Deploying Access Control in Distributed Workflow

Samiha Ayed[1] **Nora Cuppens-Boulahia**[1] **Frédéric Cuppens**[1]

[1] ENST-Bretagne
Cesson Sevigne 35576, France
Email: {Samiha.ayed, Nora.cuppens, Frederic.cuppens}@enst-bretagne.fr

Abstract

Workflows are operational business processes. Workfow Management Systems (WFMS) are concerned with the control and coordination of these workflows. In recent years, there has been a trend to integrate WFMS in distributed inter-organizational systems. In this case malfunctioning of one WFMS can affect more than one organization, making the correct functioning of a WFMS a critical issue. Thus, an important function of WFMS is to enforce the security of these inter-organizational workflows. Several works have been done to integrate the security aspects in the workflow specification. Unfortunately, these research works generally adopt a centralized management approach and are based on static access control models. In this paper, we suggest a decentralized and dynamic approach to handle access control in workflows.

Keywords: WFMS, OrBAC, Petri Nets, Security Policy.

1 Introduction

WFMS are based on representing processes as workflows. A workflow representation implies that tasks composing it are interdependent and are communicating control information and data to each other. For example, let us consider a workflow composed of tasks T_1, T_2 and T_3 which must be executed in a sequential order. If we suppose that these three tasks act on same documents, the access to these documents must be controlled according to the execution order of tasks. In other words, this access control must be synchronized with workflow execution progression. In addition, the execution of a task is related to the execution of precedent tasks. So, a workflow specification must be correlatively defined with a security policy. Several proposals (Adam, 1998; Atluri, 1996; Huang, 1999; Hung, 1999; Bertino, 1999) tried to address this problem and proposed different access control models to manage security in WFMS. Their propositions consist in (1) specifying the global workflow security policy and (2) defining a *centralized* management procedure that controls the execution of the workflow so that it remains compatible with its associated security policy. Since these approaches are generally based on the RBAC model (Sandhu, 1996) which only provides means to specify static security requirements, they present some limitations that will be further detailed in the sequel.

In this paper, we suggest a different approach. The global security policy associated with the workflow execution is specified using the OrBAC model (Abou, 2003). The OrBAC model provides means to define dynamic and contextual security requirements. For this purpose, this model defines two useful notions. The first notion is the *organization* which can be seen as an organized group of active entities. Workflow tasks may be executed in the same or different organizations. If they are executed within the same organization, the policy has to manage security in this organization. The notion becomes more useful if workflow tasks are executed in independent organizations. In this case, flows between different organizations must be managed. The second useful notion defined in OrBAC is the *context*. A context is used to express permissions or prohibitions that apply in specific circumstances. Each context has a name and its definition depends on the organization (Cuppens, 2003). Therefore, the term "context" corresponds to any constraint or extra conditions that join an expression of a rule in the access control policy. OrBAC classifies contexts according to their type. A *provisional context* depends on previous actions the subject has performed in the system. In other words, it corresponds to a history of execution. Provisional contexts are very interesting in the domain of WFMS since the execution of a task depends on the execution history of precedent tasks. Also, it permits the definition of a dynamic security policy according to contexts, a very useful requirement in WFMS. Another relevant context in OrBAC is *temporal context*. A such context permit defining temporal constraints and expressing dynamic temporal rules. This context is as interesting as provisional context in WFMS domain.

In this paper, using notions of organization and context already existing in OrBAC, we present a petri net based model for modelling workflow application and we define a security policy that we associate to it in order to define a secure execution environment. Then, we show how to manage this security policy to control the workflow execution in a *distributed* manner. For this purpose, we define an algorithm to generate the *local* security policy associated with the execution of each task that composes the workflow. The based petri net model and the global policy are provided as inputs of this algorithm, a dynamic policy is its output. Our approach remedy to the static and centralized aspect of models already proposed.

The paper is organized as follows. Section 2 introduces definition of WFMS and related works studying security in these systems. Section 3 presents an overview of petri net that we use to define our WFMS model. This model provides means to specify fine-grained execution modes between tasks using Allen's temporal intervals (Allen, 1993). Section 4 presents a brief overview of the OrBAC model and its use to model workflow security. It defines our security policy which gathers a general security policy and a coordination security policy. Section 5 addresses the issue of distributed workflow execution and defines an algorithm to generate the local policies required to execute the workflow in a secure distributed way. It also discusses the algorithm through an example. Finally, section 6 concludes the paper and outlines future work.

2 WFMS and related works

2.1 WFMS

A workflow is a representation of a process. This process is divided into many tasks having an interdependent execution. The execution of these different tasks may include temporal constraints or conditions. Actually, we can suppose different manners of tasks execution: two tasks may be executed in sequential, parallel or concurrent mode. A workflow is composed of two phases: a building phase and an execution phase. The first phase defines the formal representation of the process. The second one is an instantiation of the workflow which is based on the specification defined in the first phase.

A WFMS is a system which supports the specification, control, coordination and administration of processes using workflows. This is done through the execution of software which is based on the logic of workflows. These systems are used in different domains: research, industry, commerce, etc. Their management can be either centralized or distributed, according to the tasks of the process and the operation mode of the system. These systems may include many subjects or roles and many resources when managing and executing different workflows. This introduces the need to care about the security in these systems.

