Access Control for Monitoring System-Spanning Business Processes

Sarita Bassil[1]*, Manfred Reichert[2], Ralph Bobrik[3], Thomas Bauer[4]

[1]Dept. Comp. & Information Sciences, Holy Spirit University of Kaslik, Lebanon,
saritabassil@usek.edu.lb
[2]Information Systems Group, University of Twente, The Netherlands,
m.u.reichert@cs.utwente.nl
[3]Dept. Databases and Information Systems, University of Ulm, Germany,
bobrik@informatik.uni–ulm.de
[4]DaimlerChrysler Research & Technology, REI/ID
thomas.tb.bauer@daimlerchrysler.com

Abstract. Integrated process support is highly desirable in environments where data related to a particular (business) process are scattered over distributed and heterogeneous information systems (IS). A process monitoring component is a much-needed module in order to provide an integrated view on all these process data. Regarding process data integration, access control (AC) issues are very important but also quite complex to be addressed. A major problem arises from the fact that the involved IS are usually based on heterogeneous AC components. For several reasons, the only feasible way to tackle the problem of AC at the process monitoring level is to define access rights for the process monitoring component, hence getting rid of the burden to map access rights from the IS level. In this paper, we propose a set of requirements for AC in process monitoring, which we derived from our case studies in the automotive domain. We then present alternative approaches for AC: the view-based approach and the object-based approach. The latter is retained, and a core AC model is proposed for the definition of access rights that meet the derived requirements. AC mechanisms provided within the core model are key ingredients for the definition of model extensions.

1 Introduction

In order to streamline their way of doing business, today's companies are dealing with a number of processes involving different domains, organizations, and groups. As discussed in [1], an integrated process support is highly desirable in such an environment where data (e.g., audit trails and reports) related to a particular process (instance), and with different degrees of sensitivity, are often scattered over heterogeneous information systems (IS) (cf. Fig. 1). A process

* This work was done during a postdoctoral fellowship at the University of Twente. It has been partially funded by DaimlerChrysler Research & Technology.

monitoring component is a much-needed module in order to provide an integrated view on all these data. Despite its importance, many of current process-aware IS [2] do not offer such a component. Specifically, a process monitoring component is responsible for displaying the status of process instances, for dispatching specific activities to corresponding actors, and so on.

Different user groups or roles (e.g., technicians, managers) usually have different perspectives over processes and related data. Therefore, adequate views need to be provided. This is of particular importance when dealing with complex, long-running business processes with dozens up to thousands activities.

In the context of process data integration and process monitoring, access control (AC) issues are very important to be addressed. However, a major problem is that the involved IS are usually based on different AC components implying facts such as 1) heterogeneity regarding the meta-models based on which organizational models and related access rights are defined (e.g., users/groups and actors/roles), 2) different notions for the same entity/entity type (e.g., user and actor), and 3) non-registration of particular user(s) in all of the involved IS.

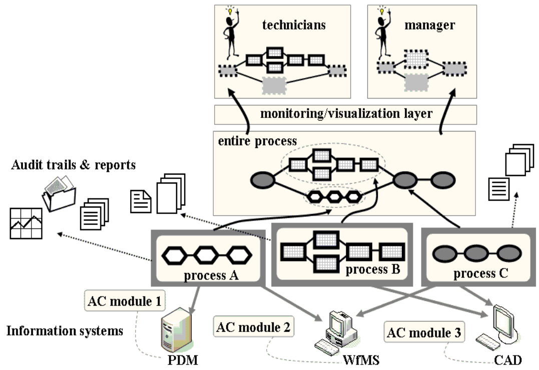

Fig. 1. Process Data Integration with Multiple Perspectives

In order to preserve integrity of AC information, AC constraints applied at the process monitoring level should be consistent with the constraints set out by the different IS. However, it has turned out that the integration of heterogeneous AC components is difficult to achieve for several reasons: 1) Access rights are not always explicitly described, but might be “hard-coded”, and hence difficult to retrieve; 2) AC modules do not always provide interfaces (i.e., APIs) in order to facilitate the access to information about AC rules (we talk about “black-box” AC modules); and 3) Rights at the IS level mainly deal with process definition and execution, and they have been not designed for the monitoring of process data by different users. Process definition and execution require administration rights, permissions to create new instances, delegation rights, and rights to work on specific activities. By contrast, process monitoring requires rights to visualize specific process activities, to display specific activity attributes, or to show different abstractions on a process (cf. Fig. 2a+b). Taking this into account, it appears that the only feasible way to tackle the problem of AC at the process monitoring level is to (re-)define AC rights for the process monitoring component, hence getting rid of the burden to inherit AC rights from the IS level. Of course, if possible, existing AC rights at the IS level should be automatically mapped to the ones at the process monitoring level, but we cannot assume this in general. Explicitly, specifying AC rights at the monitoring level also makes it possible to define them at a finer-grained level when compared with what is already defined at the IS level.

