arcs* are the only end arcs "activated". Finally, after the occurrence is allowed and the execution is ended, the post obligation rules are triggered. Formally, we can define the behavior of a transition occurrence in UCP-net, similar to the original definition 2.9 of CP-net in [7], as follows:

Definition 12 (Occurrence Rule). *When a binding element (t,b) is enabled in a marking M_1 it may occur changing the marking M_1 to anther marking M_2 defined as follows:*

(i) If $(\exists r \in R(t) : r \in IRu \wedge \neg fulfill(r))$ *then* $\forall p \in P : M_2(p) = (M_1(p) - E(p,t) < b >) + \sum_{a' \in IEA : N(a')=(t,p)} E(a') < b >$.

(ii) Else if $(\exists r \in R(t) : r \in ERu \wedge \neg fulfill(r))$ *then* $\forall p \in P : M_2(p) = (M_1(p) - E(p,t) < b >) + \sum_{a' \in EEA : N(a')=(t,p)} E(a') < b >$.

(iii) Else if $(\exists r \in R(t) : r \in PRu \wedge \neg fulfill(r))$ *then* $\forall p \in P : M_2(p) = (M_1(p) - E(p,t) < b >) + \sum_{a' \in PEA \cup A : N(a')=(t,p)} E(a') < b >$.

(iv) Else $\forall p \in P : M_2(p) = (M_1(p) - E(p,t) < b >) + \sum_{a' \in A : N(a')=(t,p)} E(a') < b >$.

Briefly, end arcs are activated according to which rules are (not) fulfilled starting from the time instant of requesting the action till post-obligations are checked. That is why the order in which rules are checked should be respected.

3.6. Usage Control Policy

A usage control policy is a UCPN that can be built based on the usage pattern $up = (sb, Pl') \in UP$, and the different rules defined for each subject action, and two set of predefined places and transitions, $Pl \subset P, Tr \subset T$, as follows (cf. Figure 4):

1) Define the rule label function, $R(t)$, for each transition, t, of the safe behavior sb . We call the safe behavior in which rules are defined for each transition a *Secure Safe Behavior*. All variables in R(t), must have a type that belongs to Σ. Subject, object, and context variables appears in $R(t)$ must be bound to the expression variables of these types in the incoming and outgoing arcs of t.
2) Add the following places: $Pl = \{Denied, Revoked, Violated\}$, where
   - $\forall t \in TRAN(up) : \exists r \in R(t) \wedge r \in IRu \Rightarrow \exists!(a_1, a_2, a_3) \in IEA^3 : N(a_1) = (t, Denied) \wedge N(a_2) = (t, Objects) \wedge N(a_3) = (t, Contexts) \wedge E(a_1) = s \in SUB \wedge E(a_2) = o \in OBJ \wedge E(a_3) = c \in CON$.
   - $\forall t \in TRAN(up) : \exists r \in R(t) \wedge r \in ERu \Rightarrow \exists!(a_1, a_2, a_3) \in EEA^3 : N(a_1) = (t, Revoked) \wedge N(a_2) = (t, Objects) \wedge N(a_3) = (t, Contexts) \wedge E(a_1) = s \in SUB \wedge E(a_2) = o \in OBJ \wedge E(a_3) = c \in CON$.
   - $\forall t \in TRAN(up) : \exists r \in R(t) \wedge r \in PRu \Rightarrow \exists! a \in PEA : N(a) = (t, Violated) \wedge E(a) = s \in SUB$.

   These places contain subjects whose request was denied, execution was revoked or obligation was violated, respectively.
3) Add the following transitions: $Tr = \{dny, cut, cmp\}$, where
   - $\bullet dny = \{Denied\} \wedge dny\bullet = \{Subjects\}$.
   - $\bullet cut = \{Revoked\} \wedge cut\bullet = \{Subjects\}$.
   - $\bullet cmp = \{Violated, Subjects\} \wedge cmp\bullet = \{Subjects\}$.

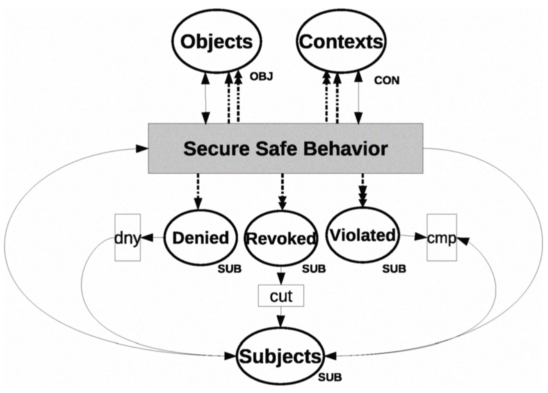

Figure 4: Usage control policy general form.