Tasks of a workflow may be executed by the same subject or by different ones having different roles and must access to different resources of the system. So, to ensure a secure execution, tasks must be executed by authorized persons or subjects and according to their execution order specification. In addition, subjects must have access only to objects available and authorized. This access must be limited to the period of task execution. Thus, granting or revoking privileges must be synchronized with the workflow execution progress. All these requirements lead us to combine the conception of the workflow and the security policy associated to it. Since security is essential and represents an integral part of a workflow, this security policy must ensure many properties: integrity, authorization, availability, confidentiality, authentication and separation of duty.

2.2 Related Works

Studying security in WFMS has been a topic of several research works. WFMC (Workflow Management Coalition) focuses on the development of workflows through standards that provide connectivity between different products. WFMC suggests WfRM as a model of reference for workflow. It presents a management system of workflow and its different interfaces. WFMC defines different services to ensure a secure environment: identification, authentication, authorization, confidentiality, integrity and non repudiation. (WFMC, 1995) studies these services using WfRM. In (WFMC, 1998), WFMC suggests a simple security model which defines a set of security operations. Also, it investigates inter-operability between two parties. Hence, it presents an extension to the inter-operability protocol to support the workflow service authentication data and the workflow inter-operability protocol data. However, this protocol does not consider the flow of authorizations among parties, tasks and resources during the workflow execution.

(Atluri, 1996; Adam, 1998) propose a conceptual and a logic model WAM (Workflow Authorization Model) to enforce the authorization flow based on the inter-dependences between tasks using coloured and temporised petri nets. The model defines an authorization template (AT). These ATs are defined during the building phase and they are used to derive the actual authorization during the execution phase. WAM tries to synchronize the flow of authorization with the workflow and to specify temporal constraints using a static approach. Different algorithms and interpretation of modelling are presented in (Adam, 1998). This model does not consider neither the order of execution of tasks for the same object nor the authorization access to resources. Their approach remains also static. Based on the WAM model, Alturi and Huang have developed in (Huang, 1999) a model called SecureFlow for the web in WFMS. The model uses a simple language of 4G to specify different constraints on authorization. (Huang, 1999) defines the specification of constraints and the security policy associated with this model. Both WAM and SecureFlow are only adequate for centralized management of workflow security. Also they do not consider integrity and availability properties.

Hung and Karlapalem (Hung, 1999) have presented an authorization model for WFMS. This model addresses three security properties: integrity, authorization and availability. The model is based on defining a set of invariants. These invariants concern several aspects: agents, events and data of the workflow. The model is defined as an abstract machine having three layers: workflow layer, control layer and data layer. The first contains authorizations which can be granted to or revoked from an agent. The second deals with events which can be generated during the execution phase. The last layer manages granting and revoking authorizations to documents. Security requirements are then transformed into a set of invariants in different layers. Using these invariants, authorizations functions are defined in order to ensure system operation. In spite of supporting concurrent execution of tasks, the model does not address different temporal constraints which can be present in a workflow system specification.

(Bertino, 1999) studies authorizations in WFMS considering different constraints. Based on the constraints evaluation time, they define three classes of constraints: static constraints, dynamic constraints and hybrid constraints. This work is based on assigning users and roles to tasks. So, this ensures the non violation of constraints but does not address temporal constraints and constraints based on events. Also, it supposes that tasks are executed in a sequential mode. In the case where two tasks are executed in concurrent mode, it is impossible to define constraints on a task based on the success or failure of tasks. Thus, this model cannot be considered complete and does not address the issue of distributed management of workflow security requirements.

The definition of the security policy of workflows in most of the aforementioned works is based on the RBAC model. This model is restricted to positive authorizations, namely permissions. Also, it defines security policies using a static approach which is not appropriate to WFMS known to have a dynamic behavior. To remedy this lack, we choose to base our security policy on OrBAC model as further explained in section 4. We first present our approach to model workflows.

3 Modelling WFMS

A workflow process is generally modelled as a set of tasks connected by arcs from one task to the task which follows in the execution order. Tasks executed in parallel can be also represented in the workflow. Conditions that can be related to the execution of tasks are represented by arcs. Also, arcs contain information which are exchanged between two successive tasks.

3.1 A petri net based WFMS model

Our WFMS model is based on petri nets (David, 1992) which provide an expressive formalism to represent synchronous activities. With the graphic representation that they offer, they are considered simple and easy to understand. In addition, they have a mathematical foundation which provides means to formally analyze obtained models. Thus, petri nets allow a flexible transition from the conceptual level to the implementation of test, where the system can be simulated and validated before proceeding to a detailed conception and implementation.