This paper discusses requirements relevant for the definition of such AC rights. These requirements have resulted from case studies we conducted in the automotive sector.[1] We propose approaches for AC, mainly a *view-based* and an *object-based* one. The retained solution (i.e., the object-based approach) is used as a backbone in order to provide a comprehensive core AC model. This model allows for the (compact) definition of AC rights at a fine-grained level. Moreover, AC rights are meant to meet the spectrum of confidentiality possibly defined on process data. Proposed AC mechanisms will be key ingredients in future definitions of extended AC models for process monitoring.

Sect. 2 discusses basic considerations by distinguishing between the model level and the instance level. It also gives a precise terminology of the concepts used later on in this paper. Sect. 3 exposes the major requirements identified. Two alternative approaches for AC are studied and compared in Sect. 4. The retained approach is well motivated. In Sect. 5, the logical AC model is introduced. Sect. 6 discusses related work and Sect. 7 gives a summary and an outlook.

2 Basic Considerations

In order to fix the basic framework of our research, and from a business process management perspective, we distinguish between the model and the instance level (cf. Fig. 2). The former gathers different kinds of enterprise models such as organizational models, functional models, data models, IT-system models, and process models. Each of the first four models gives input to the process model defined as a set of one or more linked activities (i.e., description of a piece of work), which collectively realize a business objective [2]. Specifically, these activities are carried out, in a coordinated way, by different processing entities (incl. humans and software systems) in order to reach a goal, such as changing the design of a car, delivering merchandise, or operating a patient. User-defined and pre-defined attributes may be associated with process models or activities (e.g., costs, needed resources).

In our Proviado project [1, 3], at the *model level*, we focus on the secure visualization of data related to a particular process model. Other kinds of models have not yet been considered for visualization but will be added later on. Different types of data may be involved in a process model such as process relevant data

[1] In the Proviado project [1, 3], we are aiming to propose a solution for visualizing in a secure way data related to a particular process or to a collection of processes.

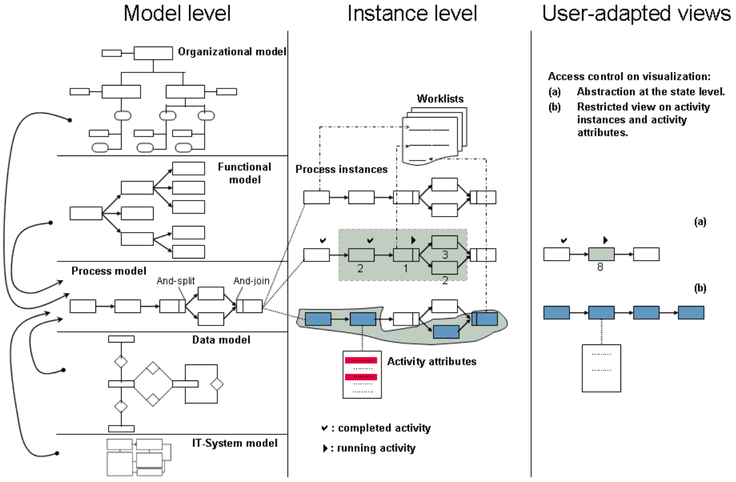

Fig. 2. Basic Considerations

and application data [4]. We are particularly interested in providing a secure way to visualize application data. These data are in general strictly managed by the application(s) supporting the process model. At the *instance level*, we focus on the secure monitoring of running process instances. A process instance is defined as the representation of a single enactment of a process model (i.e., a concrete business case) [2]. Concepts such as user worklists (i.e., lists of workitems derived from process instance activities), activity execution state (e.g., `Running`, `Completed`), and activity execution cost are associated with the instance level.

