

Managing access and flow control requirements in distributed workflows

Samiha Ayed, Nora Cuppens-Boulahia, and Frédéric Cuppens

ENST-Bretagne, Cesson Sevigne 35576, France

Abstract. Workflows are operational business processes. Workflow 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. Therefore, they do not deal with flow control, a very important requirement in WFMS. In this paper, we suggest a decentralized and dynamic approach to handle a security policy in workflows taking into account access and flow control.

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

1 Introduction

A workflow is a process consisting of many tasks. 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. On the one hand, 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 the 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 execution progression of the workflow. In addition, the execution of a task is related to the execution of precedent tasks. So an access control model is needed. On the other hand, workflow tasks can be executed within the same organization or within different ones. In the first case, the workflow security is maintained by the same organization which has the authority to manage the security of its different roles and their interactions and to control the access to its different objects. In the second case, a more rigorous treatment must be done to ensure a secure execution of the workflow. So, security measurements are not limited to the access control which must be applied but also it concerns the need to apply a flow control to manage interactions between different components of the WFMS. If an information *Inf* pass from a component C_1 to another component C_2 of the same organization or a different one, we must be sure that C_2 has privilege allowing it to accede to *Inf*. If C_2 must not accede to *Inf* and C_1 passes the information *Inf* to C_2 , a leakage of information is happened. The problem is more important if the transferred data is confidential or has a certain sensitivity degree in general. Such problem is known as the "confinement problem" [18, 17] defined as a leakage of data. In order to resolve such a problem, flow control models have to be used. So, a workflow specification must be correlatively defined with a security policy which takes into account an access and an information flow control simultaneously. Several proposals [2, 9, 10, 14] tried to address this problem and proposed different access control models to manage security in WFMS without taking into account information flow control. Their propositions consist in (1) specifying the global workflow security policy (an access control 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 [11] which only provides means to specify static security requirements, they present some limitations. Further that they present static approaches, they do not care about information flow control when defining their policy.

In this paper, we suggest a different approach to manage security in WFMS. Our security policy associated to the workflow execution is specified using OrBAC model [1]. 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 interesting 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 [4]. 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. 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. The *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 history of execution of precedent tasks. Also, it permits the definition of a dynamic security policy according to contexts, a very useful requirement in WFMS.

So, using OrBAC concepts we present a petri net based model to represent workflows. Then, we define the global WFMS security policy that we have to associate to the workflow model. A such policy deals with access and information flow control. It is based on OrBAC rules. These rules are enriched to be able to express information flow control. Afterwards, we show how to manage this WFMS 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 to the execution of each task that composes the workflow. The global policy and the petri net model are provided

as inputs of the algorithm. Our approach remedies to the static and centralized aspect of models already proposed.

The paper is organized as follows. Section 2 introduces OrBAC model. Section 3 defines our WFMS petri net based model. This model provides means to specify fine-grained execution modes of tasks using different temporal constraints. Section 4 specifies our security policy which we must associate to the workflow to ensure a secure execution environment. Section 5 addresses the issue of distributed workflow execution and defines an algorithm to generate local policies required to execute the workflow in a secure distributed way. The algorithm is discussed through an example. We present a short comparison with related works in section 6. Finally, section 7 concludes the paper and outlines future work.

2 OrBAC in brief

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

In order to specify a security policy, the OrBAC model [1] 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 modeling 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 [3]. The security rules do not apply statically but their activation may depend on contextual conditions [4]. Thus, the concept of *context* is explicitly included in OrBAC. Contexts are used to express different types of extra conditions that control activation of rules expressed in the access control policy. So, using formalism based on first order logic, security rules are modeled using a 6-places predicate.

Definition 1: *an OrBAC security rule is defined as: security_rule (type, organization, role, activity, view, context) where type \in {permission, prohibition, obligation}.*

An example of this security rule can be: security_rule (permission, a_hosp, nurse, consult, medical_record, urgency) meaning that, in organization a_hosp, a nurse is permitted to consult a medical record in the context of urgency.