Figure 4 shows how to build a usage control policy based on usage pattern and the rules defined for each transition in the safe behavior. End arcs connected to a transition, $t \in T$, for which security rules are defined, ensure that the violation of these rules results in (i) denying or cutting the execution by preventing the tokens to flow normally to the next places, (ii) moving these tokens to the appropriate places, like Violated, Revoked, or Denied, and (iii) imposing compensation actions to violating subjects afterwords.

3.7. Running Example

In order to illustrate the steps of defining a usage control policy, we consider the following example about the usage of sockets in Grid systems (Securing sockets in Grid systems was discussed in [10]). We assume a reference monitor that controls the usage of local sockets performed by applications executed on behalf of remote GRID users. We assume that one behavioral, two security, and one concurrecny requirements must be fulfilled when using the socket resources:

`BR1`: sockets must be opened before data is sent [3].

`SR1`: only registered users are allowed to open connection on socket ports with numbers between 1000 and 2000, from within the local network of the organization.

`SR2`: each user is only allowed to send packets for a maximum time of 10 minutes.

`SR3`: each user wants to send a message must send a report about the purpose of her usage to the administrator within one week, otherwise the she will not be allowed to use sockets in the future.

`CR2`: only administrators are allowed to open ports at the same time.

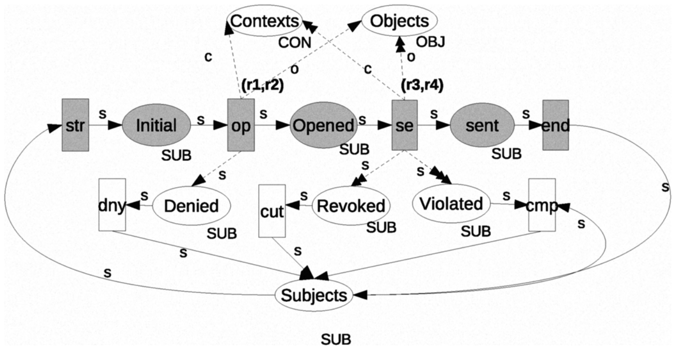

Figure 5: The usage control policy of the socket example (double headed arcs between Objects and Contexts places and the transitions of the safe behavior are omitted for clarity).

It can be seen from these requirements that subjects in this case are grid users, objects are sockets, and the needed context entity is a system locator, that returns the IP address of connecting machins. Subject attributes are, ID, of type $string$, $registered$ of type $boolean$, and $role$ of type $string$. Object's attributes are ID and $ports$ of type $integer$. Contexts attribut is IP of type $integer$.

After determining the basic elements of the policy, we start defining the different rules according to the requirements. Behavioral rule (`BR1`) can be encoded in a CPN, the shadowed sub-net of Figure 5, that starts with str transition and ends with end transition. We can identify three security rules, `SR1` is represented by an instant access right on the action $open$, as it must be fulfilled before the usage is granted; `SR2` is represented by a temporal access right on the action $send$; and `SR3` is represented by a temporal obligation rule (post rule) as it must be fulfilled after the usage is finished. Finally, `CR1` is represented by a concurrency rule on the action *open*. For example, we can specify the first rule in ML as follows:

```
colset SUB = record id:String*registered:Bool;
colset OBJ = record id:String*ports:Integer;
colset CON = record id:String*IP:String;
colset PRO = product SUB*OBJ*CON;
type IR = PRO*(PRO->Bool); //specification of an instant right
var s:PHY, o:REC, c:CON;
fun authCheck (s,o,c) = if #2 s=true andalso
#2 o > 1000 ansalso #2 o <2000 andalso
#2 c IP = "x.w.y.z" 2000 then true else false;
var r1 = (s,o,c,authCheck); //the instant right r1
```

4. Related Work

The closest to our work is the specification [18], [6], [11] of $UCON$ usage control model [13]. The main problem of UCON is the lack of post obligation support, the integration between functional and security aspects, and concurrency control. Our approach tries to have a clear integration between the safe behavior of the application and the security policy. We have tried in previous work [9] to enhance the obligation expressiveness of UCON, however we overlooked the concurrency issue and the approach was done in an ad-hoc manner that makes the analysis of the resulted policy difficult. Finally, concurrency was not consider by UCON.

Hilty et al. [4] proposed *OSL* (Obligation Specification Language) as a policy language for distributed usage control. OSL proposes an expressive policy language for usage control considering temporal, cardinal and permit rules. However, the continuity aspect of control beyond one action, and concurrency issues were overlooked.