At the model and instance levels, different kinds of rights are to be defined: administration rights, data access rights, permission to create process instances from a given process model, rights to execute a particular work item, delegation rights, etc. At the model level (resp. instance level), the visualization (resp. the monitoring) of user-adapted views derived from specific process models (resp. process instances) is required. These views must take into account the access rights of the involved user. Access rights may be defined on different aspects related to the model and instance levels: process model, activity, process instance, activity instance, data elements, pre-defined and user-defined attributes, attribute current value, attribute history, etc.

3 Access Control Major Requirements

We investigated a number of case studies in the automotive domain from which we derived requirements as input for our work. Indeed, we studied different processes including *car engineering*, *change management* (cf. Fig. 3a) and *release management*. As the fruit of these case studies, we derived major requirements for AC in process monitoring.

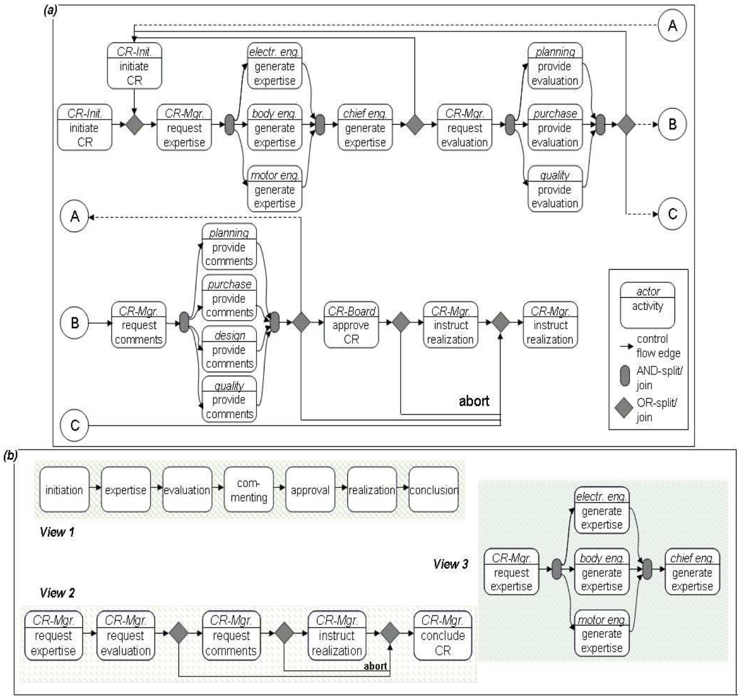

Fig. 3. Automotive Domain – (a) Simplified Process of Dealing with Change Requests (CR), (b) Different Views on CR Process

Requirement 1 (Definition of AC rights at a fine-grained level). AC rights for process monitorning should meet the spectrum of confidentiality defined on data related to a particular process (instance). Moreover, they should be definable on different aspects/objects of the model and instance levels (e.g., the process itself, the activities, the attributes, and the data elements).

- *Requirement 1.1 (Meeting a spectrum of confidentiality).* A distinction should be made between at least three levels of confidentiality: a first level in which all available information can be accessed, a second level where only high-level information can be accessed (i.e., abstraction), and a third level where no information is available at all. Considering the process of managing change requests (cf. Fig. 3a), for example, we may think about a (pre-defined) attribute (e.g., *activity cost*) associated with a specific activity (e.g., `generate expertise`). Such an activity may require a "two days by person" cost to be accomplished. One may have the right to access this information (i.e., the exact value of the attribute), to access abstracted information such as "less than one week (i.e., less than five days by person)", or to access nothing. The spectrum of confidentiality may also be restricted to only two levels: "give" or "don't give information". Regarding change management for example, an external partner may design part of the car; internally, a verification of this component may be done before it is integrated with the overall design of the car. The external partner might or might not have the right to know about the *existence* of the verification activities.

- *Requirement 1.2 (AC rights definable on different objects of the model / instance levels).* We define "object" as an entity of a process model or process instance respectively. For example, an *expertise document* produced as output of a `generate expertise` activity is considered as a data object. The `generate expertise` activity itself as well as the change request (CR) process model are considered as two different objects. Moreover, a group of objects is also an object. E.g., AC rights may be defined 1) on all running CR process instances, or 2) on specific CR process instances.