3 Modelling WFMS

3.1 Petri net based model

Our WFMS model is based on petri nets [12] 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 2: *a Petri net is defined as a 5-tuple, $PN = (P, T, F, W, M_0)$, where: (1) P is a finite set of places, (2) T is a finite set of transitions, (3) $F \subseteq (P \times T) \cup (T \times P)$: a set of arcs from places to transitions and from transitions to places, (4) $W : F \rightarrow \{1, 2, 3, \dots\}$: a weight function, it defines weights assigned to different arcs, (5) $M_0 : P \rightarrow \{0, 1, 2, \dots\}$: the initial marking, it describes initial place contents.*

Definition 3: *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 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 to transitions.*

The model that we suggest uses interpreted petri nets. This choice is motivated by the requirements of processes we want to represent. In fact, choosing interpreted petri nets is related to the operative part of the process execution. Therefore, we assign operations or tasks to different places of the petri net. So, we need an operative part in the petri net representation. On the other hand, we need synchronized petri net to manage information flow control.

3.2 Constraints requirements

A workflow specification is correlatively defined with a tasks execution order or with temporal constraints of different tasks. In our WFMS model, we do not want to restrict execution modes of tasks to sequential and parallel execution. Instead we take into account different temporal constraints that can be present between two tasks. The table 1 defines these atomic temporal constraints.

Symbol	Temporal constraint
$\top(T_i, T_j)$	T_i must begin with T_j
$\nexists(T_i, T_j)$	T_i must begin after T_j
$\perp(T_i, T_j)$	T_i must end with T_j
$\pm(T_i, T_j)$	T_i must end before T_j

Table 1. Different temporal constraints between two tasks.

Using combination of these different cases we can reconstruct different intervals relationships present in [6] which defines complete set of execution modes of two tasks having each an execution time interval. As an example of these Allen's cases, let us consider T_1 and T_2 two tasks having respectively an execution interval I_1 and I_2 . If we suppose that these intervals must have an *Equal* relation, this is

can be expressed as follows: $\text{Equal}(I_1, I_2) = \pm(T_i, T_j) \wedge \mp(T_i, T_j)$. These constraints are taken into account further in our policy.

3.3 WFMS petri net model

In this section, we use 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 and to be easy further to express our security policy using this structuring. Then, such compatible 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 to places. An activity can be an atomic operation or a composed operation. A place can be validated only if all operations associated to it are executed.

Organization notion within the petri net formalism: The concept of organization is associated to 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 ones 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 in the system.

Context notion within the petri net formalism: The notion of contexts is also associated to places. OrBAC defines several categories of contexts [4]. 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 to 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 to a context $\text{same_subject}(P_1)$ to constrain subjects who are executing activities assigned to place P_1 and P_2 to be the same. Temporal constraints defined in section 3.2 can be expressed in a temporal context to define relation between different task execution.

Also, the *Provisional context*, that depends on previous actions the subject has performed in the system, is relevant in WFMS. In fact, in a workflow execution, a task execution depends on the execution history of precedent tasks. Hence, to each place we can associate a context. It represents the context of execution of operations associated to this place. It can be defined in conjunction with prerequisite and temporal contexts (see section 3.2).

Definition 5: *If P_i is a place of a petri net model, its context $Cont_i$ is defined as a triple:*

$$Cont_i = (Pre_context, temp_context, exec_context).$$

The first context contains conditions related to prerequisite context. The second one deals with temporal constraints defined between different tasks. The third context concerns the execution context. It defines requirements associated to P_i execution. It indicates precedent tasks that must be executed before the execution of operations associated to P_i .

However, in our model, we suppose that, when defining the security policy 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 policy, the execution context of a place is only containing the place itself. Then it will be enriched.

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

When the process starts its execution, contexts change with it. These changes are explained in the algorithm proposed in section 5.

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 next section. So, a system manager must define a petri net structuring of its WFMS application according to our proposed model approach.