A plethora of work in the literature to specify obligation and access control policies, not in the context of usage control, can be found. Some concentrate on the area of privacy like in [12], others proposed general security and system management policies [3], [15]. The closest among these policy languages is the work done by Bandara et al. [2] which uses event calculus or in [1] which is based on first-order logic formulas. The major distinct and novel contribution in this work comparing to these policy languages are the continuity of control and the integration of functional (system behavior) and non-functional (security rules) requirements, which we believe is an essential issue when dealing with usage control. Furthermore, concurrency tends to be overlooked. To our knowledge the only work that tackles concurrency in the context of usage control is done by Janicke et al. [5]. They propose a static analysis method by creating the dependency graph between different controllers/users, however fail to define concurrency rules for users within one controller.

Few work have used CP-nets for analyzing and modeling existing access control policy models like chinese wall policy [19], integrity policy (Biba Model) [20], mandatory access control [8], and role-based access control policy [14],

3. Please note that this is just a hypothetical example to show the feasibility of the approach.

[16]. The main difference to our work is that we are proposing a new security model aiming at meeting the new security requirement in collaboration and distributed environments. Thus ours is an attempt to tackle fundamental questions of security policies. on the other hand, the other work takes one existing model and analyze some security properties of it. Furthermore, Concurrency and integration between functional and security aspects are unique in our treatise.

5. Conclusion and Future Work

In this paper we present a novel approach for the specification of a usage control policy based on Coloured Petri Nets. Our usage control policy provides (i) continuity of control and integration of functional and security aspects by supporting behavioral rules, (ii) post obligations and the compensation actions they require, and finally (iii) concurrency control. For the purpose of the specification we introduce *Usage Control Coloured Petri Nets (UCPN)*. UCPN extends traditional CPN with security aspects, namely with the concepts of rule label functions and end arcs. The full enforcement semantics of the defined UCPN and their analysis are planned for future work. Our final goal is close the gap between our usage control policy and the enforcement mechanisms of its reference monitor.

References

[1] A. Bandara, J. Lobo, S. Calo, E. Lupu, A. Russo, and M. Sloman. Toward a formal characterization of policy specification analysis. In *Annual Conference of ITA (ACITA), University of Maryland, USA*, August 2007.

[2] A. Bandara, E. Lupu, and A. Russo. Using event calculus to formalise policy specification and analysis. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 26, Washington, DC, USA, 2003. IEEE Computer Society.

[3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 2001.

[4] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. volume 4734, page 531. Springer, 2007.

[5] H Janicke, A Cau, F Siewe, and H. Zedan. Concurrent enforcement of usage control policies. *Policy 2008*, 2008.

[6] H. Janicke, A. Cau, and H. Zedan. A note on the formalization of ucon. *SACMAT' 07, June 20-22, 2007, Sophia Antipolis, France*, 2007.

[7] K. Jensen. *Coloured Petri Nets*, volume 1. Springer-Verlag, 1992.

[8] K. Juszczyszyn. Verifying enterprise 's mandatory access control policies with coloured petri nets. In *WETICE '03: Proceedings of the Twelfth International Workshop on Enabling Technologies*.

[9] B. Katt, X. Zhang, R. Breu, M. Hafner, and J.-P. Seifert. A general obligation model and continuity enhanced policy enforcement engine for usage control. In Indrakshi Ray and Ninghui Li, editors, *SACMAT '08: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies.*.

[10] H. Koshutanski, F. Martinelli, P. Mori, and A. Vaccarelli. Fine-grained and history-based access control with trust management for autonomic grid services. In *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*.

[11] F. Martinelli and P. Mori. A Model for Usage Control in GRID systems. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 169–175, 2007.

[12] Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control policies and privacy policies. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*.

[13] J. Park and R. Sandhu. The ucon_abc usage control model. *ACM Transactions of Information and System Security*, 7(1):128–174, 2004.

[14] H. Rakkay and H. Boucheneb. Security analysis of role based access control models using colored petri nets and cpntools. pages 149–176, 2009.

[15] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guede. Spl: An access control language for security policies with complex constraints. In *Proc. of the Network and Distributed System Security Symposium*, 2001.

[16] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor. A role-based access control policy verification framework for real-time systems. In *WORDS '05: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*.

[17] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security*, 8(4):351–387, 2005.

[18] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. *SACMAT04*, 2004.

[19] Z.-L. Zhang, F. Hong, and J. Liao. Modeling chinese wall policy using colored petri nets. In *CIT '06: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*.

[20] Z.-L. Zhang, Fan H., and H. Xiao. Verification of strict integrity policy via petri nets. *Systems and Networks Communication, International Conference on*, 0:23, 2006.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8394
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8394