Requirement 2 (Definition of static AC rights). A differentiation is done between AC rights that are independent from the execution of a process instance ("static" AC rights), and those that depend on the execution of a process instance ("dynamic" AC rights). The latter are based on elements such as activity status and control principles (e.g., separation/binding of duties, dual control, and inter-case constraints) [5, 6]. Going back to our CR process, a person from a specific department (e.g., motor eng.) responsible for generating expertise might not be allowed to access the *expertise document* generated by the other departments (car body eng. and electronic eng.) unless she finishes generating her own expertise. In this paper, we focus on the definition of static AC rights.

Requirement 3 (Usability and maintainability of AC rights). AC rights should be simple to define and easy to maintain. As discussed in [7], a significant challenge is to balance collaboration and flexibility. It is to ensure that the advantages provided by collaborative systems, e.g., process-aware IS, are not reduced by AC rights too rigidly defined. For this purpose, abstractions are required at the objects' level. It is for example an obligation to define hierarchies on objects in order to specify AC rights at different levels of granularity. E.g., it might be reasonable to authorize a particular user (e.g., a manager) to access all running CR process instances. However, regular users might have access to specific CR process instances: a CR initiator may have the right to access only CR process instances that correspond to change resquests initiated by her.

Table 1 gathers major requirements identified. The ones highlighted (i.e., R1, R2, and R3) are addressed by the solution proposed in Sect. 5. Other requirements (R4 - R7) were identified, but are not considered in this paper.

Table 1. *Access Control Major Requirements*

Requirements	Requirements' description
R1	**Definition of AC rights at a fine-grained level**
	R1.1 Meeting a spectrum of confidentiality
	R1.2 AC rights definable on diff. aspects of the mod./inst. levels
R2	**Definition of static AC rights**
R3	**Usability and maintainability of AC rights**
R4	Definition of dynamic AC rights
R5	Definition of AC rights on the visualization of a collection of processes
R6	Definition of AC rights for the look-ahead problem
R7	Completeness of the AC component

4 Candidate Solution Approaches for Access Control

Among a list of possible approaches for AC, we feature two candidate solutions that we study and compare: the view-based and the object-based approach. In both approaches we follow the main idea proposed by a generalized approach to AC, that is RBAC (Role-Based Access Control) [8], in which AC rights are not directly linked to concrete users, but to roles. The *view-based* approach consists of defining one basic view per user role; this view implicitly reflects the AC rights of the role over a process by only showing the information to be accessed by users with the respective role. The *object-based* approach consists of defining, for each role, AC rights on the different aspects of a process (e.g., activity, activity attributes, process instance). We first illustrate each of the two featured approaches (Sect. 4.1). We then summarize the advantages and drawbacks of these approaches (Sect. 4.2). This helps us to clearly motivate the object-based approach as the solution approach retained and elaborated in the following.

4.1 Description of Solution Approaches

View-based Approach. Considering a particular process model such as the CR process (cf. Fig. 3a), a number of views could be (manually) defined on this process. Each of these views would then reflect the information accessible for users with a particular role. Access rights over the process may be derived implicitly from each view. Suppose the following views are defined on the CR process (cf. Fig. 3b): *(View 1)* High-level view on CR process, *(View 2)* View on `expertise` activities of CR process, and *(View 3)* View on `request` activities of CR process. Then one basic view per role may be defined: *("general manager", View 1)*, *("CR manager", View 2)*, and *("engineer", View 3)*. Each of the views implicitly reflects the read access rights of the particular role when visualizing/monitoring the CR process:

- A *general manager* may have access to high-level activities like `initiation`, `expertise`, `evaluation`, `commenting`, and so on.
- A *CR manager* may have access to activities `request expertise`, `request evaluation`, `request comments`, `instruct realization`, and `conclude CR`.
- An *engineer* may have access to concrete activities `request expertise` and `generate expertise`.

Object-based Approach. It consists of explicitly defining an extensible set of access rights for each role:

- *("general manager",* {`initiation`*,* `expertise`*,* `evaluation`*,* `commenting`*,* `approval`*,* `realization`*,* `conclusion`}*, Read)*
- *("CR manager",* {`request expertise`*,* `request evaluation`*,* `request comments`*,* `instruct realization`*,* `conclude CR`}*, Read)*
- *("engineer",* {`request expertise`*,* `generate expertise`}*, Read)*

A view may then be generated for a specific user based on the access rights associated with the role(s) played by this user. As an example, a view such as *View 3* illustrated in Fig. 3b would be generated for motor engineer *John Smith.*

4.2 Solution Approaches: Advantages and Drawbacks

We discuss the merits and shortcomings of these two approaches.