4 WFMS global security policy

Once the system manager has defined the petri net modelling its WFMS application, it is time to define the security policy that he must associate to the model. To ensure a secure execution environment of the workflow, we must take into account three aspects:

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.
3. We have to deal with information flow control.

These aspects could make our security policy more useful and increases assurance that it is correctly specified. So a system manager must have as inputs, his petri net modelling and the global policy associated to it in order to generate a dynamic policy that it is updated with the workflow execution progression. Tasks execution are dynamically synchronized with our defined global policy using the dynamic policy.

4.1 Access control policy

Our access control policy is premised on a general security policy to manage access to different system objects and on a coordination security policy to enforce temporal constraints and execution modes of different tasks.

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 3.3). Thus, let us consider the petri net of figure 1. A security rule defined as a permission granted to a role R_3 within the organization Org to execute an activity A_3 associated to the place P_3 by the same subject that has executed A_1 and must end before A_2 in this petri net and using a view V_3 will be expressed as follows:

SR (permission, Org, R_3 , A_3 , V_3 , (same_subject(A_1), $\pm(A_3, A_2)$, P_1)).

We have already supposed that our initial execution contexts contain just the place it self. It is not necessary to explicitly specify the execution context associated to 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 and prerequisite contexts defined in OrBAC.

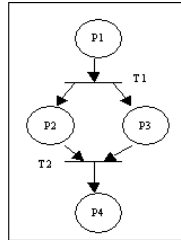


Fig. 1. Petri net application

WFMS coordination security policy The general security policy manages access control but it does not deal with different tasks execution modes. So to complete it we propose a coordination security policy which ensures a secure environment to execute workflows. This coordination security policy is collateral to our general security policy. This policy controls different temporal constraints 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 [4]. To

express our coordination contexts, we reuse predicates defined in nomad model [5]: $start(T)$ ("starting T"), $doing(T)$ ("doing T") and $done(T)$ ("finishing T"), where T is a task.

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 temporal constraints. To ensure these execution orders, we are led to use not only permissions to control our workflow execution but also obligations imposed by temporal constraints. We present our coordination policy in table 2. This policy makes explicit different temporal constraints 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 `temp_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.

4.2 Flow control security policy

As we have presented, OrBAC is an efficient model to express access control rules. So, it will be very useful if we succeed to express access and flow control using the same model. DTE (Domain and Type Enforcement) formalism developed in [13] shows and details how to use OrBAC rules to express either access control rules or information flow control rules. We reuse this reflection to define our flow control security policy. We suppose, due to space limitation, that this policy is closed meaning that all which is not permitted is denied. Our flow control policy is based on the entry point notion. Thus, we recall the definition. Let us consider the following definitions:

Definition 6: (*domain*) S is a set of all active entities of a system. S is divided into equivalence classes. Each class represents a domain D including a set of subjects having the same role in the system.

Definition 7: (*Entry Point*) An entry point is a program which must be executed to pass from a domain D_1 to a domain D_2 , denoted $EP(D_1, D_2)$ or $EP_{1,2}$. An entry point implies the following rule:

$\text{Auth}(s, D_1, \text{priv}) \wedge \text{execute}(s, \text{EP}(D_1, D_2)) \leftarrow \text{Auth}(s, D_2, \text{priv}') \wedge \text{!}=(\text{priv}', \text{priv}) \wedge \text{not Auth}(s, D_1, \text{priv})$.

Where s is the subject who wants to transit from D_1 to D_2 , priv is its privilege in D_1 and priv' the one of D_2 .