Definition 1: *a petri net is defined as a 5-tuple, PN = (P, T, F, W, M_0), where:*

- *P is a finite set of places,*
- *T is a finite set of transitions,*
- *$F \subseteq (P \times T) \cup (T \times P)$: a set of arcs from places to transitions and from transitions to places,*
- *$W : F \rightarrow \{1, 2, 3, ...\}$: a weight function, it defines weights assigned to different arcs,*
- *M_0: $P \rightarrow \{0, 1, 2, ...\}$: the initial marking, it describes initial place contents.*

Definition 2: *in a synchronized petri net, to each transition is associated an event and the release of the transition will happen (1) if the transition is valid, (2) when the event occurs.*

Definition 3: *a P-temporized petri net is a couple <R, Tempo> where (1) R is a marked petri net and, (2) Tempo is an application from the set of places to the set of positive and rational numbers. Tempo(P_i) = d_i : temporisation associated with P_i.*

Definition 4: *an interpreted petri net has the following three characteristics: (1) It is synchronized, (2) It is P-temporized, (3) It contains an operative part where the state is defined by a set of variables V. This state is modified by operations Op which are assigned to places. It determines the truth value of conditions (predicates) C which are associated with transitions.*

Definition 5: *a coloured petri net assigns colours to marks. So a coloured petri net differs from a general petri net by the addition of a set of colours.*

The model that we suggest uses a combination of interpreted and coloured petri nets. This choice is motivated by the requirements of processes we want to represent. In fact, choosing interpreted petri nets is related to conditions which can be associated with process execution. Therefore, we assign operations to different tasks. So, we need an operative part in the petri net representation. On the other hand, the choice of coloured petri net is done to clarify the process execution. In this case, each colour indicates a new process execution. A new execution of the workflow is defined using a new colour. An example of a petri net is presented in figure 1.

Intervals Relations	Associated Symbols	Execution modes of T_i and T_j
Meets(I_1 , I_2)	$\mid$	$T_i \mid T_j$
Before(I_1 , I_2)	$\mid_<$	$T_i \mid_< T_j$
Equal(I_1 , I_2)	$//_=$	$T_i //_= T_j$
Starts(I_1 , I_2)	$//_<$	$T_i //_< T_j$
Finishes(I_1 , I_2)	$//_>$	$T_i //_> T_j$
During(I_1 , I_2)	$//_\subset$	$T_i //_\subset T_j$
Overlap (I_1 , I_2)	$//_{<>}$	$T_i //_{<>} T_j$

Table 1: Different execution modes.

3.2 Different execution modes

In our WFMS model, we do not want to restrict execution modes of two tasks to sequential and parallel execution. Instead we take our inspiration from which defines complete set of execution modes of two tasks having each an execution time interval.

So, let T_i and T_j be two tasks having respectively two execution intervals I_1 and I_2. There is a basic set of mutually exclusive primitive relations that can hold between temporal intervals (Allen, 1993). We recall these relationships which can exist between the two intervals and classify them in two classes describing sequential and parallel execution.

Sequential execution

- Before(I_1, I_2): time interval I_1 is before interval I_2, and they do not overlap;
- Meets(I_1, I_2): this is a particular case of Before. Interval I_1 is before interval I_2, but there is no interval between them, i.e., I_1 ends when I_2 starts.

Parallel execution

- Equal(I_1, I_2): I_1 and I_2 are the same interval;
- Starts(I_1, I_2): time interval I_1 shares the same beginning as I_2, but ends before I_2 ends;
- Finishes(I_1, I_2): time interval I_1 shares the same end as I_2, but begins after I_2 begins;
- During(I_1, I_2): time interval I_1 is fully contained within I_2;
- Overlap(I_1, I_2): time interval I_1 starts before I_2 and they overlap.

To define the mode of execution of two tasks we actually refer to their execution intervals. So to model this order, we associate symbols to represent different relations. These symbols are presented below. The execution mode of two tasks T_i and T_j having respectively an execution interval I_1 and I_2 is presented in the third column of the table 1.

These different execution modes of tasks are used in our WFMS security policy to ensure a secure execution environment of workflow.

4 Specifying WFMS global security policy

To define a secure execution environment of workflows, we suggest using the OrBAC model and its concepts. Thus, before presenting how to express security requirements associated with the workflow execution, we first briefly recall basic concepts suggested in the OrBAC model and we present our modelling of WFMS applications..

4.1 OrBAC in brief

In order to specify a security policy, the OrBAC model (Abou, 2003; Cuppens, 2004) defines several entities and relations. It first introduces the concept of *organization* which is central in OrBAC. An organization is any active entity that is responsible for managing a security policy. Each organization can define its proper policy using OrBAC. Then, instead of modelling the policy by using the concrete implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned to in an organization. The role of a subject is simply called a role as in the RBAC model. The role of an action is called activity and the role of an object is called view. Each organization can then define security rules which specify that some roles are permitted or prohibited to carry out some activities on some views. Particularly, an organization can be structured in many sub organizations, each one having its own policy. It is also possible to define a generic security policy in the root organization. Its sub organizations will inherit from its security policy. Also, they can add or delete some rules and so, define their proper policy. The definition of an organization and the hierarchy of its sub organizations facilitate the administration (CM, 2004). The security rules do not apply statically but their activation may depend on contextual conditions (Cuppens, 2003). For this purpose, the concept of *context* is explicitly included in OrBAC. Contexts are used to express different types of extra conditions or constraints that control activation of rules expressed in the access control policy. So, using formalism based on first order logic, security rules are modelled using a 6-places predicate: security_rule (type, organization, role, activity, view, context) where type belongs to permission, prohibition, obligation. As an example, we can define the following security rule: security_rule(permission, a_hosp, nurse, consult, medical_record, urgency) means that, in organization a_hosp, a nurse is permitted to consult a medical record in the context of urgency.