View-based Approach. The most obvious advantage comes from the fact that an existing concept (e.g., View Definition Language) can be explicitly reused in order to reflect the access rights over processes. Hence, there is no need for defining a new AC language (assuming that the process-aware IS clearly supports a View Definition Language). However, three drawbacks can be identified:

Costly maintenance of views: Considering a particular process model P together with the views derived from P. Suppose a modification is brought to P: (1) the views affected by this modification have to be identified possibly among a large number of existing views; (2) the identified views have to be adapted to reflect the modification brought to P. This adaptation should be done without any failure; (3) the modified views imply an implicit modification over AC rights.

Complexity of views combination: Since a single user may play more than one role (e.g., *John Smith* being a general manager as well as a motor engineer), this may lead to the necessity of combining multiple views (e.g., *View 1* and *View 3*). The resulting view, automatically generated or even manually modeled out of multiple views, will be shown to the user. On the one hand, we are facing a combinatorial problem (i.e., the different ways of arranging views in order to combine them). On the other hand, conflicts may exist between access rights reflected by the views to be combined. Such conflicts, first, must be detected, and second, be solved, probably by applying specific conflict resolution policies [9, 10] in order to correctly derive the combined view to be shown to the user.

Occurrence of redundant information due to lack of abstraction: Suppose that a specific role R has access, among other things, to a specific activity A in all processes involving this activity. Using the view-based approach, this access right would be reflected by showing activity A within all the views respectively defined on the processes involving activity A. This leads to redundant information due to the definition of access rights at the level of process models, not involving functional models (cf. Sect. 2), for example. The redundancy of information is an issue not only for the view-based approach, but for other approaches as well, as long as the notion of abstraction is missing (e.g., at the activities level). However, redundancy has more impact in conjunction with the view-based approach than in conjunction with the object-based one, since for the latter the definition of abstractions is easier to achieve (cf. Sect. 5.3).

Object-based Approach. The main advantage of this approach is threefold. Indeed, the drawbacks identified for the view-based approach appear to be advantages here. First, there is no maintenance of views; the cost behind the maintenance operation is abolished. Second, views have not to be combined and hence the complexity behind this operation does not exist. Third, if it is possible to define different levels of abstractions on objects, this will reduce redundancy when specifying access rights. The object-based approach may be criticized for not being intuitive since AC rights, instead of basic views, are initially defined for each role. However when compared with the drawbacks of the view-based approach, we voluntarily accept this only criticism, and select the object-based approach in order to elaborate the core solution for our logical AC model.

Taking into account the discussion of advantages and drawbacks, Table 2 gives a summary of the most important criteria that play either in favor of or against each of the considered approaches. As we can see, among five criteria, three criteria play in favor of the object-based approach, while only one criterion plays in favor of the view-based approach.

Table 2. *Comparison of the View-based and Object-based Approaches*

Criteria/Approaches	**View-based**	**Object-based**
Ease of AC rights definition	+	-
Ease of AC rights maintenance	-	+
Ease of conflicts resolution	-	-
Ease of AC rights combination	-	+*
Redundancy-free	-	+

+ Criterion plays in favor of the approach
- Criterion plays against the approach
* This criterion is reduced to the "Ease of conflicts resolution" criterion

5 An Access Control Model

An AC model for process monitoring must allow for the restriction of access to authorized users only. In Sect. 5.1, we present the formal framework for AC rights definition and manipulation. In Sect. 5.2 and Sect. 5.3, we discuss AC model extensions 1) for coping with the problem of users playing multiple roles, and 2) for meeting the specific requirement of AC rights usability and maintainability.

5.1 Core AC Model

The specification of an AC module at the process monitoring level requires, first and foremost, the definition of access rights. A first step towards meeting *Requirement R1* (cf. Table 1) consists of defining access rights on *attributes* associated with specific process aspects that we call *objects*. Activities, process models or process instances are examples of accessed objects; attributes, indeed, reflect fine-grained characteritics of such objects. For this purpose, first of all, we formally define the link between an object and its associated attributes.