A domain in DTE has the same significance as a role definition in OrBAC. An entry point manages the transition between different domains and so it controls information flows within the system. Thus, if a subject wants to transit from a domain D_1 to another domain D_2 , he must execute an entry point denoted $\text{EP}(1,2)$ or $\text{EP}_{1,2}$.

temporal constraints	Associated Policy
$T_i \mid T_j$: T_j must begin after T_i end	$- P(\text{begin}(T_i), \text{default})$ $- O(\text{begin}(T_j), \text{done}(T_i))$
$T_i \top T_j$: T_i must begin with T_j	$- P(\text{begin}(T_i), \text{default})$ $- P(\text{begin}(T_j), \text{default})$ $- O(\text{begin}(T_i), \text{start}(T_j))$ $- O(\text{begin}(T_j), \text{start}(T_i))$
$T_i \perp T_j$: T_i must end with T_j	$- P(\text{begin}(T_i), \text{default})$ $- P(\text{begin}(T_j), \text{default})$ $- O(\text{end}(T_i), \text{done}(T_j))$ $- O(\text{end}(T_j), \text{done}(T_i))$
$T_i \mp T_j$: T_i must begin after T_j	$- P(\text{begin}(T_j), \text{default})$ $- P(\text{begin}(T_i), \text{doing}(T_j))$
$T_i \pm T_j$: T_i must end before T_j	$- P(\text{begin}(T_j), \text{default})$ $- P(\text{end}(T_i), \text{doing}(T_j))$

Table 2. WFMS coordination security policy.

This entry point consists in a program or an activity. It defines the set of privileges that the subject will obtain. All privileges of the source domain will be lost. Based on OrBAC rule, an information flow control rule is defined as follows:

(1) the source domain is considered as the role in the OrBAC rule (we have already said that the two notions have equivalent significance), (2) the destination domain is considered as the view in OrBAC rule, (3) the transition between two domains can be expressed as an OrBAC activity since the basic role of this specific rule is to handle interactions between domains. So, we define the OrBAC activity as an **Enter** activity, (4) the entry point defines the manner to enter a domain. Thus, we can consider it as a condition of rule validation. Therefore, an entry point can be defined as a spe-

cific context in an OrBAC rule denoted **through**($E_{i,j}$). This context specifies that the rule is valid only through $E_{i,j}$ execution.

Thus, such a rule is expressed, in a specific organization *org* and handling transition between D_1 and D_2 , as follows:

SR (*permission*, *org*, D_1 , **Enter**, D_2 , **through**($E_{1,2}$)).

If there is no need to execute any entry point to transit to another domain, *auto* mode is used instead of the **through**($E_{1,2}$) context. Thus the rule will be:

SR (*permission*, *org*, D_1 , **Enter**, D_2 , *auto*).

These flow control rules can be enriched with other OrBAC contexts. Indeed, **through**($E_{i,j}$) context can be used in conjunction with different OrBAC contexts, for example temporal contexts, to express more restrictive or conditioned flow control. Also, We recall that a transition from a domain D_1 to a domain D_2 is possible only if there is the corresponding rule in the policy. In other words, handling domains interactions does not contain prohibitions. This interaction is allowed only if there is a corresponding permission. Such transitions between domains correspond to a context change. Since transiting to another domain corresponds to the activation of a new context, new rules will be activated. These rules are those for which this context is valid. This dynamic management of the security policy and the closed policy hypothesis guarantee the loss of source domain privileges during transition. New granted privileges are defined according to the entry point executed. The entity *organization* is useful to control the flow in the inter-organizational environment. This will be developed in a forthcoming paper.

5 Deploying WFMS security requirements

5.1 Decentralized control

Section 4 showed how to specify the workflow security policy as a set of OrBAC security rules. We apply access control rules to places and information flow control rules to transitions in our WFMS petri net model. 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 no 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 inputs the petri net description of the workflow and the global security policy associated to 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 to 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 according to contexts. This policy synchronizes the global policy with the execution of the petri net. So, access to different system objects and temporal constraints 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. Our security policy does not only deal with access control but also with information flow control. Our information flow control is based on transitions of our WFMS petri net model. Since we use interpreted petri nets, we can synchronize transitions to events. In our model, we consider that these events correspond to the activation of information flow control rules. A transition relates two places. If operations associated to these two places have to be executed by roles R_1 and R_2 respectively and by the same subject, a transition from role R_1 to R_2 is needed. This transition is controlled using our information flow rules associated to transitions. Thus a transition in our WFMS petri net will be valid or activated if and only if all operations associated to its entry places finish and the information flow control rule associated to it is applied.