4.2 WFMS petri net model

In this section, we use the OrBAC concepts to define our WFMS petri net model. It is based on a formal representation of workflows using petri nets. Our modelling is constructed in term of roles, views, activities, organizations and contexts. Roles, views and activities notions are used as they are defined in OrBAC. These concepts are defined within the formalism of petri nets to express functional aspects of a business process. Then, a such structuring will allow us to define merely our security policy. So, roles, views and activities defined in OrBAC are respectively interpreted within the petri net context: (1) R: a finite set of roles defined in the process, (2) V: a finite set of views used during the execution of the process, (3) A: a finite set of activities associated with places. An activity can be an atomic operation or a composed operation. A place can be validated only if all operations associated with this place are executed.

We also use the concepts of context and organization suggested in OrBAC. The concept of organization is associated with places. Different operations defined in relation with a place are executed within an organization. Different places can belong to the same or to different organizations. Also, a hierarchy may exist between different organizations. Managing these different organizations can be either centralized or distributed. In the first case, a root organization must manage security with different other organizations which can be sub organizations of the root organization. Then, they receive or inherit their security policy from the root organization. In the second case, each organization defines and manipulates its proper security policy. In this case, information can be exchanged between the set of organizations to be able to know what is happening globally.

The notion of contexts is also associated with places. OrBAC defines several categories of contexts (Cuppens, 2003). The *Prerequisite context* aims to restrict or extend privileges granted to a role depending on some conditions. So, this context category is useful in WFMS to specify constraints associated with the workflow process execution. For instance, if P_1 and P_2 are two places of a workflow, we can associate the place P_2 with a context same_subject(P_1) to constrain subjects who are executing activities assigned to place P_1 and P_2 to be equal.

Also, the *Provisional context*, that depends on previous actions the subject has performed in the system, is relevant for WFMS. In fact, in a workflow execution, a task execution depends on the execution history of precedent tasks. Hence, with each place we can associate a context. It represents the context of execution of operations associated with this place. It consists of two parameters: a color and a provisional context called execution context. So, let Cont_i be the context of the place P_i. Cont_i = (colour, execution_context): the colour is used to identify the instance of process execution. The second parameter defines requirements associated with P_i execution. It indicates precedent tasks that must be executed before the execution of operations associated with P_i. It represents places according to their execution order using definitions of execution modes done in sub section 3.2.

However, in our model, we suppose that, when defining the global security of the WFMS, the execution context of each place is not explicitly defined because it can be derived from the petri net representing the workflow. Thus, in the global policy, the execution context of a place is only containing the place itself. Then it will be enriched. We go in further details in our security policy by taking into account when construction this execution context different execution modes. This will allow us to deal with different temporal constraints present in a workflow.

Finally, we define a specific type of tokens which are used in our model. A token is a triple <r, v, cont(P)>. For a token <r_i, v_i, cont(P_i)> placed in a place P_i, the first parameter (r_i) indicates the role eligible to execute the operation associated with P_i. The second parameter (v_i) designates the view or the type of object which will be used in the operation associated with P_i. The last parameter (cont(P_i)) represents the context of the place P_i.

4.3 Example

To clarify the model proposed, we apply it to an application of initiating a mission for an enterprise employee. This application will need to reserve a fly, a hotel and a car. First, we introduce essential elements to define the petri net model that we propose to represent the application of initiating a mission. Then, we present the model in figure 2.

- Roles defined in the model :
  - r_1 : traveller
  - r_2 : checker
  - r_3 : a responsible of the agency of car rental
  - r_4 : an agent of the hotel
  - r_5 : a responsible of the airline

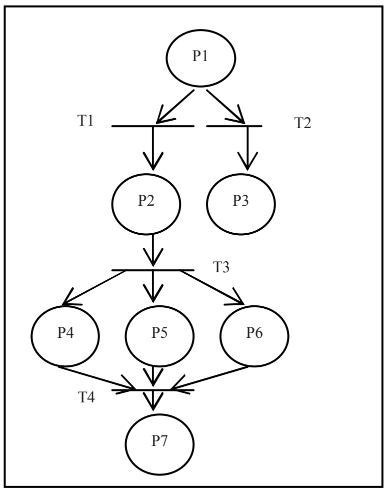

Figure 1: Example of a petri net

- Types of used objects :
  - o_1 : slip of a mission
  - o_2 : demand of travel
  - o_3 : slip of reservation
  - o_4 : slip of validation
  - o_5 : mail of information
- Operations associated to different places, an operation Op_i is associated to a place P_i :
  - Op_1 : creating a mission
  - Op_2 : submit the demand of travel
  - Op_3 : reservation of a car
  - Op_4 : reservation of a hotel
  - Op_5 : reservation of a flight
  - Op_6 : validation of the mission
  - Op_7 : informing the traveller
- Execution intervals defining during time of each operation: I_1= [0, 2], I_2 = [3, 5], I_3 = [6, 8], I_4 = [7, 8], I_5 = [6, 9], I_6 = [9, 10], I_7 = [10, 11]

Using these elements, we represent the initial model in reference with the application. This representation (figure 2) introduces the model before the execution of the process. So, contexts contain just places to which they are associated. With the execution progression of the process, they are changed dynamically.

For example, if we consider T_3, this transition will be valid and so we will be able to validate the mission (Op_6) only if we accomplish Op_3, Op_4 and Op_5.

When the process starts its execution, contexts change with it. These changes are explained in the algorithm proposed in the next section. This petri net modelling is the first input of our algorithm presented in the sequel. The second input will be the security policy that we specify in section 5. So, a system manager must define a petri net structuring of its WFMS application according to our proposed model approach.