In figure 1, the activity A_3 has to be executed by the same subject who has executed A_1 . So, a transition from R_1 to R_3 is needed. Such a transition is managed using the following control flow rule:

SR (permission, org, R_1 , Enter, R_3 , Through(/etc/logind)).

This rule is associated to the transition T_1 of the petri net.

5.2 Algorithm to deploy WFMS security requirements

To define our algorithm, we first introduce some 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.

Hypothesis: The initial marking of the petri net is a token in the first place. This comes from the definition of a workflow.

Variables: (1) i, j, k, n, m : integer, (2) Bool, result : Boolean (3) $\text{Cont}(P_i)$: context of the place P_i (4) A token $\langle \Gamma_k, v_k, \text{Cont}(P_k) \rangle$, $\text{Cont}(P_k) = (\text{Pre_context}(P_k), \text{temp_context}(P_k), \text{exec_context}(P_k))$

Functions: (1) $\text{Card}(E) = |E|$: returns the cardinality of the set E . (2) $M(P_j)$: returns the set of tokens of the place P_j . (3) $\text{Index}(E)$: returns index of different elements of the set E . (4) $\text{Valid_transition}(\text{bool}, i)$: function which verifies if the petri net contains a valid transition. In this case, bool will be true and i will return its index.

Sets: (1) P : set of places (2) T : set of transitions (3) $|P| = m$ and $|T| = n$ (4) oP_j : set of input places of the transition T_j (5) P_j^o : set of output places of the transition T_j (6) oT_j : set of input transitions of the place P_j (7) T_j^o : set of output transitions of the place P_j (8) $\text{Cond}_j = \text{cond}(T_j) = \{C_{ex}, C_1, \dots, C_k\}$: set of conditions associated with the transition T_j where: $C_{ex} = \{C_{ex_i}, C_{ex_j}, \dots\}$: set of execution conditions of input places of the transition T_j . $\text{Cond}_j = \text{true}$ 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) We use tables Org, R, A and V to represent respectively organizations, roles, activities and views used in the process.

Proposed algorithm The execution of the petri net is described in 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 transitions. So, if the first transition is valid (it contains at least one token) we check if the token of this place verifies the global security policy defined as an input of the algorithm. In other words, for a token $\langle r_k, v_k, \text{Cont}(P_k) \rangle$ we check if the role r_k has access to the view v_k in the organization Org_k associated to the place P_k . After that, we construct the security local policy containing rules which manage execution of operations associated to each place. So, we check if $\text{exec_context}(P_k)$ contains the place P_k . If it is the case, the local policy is activated. Thus, r_k can execute activities associated to P_k using the view v_k . After finishing execution, the local policy is deactivated by subtracting P_k from the set $\text{exec_context}(P_k)$. Later, we apply the coordination policy according to policies defined in section 4.1. These policies are presented in table 2. So, to each case presented in the $\text{temp_context}(P_k)$, we have to apply the set of rules associated to it in table 2. 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, (4) the marking of the petri net by removing $\langle r_k, v_k, \text{Cont}(P_k) \rangle$ from the set $M(P_k)$. The next transition releases according to information flow control rules associated to it. This processing is repeated for all places of the petri net. So, using this algorithm we can ensure a secure execution of the process.

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 $O(n^3)$).

Example To exemplify the algorithm, let us reconsider the figure 1. We suppose that the activity A_3 associated to the place P_3 by the same subject that has executed A_1 and must end before A_2 . Table 3 represents the global security policy that we associate to the petri net modelling of the application. Rules 1-4 represent the general policy. Rules 5 and 6 are coordination rules, since there is a condition on executing A_3 . As A_3 has to be executed by the same subject who had executed A_1 , a transition between R_1 and R_2 is needed. This transition is managed by rule 7 which is associated to the transition T_1 of the petri net. It needs the execution of `/etc/logind`. Rule 8 means that the transition from R_1 to R_2 needs no program to be executed. The policy generated by the execution of the proposed algorithm is presented in table 4.

1	$P(\text{org}, R_1, \text{execute}(A_1), V_1, \text{default})$	5	$P(\text{begin}(A_2), \text{default})$
2	$P(\text{org}, R_2, \text{execute}(A_2), V_2, \text{default})$	6	$P(\text{end}(A_3), \text{doing}(T_2))$
3	$P(\text{org}, R_3, \text{execute}(A_3), V_3, \text{default})$	7	$P(\text{org}, R_1, \text{Enter}, R_3, \text{Through}(\text{/etc/logind}))$
4	$P(\text{org}, R_4, \text{execute}(A_4), V_4, \text{default})$	8	$P(\text{org}, R_1, \text{Enter}, R_2, \text{auto})$

Table 3. Algorithm input policy.

1	$P(\text{org}, R_1, \text{execute}(A_1), V_1, (-, -, -))$
2	$P(\text{org}, R_2, \text{execute}(A_2), V_2, (-, -, P_1))$
3	$P(\text{org}, R_3, \text{execute}(A_3), V_3, (\text{same_subject}(A_1), \pm(A_3, A_2), P_1))$
4	$P(\text{org}, R_4, \text{execute}(A_4), V_4, (-, -, (P_1, P_2, P_3)))$

Table 4. Algorithm output policy.

Algorithm 1: Main Algorithm

```
1 for i in [1 ... |T|] do
2   | Cex(Ti) ← false /* initialisation of execution conditions */
3 end
4 for i in index (P) do
5   | exec_context(Pi) ← { Pi } /* initialisation of contexts */
6 end
7 /* processing for the first place */
8 if (permission, Org[1], R[1], V[1], A[1], (Pre_context(P1), temp_context(P1), -)) ∈ SB then
9   | DB ← DB ∪ (permission, Org[1], R[1], V[1], A[1], (Pre_context(P1), temp_context(P1),
   |   exec_context(P1)))
10 end
11 if (|oT1| = 0 & M(P1) <> {} & P1 ⊆ exec_context(P1)) then
12   | Execute (A[1], result)
13 end
14 for j in index (T1o) do
15   | if (result = true) then
16     |   Cex1(Tj) ← true; /* execution validation associated to P1 */
17     |   Apply(flow control policy(Tj)); /* information flow control rules */
18   end
19   | if (condj = true) & valid (flow control policy(Tj)) then
20     |   Valid (j) ← true;
21   end
22 end
23 for k in index (P1oindex(T1o)) do
24   | exec_context(Pk) ← exec_context(Pk) ∪ P1;
25 end
26 exec_context(P1) ← exec_context(P1) \ {P1};
27 repeat
28   Valid_transition (bool, i); /* it returns a valid transition in the petri net and its index i*/
29   if (bool = true) then
30     for k in index (Pio) do
31       if (permission, Org[k], R[k], V[k], A[k], (Pre_context(Pk), temp_context(Pk), -))
32         ∈ SB then
33         | DB ← DB ∪ (permission, Org[k], R[k], V[k], A[k], (Pre_context(Pk),
34         |   temp_context(Pk), exec_context(Pk))) /* local dynamic policy */
35       end
36       /*updating RdP marking*/
37       M(Pk) ← M(Pk) ∪ <rk, vk, Cont(Pk)>; /* Cont(Pk) = (Pre_context(Pk),
38       |   temp_context(Pk), exec_context(Pk))
39       if (M(Pk) <> {} & Pk ⊆ Cont(Pk)) then
40         | Apply ( Coordination Policy(temp_context(Pk))); /*Coordination policy*/
41         | Execute (ak, result);
42       end
43       for j in index (Tko) do
44         if (result = true) then
45           | Cexk(Tj) ← true;
46           | Apply(flow control policy(Tj));
47         end
48         if (condj = true) & valid (flow control policy(Tj)) then
49           | Valid (j) ← true;
50         end
51       end
52       for i in index (Pkoindex(Tko)) do
53         | exec_context(Pi) ← exec_context(Pi) ∪ Pk;
54       end
55       exec_context(Pk) ← exec_context(Pk) \ {Pk}; /* updating RdP marking */
56       M(Pk) ← M(Pk) \ {<rk, vk, Cont(Pk)>;
57     end
58   end
59 until (bool = false) ;
```