4.4 WFMS security policy

Once we have the petri net modelling of our WFMS application, we define the security policy that we must associate to the model. To ensure a secure execution environment of the workflow, we must take into account two aspects:

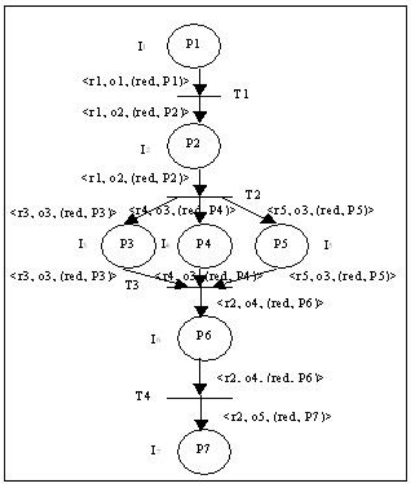

Figure 2: Petri net representation of the application

1. We have to control access to objects during tasks executions
2. We have to ensure and enforce different execution modes defined in the workflow process.

This twofold control could make our security policy more useful and increases assurance that it is correctly specified. Our policy is premised on a general security policy and on a coordination security policy. This security policy is the second input of our algorithm. So a system manager must have as inputs, his petri net modelling and the policy associate to it in order to generate a dynamic policy that it is updated with the workflow execution progression.

4.4.1 WFMS general security policy

Our general security policy defines access control rules. These rules are expressed using OrBAC model. So a general security rule is defined as a 6-tuple: *security-rule (type, organization, role, activity, view, context)*. These rules use the specific context defined in our petri net model (see section 4.2). Thus, let us consider the petri net of figure 1. A security rule defined as a permission granted to a role R_5 within the organization Org_5 to execute an activity A_5 associated to the place P_5 in this petri net and using a view V_5 will be expressed as follow: *security-rule (permission, Org_5, R_5, A_5, $view_5$, (red, P_5))*. "red" represents the instance of process execution associated with this rule. We have already supposed that our initial contexts contain just the place it self. It is not necessary to explicitly specify the execution context associated with this rule. This execution context will be automatically derived from the workflow description using the algorithm presented in the following section. In this policy we exploit provisional contexts defined in OrBAC.

4.4.2 WFMS coordination security policy

The general security policy manages access control but it does not deal with different tasks execution modes. So to complete our security policy we propose a coordination security policy which ensure a secure environment to execute workflows. This coordination security policy is collateral to our general security policy. This policy controls different execution modes presented in section 3.2. Also, it enforces different task dependencies and so it preserves the well functional execution of workflows.

Coordination policy is based on temporal contexts. These contexts can be used in conjunction with other contexts or conditions. They depend on the time at which the subject is requesting for an access to an object or a view. With temporal contexts, it should be possible to express that a given action made by a given user on a given object is authorized only at a given time or during a given time interval or also after or before another task execution (Cuppens, 2003). To express our coordination contexts, we reuse predicates defined in (Cuppens, 2005): *start(T)* ("starting T"), *doing(T)* ("doing T") and *done(T)* ("finishing T"), T is a task. We define another predicate *complete(T)* which means that T is finished since a moment. The difference between *done(T)* and *complete(T)* is that in the second case there is a duration since the task was accomplished. In the first case this duration does not exist.

To express a conjunction between different contexts and conditions, we use the first order logic.

To each task we associate two specific activities called *begin* and *end.* begin(T_i) means the activity to start the task T_i. end(T_i) means the activity to finish the task T_i.

Our coordination policy is defined as a set of rules associated to each case of different execution modes. To ensure these execution orders, we are lead to use not only permissions to control our workflow execution but also obligations. Using obligations is imposed by constraints defined in different execution modes. For instance if we consider the case of Equal(I_1, I_2), then T_1 and T_2 are obliged to begin and end at the same moment.

We present our coordination policy in table 2. This policy makes explicit different execution modes presented in section 3.2. Thus, to each case, we associate its coordination security policy that we must respect and apply. For simplicity, we represent these coordination rules using just the activity and the context predicates. But these rules are defined in reality within an organization, for a specific role and using a specific view. Also, we use *"default"* context to define a context where neither conditions nor constraints on the activity are required.

Execution mode	Associated Policy
$T_i \mid T_j$	- P(begin(T_i), default) - O(begin(T_j), done(T_i))
$T_i \mid_< T_j$	- P(begin(T_i), default) - O(begin(T_j), complete(T_i))
$T_i \ //_= \ T_j$	- P(begin(T_i), default) - P(begin(T_j), default) - O(begin(T_i), start(T_j)) - O(begin(T_j), start(T_i)) - O(end(T_i), done(T_j)) - O(end(T_j), done(T_i))
$T_i \ //_< \ T_j$	- P(begin(T_i), default) - P(begin(T_j), default) - O(begin(T_i), start(T_j)) - O(begin(T_j), start(T_i)) - O(end(T_i), doing(T_j))
$T_i \ //_> \ T_j$	- P(begin(T_j), default) - O(begin(T_i), doing(T_j)) - O(end(T_i), done(T_j)) - O(end(T_j), done(T_i))
$T_i \ //_\subset \ T_j$	- P(begin(T_j), default) - O(begin(T_i), doing(T_j)) - O(end(T_i), doing(T_j))
$T_i \ //_{<>} \ T_j$	- P(begin(T_i), default) - O(begin(T_j), doing(T_i)) - O(end(T_i), doing(T_j))

Table 2: WFMS coordination security policy.

5 Deploying WFMS security requirements

5.1 Decentralized control

Section 4 showed how to specify the global workflow security policy as a set of OrBAC security rules that apply to places. It is straightforward to define a management procedure compliant with a given global policy if we assume that this workflow is *centrally* managed. In this case, a request by a subject to perform an action on an object in this workflow is authorized if (1) this request is permitted by the global security policy and (2) this action can be activated according to the workflow current marking.

However, if we assume that the workflow management is distributed on several components, we can longer assume that each component has a complete view of the workflow. Thus, our objective in this section is to derive, from the global policy, the local policy to be managed by such distributed components. For this purpose we define an algorithm that takes as input the workflow description and the global security policy associated with this workflow. This algorithm provides as output the local policy. This local policy is conditioned with the context of the place. A place P_i is valid, and so its operations will be executed, only if its context contains the place P_i. After the end of operations execution associated with P_i, the place P_i is deleted from its own context. Thus, it will be not valid until another execution or another event in the petri net adds the place in its context. The local policy dynamically changes in relation with contexts. This policy synchronizes the global policy with the execution of the petri net. So, flows of information between different places are secured with this local policy. In fact, when a place P_i is using a document d and another place P_j must use the same document after being used by P_i, P_i must not have access to this document when it is used by P_j. This is to ensure the integrity of the system. Using the local security policy, this risk is eliminated since access to different documents and objects is controlled by contexts. So, this approach addresses the subject of security in distributed WFMS with a more appropriate manner.

5.2 Algorithm to deploy decentralized access control

To define our algorithm, we first define the concept of token and introduce some hypotheses, variables, functions, sets and tables. We then present the algorithm. It introduces two levels of security policy: a global security policy and a local security policy. The first one is considered an input of the algorithm. The second is provided as output to follow the process execution. We can qualify the global security policy as static and the local security policy as dynamic since it depends on contexts of different places.

Hypotheses: (1) The initial marking of the petri net is a token in the first place. This comes from the definition of a workflow. (2) All transitions are synchronized to the event e: this event is always occurring.

Variables: (1) i, j, k, n, m : integer, (2) Bool, result : Boolean (3) Cont(P_i): context of the place P_i (4) A token <r_k, v_k, Cont(P_k)> (5) I_k = [ts_k, te_k]: execution interval of a task where ts_k is the beginning and te_k is the end of the task.

Functions: (1) Card (E) = | E | : returns the cardinality of the set E. (2) M (P_j): returns the set of tokens of the place P_j. (3) Index (E): returns index of different elements of the set E. (4) State (i) : returns the temporal execution mode associated with the transition T_i (different execution modes presented in section 3.2) (5) Valid_transition (bool, i): function which verifies if the petri net contains a valid transition. If this valid transition exists, bool will be true and i will return its index.

Sets: (1) P: set of places (2) T: set of transitions (3) | P | = m et | T | = n (4) $^{o}P_j$: set of input places of the transition T_j (5) P_j^o: set of output places of the transition T_j (6) $^{o}T_j$: set of input transitions of the place P_j (7) $^{o}T_j$: set of output transitions of the place P_j (8) $P^o_{\{i,j..,k\}} = P^o_i \cup P^o_j \cup ... \cup P^o_k$, (identically for other sets) (9) $Cond_j$ = cond(T_j) = C_{ex}, C_1, ...,C_k: set of conditions associated with the transition T_j where: $C_{ex} = \{C_{ex_i}, C_{ex_j}, ...\}$: set of conditions of execution of input places of the transition T_j. $Cond_j$ = true only if all conditions associated with T_j are true. C_{ex} is true only if all conditions of execution of input places are true.

Tables: (1) Valid[1..n]: table of Boolean which indicates validity of transitions. (2) Org[1..m]: table which indicates organizations of different places of the petri net. (3) R[1..m]: table which indicates roles associated with different operations of places. (4) A[1..m]: table which indicates different activities (operations) associated with different places. (5) V[1..m]: table which indicates different views (objects) associated with places of the petri net.

Proposed algorithm The execution of the petri net is described by the algorithm presented by algorithm 1. This algorithm assumes a petri net representing a workflow and a global security policy as inputs. It securely executes the process by generating local contextual policies. These local policies are associated to different places of the petri net.

The algorithm starts by initializing all execution conditions to false since no operation is being executed until now. Also, each context contains initially only the place to which it is associated. The processing of the first place is done separately. Indeed, the test for all other places processing is done on transition. So, if the first transition is valid (it contains at least one token) we check if the token of this place verify the global security policy defined as an input of the algorithm. In other words, for a token <r_k, v_k, Cont(P_k)> we check if the role r_k has access to the object o_k in the organization Org[k] associated with the place P_k. Then, we apply the coordination policy according to the case presented when calling state(). This policy is presented in table 2. So, to each case presented in the Cont(P_k) returned by state(i, k), we have to apply the set of rules (permissions and obligations) associated to it in table 2. If the global policy is checked, we pass to the local policy. So, we check if Cont(P_k) contains the place P_k. If it is the case, the local policy is activated. Thus, r_k can execute activities associated with P_k using the view v_k. After finishing execution, the local policy is deactivated by subtracting P_k from the set Cont(P_k). Then, we have to update (1) execution condition of transitions in relation with P_k to true, (2) transitions in relation with P_k if all conditions associated to them are true, (3) contexts of places following P_k in the order of execution, this is done by calling the function state(), (4) the marking of the petri net by removing <r_k, v_k, Cont(P_k)> from the set M(P_k). This processing is repeated for all places of the petri net. So, using this algorithm we can ensure a secure execution of the process. The algorithm uses two functions. The algorithms of these two functions are given by algorithm 2 and 3.