6 Comparison with related works

Adam, Alturi and Huang [9, 14] 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. 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, Atluri, huang and bertino have used a convergence between RBAC and MLS to apply it to WorkFlow Management Systems. They define in [15, 16] models of WFMS based on petri nets and they define an RBAC security policy. They associate levels to different objects used in the system and so to tasks using these objects. Then, they apply the MLS approach to their model taking into account different task dependencies. But their approach is localized into a workflow execution fully secure and partially correct. In other words, the approach does not enforce all task dependencies. So it can affect functional workflow execution.

Our approach makes the difference of these works by proposing a dynamic management of workflow authorizations and by considering a security policy taking into account flow control. This control is more robust and more practise that one used by Bertino [16] since it is based on the same model OrBAC which is enriched to express simultaneously access and flow control. This model is more flexible and less constraining.

7 Conclusion

In this paper, we have presented a petri net based model of workflows and we have defined the security policy that we associate to it. This model and security policy are based on OrBAC concepts. Thus, they reuse organization and context notions defined in this access control model. Our security policy takes into account different possible temporal constraints between two tasks. It is composed of a general security policy, a coordination security policy and an information flow policy. In a second part, we have presented an algorithm allowing us to synchronize authorization flows with workflow execu-

tion. This algorithm defines 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. In fact, exchanging flows between organizations may implies a confinement problem [17, 18]. Thus, these exchanges have to be controlled in order to keep a secure environment of processes execution.

References

1. 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 Como, Italy.
2. 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.
3. F. Cuppens, N. Cuppens-Boualahia, 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.
4. F. Cuppens and A. Miège (2003). Modelling contexts in the Or-BAC model. 19th Annual Computer Security Applications Conference, Las Vegas.
5. F. Cuppens, N. Cuppens-Boualahia 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.
6. James F. Allen (1993). Towards a General Theory of Action and Time. Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society.
7. Jan Kiszka, Bernardo Wagner (2003). Domain and Type Enforcement for Real-Time Operating Systems. Proceedings ETFA '03, Emerging Technologies and Factory Automation.

8. Lee Badger, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, Sheila A. Haghghat (1995). Practical Domain and Type Enforcement for Unix. IEEE Symposium on Security and Privacy. Oakland, CA, USA.
9. N.R. Adam, V. Atluri and W-K. Huang. Modeling and Analysis of Workflows Using Petri Nets (1998). Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Volume10.
10. P. Hung, Karlapalem (1999). A Secure Workflow Model. AISW (Australian Information Security Workshop).
11. Ravi Sandhu (1996). Role Hierarchies and Constraints for Lattice-Based Access Controls. Proc. Fourth European Symposium on Research in Computer Security. Rome, Italy.
12. René David, Hassane Alla (1992). Du grafcet aux réseaux de Petri, Hermès. Paris, 2nd edition.
13. Samiha Ayed, Nora Cuppens-Boulahia, Frédéric Cuppens. An integrated model for access control and information flow requirements (2007). ASIAN'07, Doha, Qatar.
14. V. Atluri, W. Huang. An authorization Model for Workflows (1996). Proceedings of the Fifth European Symposium on Research in Computer Security, Rome, Italy, pages 44-64.
15. V. Atluri and Wei-Kuang Huang (1997). Enforcing Mandatory and Discretionary security in Workflow Management Systems. Journal of Computer Security, 5(4):303-339.
16. Vijayalakshmi Atluri, Wei-Kuang Huang, Elisa Bertino (2000). A semantic Based Execution Model for Multilevel Secure Workflows. Journal of Computer Security, 8(1).
17. 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).
18. 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.