The algorithm consider two bases of security rules:

1. Static base of security rules (global security policy): security rules using a default context, without taking into account conditions or circumstances of execution and coordination security rules. It represent a security aspect. It defines permissions of roles on objects within an organization. It is an input of the algorithm.

2. Dynamic base of security rules (local security policy): generated security rules according to workflow execution, so depending on contexts generated or built during workflow execution. It present a security and a functional aspect. It is an output of the algorithm. It is dynamic during workflow execution. To add a contextual security rule to this base we must verify or check if there is a corresponding non contextual rule to this rule in the static base.

To have a global view of the workflow specification, we can refer to the dynamic base of security rules. Security rules contexts present a historic of different tasks executed before each task. So, grouping whole contexts we can redesign the workflow scheme. Since these contexts express different temporal relations between workflow tasks, this design is simple and logic to be reconstructed.

The algorithm does not affect initial properties of the petri net representation. Indeed, the petri net modelling the workflow preserves its properties of reachability, liveness and boundedness. These properties can be studied using matricial representation of the petri net marking and graph theory. In complexity terms, the algorithm has a polynomial execution time (in O(n^3)).

5.3 Example

To clarify the algorithm, we consider an application represented by the petri net of figure 3. This application include four tasks that must be executed in a specific order. To each task T_i we associate an execution interval I_i: I_1 = [1, 3], I_2 = [3, 5], I_3 = [3, 6], I_4 = [6, 8] . Using these intervals we define different relations between tasks: (1) $T_1 \mid T_2$, (2) $T_1 \mid T_3$, (3) $T_2 //_< T_3$, (4) $T_2 |_< T_4$ and (5) $T_3 \mid T_4$. The application is defined within an organization org and includes four roles (R_1, R_2, R_3 and R_4) and four views(V_1, V_2, V_3 and V_4).

We define our input static global policy of this application in table 3. The execution of the algorithm generates a local dynamic policy presented in table 4. This policy is the output of our proposed algorithm during the process execution.

We remember that in our security policy obligations are prior than permissions. So there is no conflict that can be generated.

We remember that the colour defined in the context (in this case "red") is used to identify the process execution. Each new execution process is defined using a new colour.

Algorithm 1: Main Algortihm

```
for i in [1 ... |T|] do
    /* initialisation of conditions of execution */
    $C_{ex}$ ($T_i$) ← false
end
for i in index (P) do
    /* initialisation of contexts */
    Cont($P_i$) ← { $P_i$ }
end
/* processing for the first place */
if ($|^{o} T_1| = 0$ & M($P_1$)<>{} & $P_1 \subseteq$ Cont($P_1$)) then
    /* local policy */
    SR (permission, Org[1], R[1], V[1], A[1], Cont($P_1$));
    Execute (A[1], result, $I_1$);
    for j in index ($T_1^o$) do
        /* validation of execution associated with $P_1$ */
        if (result = true) then
            $C_{ex_1}$ ($T_j$) ← true;
        end
        if ($cond_j$ = true) then
            Valid (j) ← true;
        end
    end
    for k in index ($P^o_{index(T_1^o)}$)) do
        Cont($P_k$) ← $P_1$;
        Cont($P_1$) ← Cont($P_1$) \ {$P_1$};
    end
end
repeat
    Valid_transition (bool, i);
    if (bool = true) then
        for k in index ($P_i^o$) do
            State (i, k);
            /*Coordination policy*/
            Apply ( Coordination Policy(Cont ($P_k$)));
            /*local dynamic policy*/
            SR (permission, Org[k], R[k], V[k], A[k],
            Cont ($P_k$));
            /*updating RdP marking*/
            M($P_k$) ← M($P_k$) ∪ <$r_k$, $v_k$, Cont($P_k$)> ;
            if (M($P_k$)<> {} & $P_k \subseteq$ Cont($P_k$)) then
                Execute ($a_k$, result, $I_k$);
            end
            for j in index ($T_k^o$) do
                if (result = true) then
                    $C_{ex_k}$ ($T_j$) ← true;
                end
                if ($cond_j$ = true) then
                    Valid (j) ← true;
                end
            end
            Cont($P_k$) ← Cont($P_k$) \ {$P_k$};
            M($P_k$) ← M($P_k$) \ {<$r_k$, $v_k$, Cont($P_k$)>};
            /* updating RdP marking */
        end
    end
until (bool = false) ;
```

Algorithm 2: Function valid_transition(bool, i)

```
/* searches for a valid transition in the Petri net
and returns its index */
Variables: bool: Boolean; j: integer
Bool ← false;
j ← 1 ;
while (bool = false & j < |T|) do
    if (valid [i] = true) then
        bool ← true ;
        i ← j ;
    j ← j + 1 ;
Return (bool, i)
```

Algorithm 3: Function State(i, k)

```
/* builds the context according to the execution
order of different places */
Variables: j: integer
for j in index ($P^o_{index(T_k^o)}$) do
/* recursive Construction of the context */
if Equal ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) //= {$P_k$}
if Starts ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) //< {$P_k$}
if Finishes ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) //> {$P_k$}
if During ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) //⊂ {$P_k$}
if Overlap ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) //<> {$P_k$}
if Before ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) |< {$P_k$}
if Meets ($I_i$ , $I_k$) then
    Cont($P_k$) ← Cont($P_i$) | {$P_k$}
return ($P_k$)
```

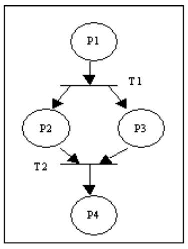

Figure 3: Petri net application

Global security policy
General security policy
P(org, R_1, execute, V_1, default)
P(org, R_2, execute, V_2, default)
P(org, R_3, execute, V_3, default)
P(org, R_4, execute, V_4, default)
Coordination security policy
P(begin(T_1), default)
O(begin(T_2), done(T1) ∧ start(T_3))
O(begin(T_3), done(T_1) ∧ start(T_2))
P(begin(T_2), default)
P(begin(T_3), default)
O(end(T_2), doing(T_3))
O(begin(T_4), complete(T_2) ∧ done(T_3))

Table 3: Algorithm input policy.

Local and dynamic security policy
P(org, R_1, execute, V_1, (red, P_1))
P(org, R_2, execute, V_2, (red, P_1 | P_2))
P(org, R_3, execute, V_3, (red, P_1 | P_3))
P(org, R_4, execute, V_4, (red, P_1 | (P_2 $//_{<}$$P_3$) | P_4))

Table 4: Algorithm output policy.

6 Conclusion

In this paper, we have presented a petri net based model for moddeling workflows and we have defined the security policy that we associate to it. This model and security policy are based on OrBAC model. Thus, they reuse organization and context notions given in this access control model. Our security policy takes into account different possible execution modes of two tasks. It is composed of a general security policy and a coordination security policy. In a second part, we have presented an algorithm allowing us to synchronize authorization flows with workflow execution. This algorithm define how to execute the suggested model in a *distributed* WFMS environment. As part of future work, we will enrich our algorithm by handling information flows between different organizations. Indeed, organizations must exchange flows to have knowledge of what is happening globally in the system. These flows must be managed in order to keep a secure execution environment of the process. In fact, exchanging flows between organizations may imply a confinement problem (Boebert, 1996, 1985). Thus, these exchanges have to be controlled in order to keep a secure environment of execution processes.

References

A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel and G. Trouessin (2003), Organization Based Access Control. IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Come, Italy.

N.R. Adam, V. Atluri and W-K. Huang (1998), Modeling and Analysis of Workflows Using Petri Nets. Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Volume10.

V. Atluri, W. Huang (1996), An authorization Model for Workflows. Proceedings of the Fifth European Symposium on Research in Computer Security, Rome, Italy, pages 44-64.

E. Bertino, E. Ferrari, V. Alturi (1999), The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. ACM Transactions on Information and System Security.

William E. Boebert, Richard Y. Kain (1996), A further Note on the Confinment Problem. Proceedings of the IEEE 1996 International Carnahan Conference on Security Technology. New York: IEEE Computer Society.

W. E. Boebert, R. Y. Kain, W. D. Young (1985), The extended Access Matrix Model of Computer Security. ACM Sigsoft Software Engineering Notes, vol 10(4).

F. Cuppens and A. Miège (2003), Modelling contexts in the Or-BAC model. 19th Annual Computer Security Applications Conference, Las Vegas.

F. Cuppens, N. Cuppens-Boulahia, and A. Miège (2004), Inheritance hierarchies in the Or-BAC model and application in a network environment. Second Foundations of Computer Security Workshop (FCS'04). Turku, Finlande.

F. Cuppens, N. Cuppens-Boulahia, T. Sans and A. Miège (2004), A formal approach to specify and deploy a network security policy. Second Workshop on Formal Aspects in Security and Trust (FAST). Toulouse, France.

René David, Hassane Alla (1992), *Du grafcet aux réseaux de Petri*, Hermès. Paris, 2nd edition.

W. Huang, V. Atluri (1999), SecureFlow: A Secure Web-enabled Workflow Management System. Proceedings of the fourth ACM workshop on Role-based access control, p.83-94, Fairfax, Virginia, United States.

P. Hung, Karlapalem (1999), A Secure Workflow Model. AISW (Australian Information Security Workshop.

James F. Allen (1993), Towards a General Theory of Action and Time. Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society.

R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman (1996), Role-Based Access Control Models. Computer, volume 29-Issue 2.

Workflow Management Coalition (1995), The Workflow Reference Model. WFMC-TC-1003.

Workflow Management Coalition (1998), Workflow Security Considerations. White Paper. Document number WFMC-TC-1019, Document Status - Issue 1.0.

F. Cuppens, N. Cuppens-Boulahia et T. Sans, (2005), Nomad: A Security Model with Non Atomic Actions and Deadlines . 18th IEEE Computer Security Foundations Workshop (CSFW'05), Aix-en-Provence, France.