Definition 1 (Set of Attributes Associated with an Object). *Let ObjSet and AttSet respectively be the set of objects and the set of attributes involved in the process monitoring component. Then function attributeSet determines all attributes associated with an object obj* $\in$ *ObjSet. Formally:*
attributeSet: ObjSet $\mapsto$ *AttSet*^{P}
with $\forall att \in attributeSet(obj)$*: att is a valid attribute defined on obj.*

We associate with every object involved in the process monitoring component a set of attributes. Formally: $\forall obj \in ObjSet$: $attributeSet(obj) \subseteq AttSet$

In order to illustrate Def. 1, we reconsider the process from Fig. 3a. For the sake of simplicity, we will only retain the concrete concept of *activity* instead of the generalized concept of *object*. Let ObjSet = {`request expertise`, `generate expertise`, `request evaluation`, `provide evaluation`, `request comments`, `provide comments`} be a set of activities involved in the CR process. Let further AttSet = {Att_1, Att_2, Att_3, Att_4, Att_5} be the set of attributes involved in the CR process. Taking into account Def. 1, suppose that the set of attributes associated with each activity is captured as follows: attributeSet(`req. expertise`) = {Att_1, Att_3}; attributeSet(`gen. expertise`) = {Att_1, Att_2, Att_4, Att_5}; attributeSet(`req. evaluation`) = {Att_1, Att_3}; attributeSet(`prov. evaluation`) = {Att_1, Att_2, Att_5}; attributeSet(`req. comments`) = {Att_1, Att_3}; attributeSet(`prov. comments`) = {Att_1, Att_2}. We may think of Att_1 as the *activity status* that could take values from the set {`NotActivated`, `Activated`, `Running`, `Completed`, `Skipped`}. Att_2 may be the *starting date/time* of an activity. Att_3 could be the *employee black list* with possible values {`Yes`, `No`} specifying whether this list should be taken into account (or not) when employees are chosen to work on a specific task (e.g., `generate expertise`). If this list is taken into account, employees on black list may be excluded from those that may work on the task.

Based on Def. 1, we retain two types of information that may be checked/read: *the existence* and *the value* of an object's attribute. We distinguish between two different spectra of confidentiality defined on this information: 1) "Allow"/"don't allow" to check the existence of an attribute within an object; 2) "Allow"/"don't allow" to read the value of an attribute within an object, or allow to read another form of the value. From this we derive Def. 2.

Definition 2 (Access Control on Existence/Value of Attribute). *Let (obj, att) (obj ∈ ObjSet, att ∈ attributeSet(obj)) denote an attribute att associated with object obj. Then* $Exist_{obj,att}$ *determines whether it is allowed for someone (or not) to check the existence of attribute att within object obj;* $Val_{obj,att}$ *determines whether it is allowed for someone (or not) to read the value of attribute att within object obj. Formally:*

$$Exist_{obj,att} := \begin{cases} 0 & if\ not\ allowed\ to\ check\ existence\ of\ att\ within\ obj \\ 1 & if\ allowed\ to\ check\ existence\ of\ att\ within\ obj \end{cases}$$

$$Val_{obj,att} := \begin{cases} 0 & if\ not\ allowed\ to\ read\ value\ of\ att\ within\ obj \\ 1 & if\ allowed\ to\ read\ only\ another\ form\ of\ value \\ 2 & if\ allowed\ to\ read\ value\ of\ att\ within\ obj \end{cases}$$

Back to our example from Fig. 3a, suppose role "engineer" has the following access rights on the CR process:

- Access to activities `request expertise` and `generate expertise`.
- Access to the value of Att_1, access to another form of the value of Att_2.
- Access to the existence of Att_3 within `request expertise`.

Taking into account Def. 2, the AC on the existence/value of the different attributes can be captured as follows:

$Val_{\texttt{generate expertise},\ Att_1} = 2$, $Val_{\texttt{generate expertise},\ Att_2} = 1$,
$Val_{\texttt{request expertise},\ Att_1} = 2$, $Exist_{\texttt{request expertise},\ Att_3} = 1$

By default, we may suppose that the *closed policy*, considered as a classical approach for AC [11], applies. If not specified otherwise:

$Val_{obj,att} = 0$ and $Exist_{obj,att} = 0$, $\forall$ obj $\in$ ObjSet, att $\in$ attributeSet(obj)

In this context, two classical approaches for AC are discussed in literature [11]. The *closed policy* where positive rights need to be specified explicitly, and the *open policy* where negative rights need to be specified explicitly. The closed policy approach is known to ensure better protection than the open policy. In the latter, the need for protection is not strong: by default, access is to be granted.

Intuitively, we may also suppose that a specific operation prevails on another (cf. Fig. 4). For example, whenever it is allowed to read the value of an attribute, this implies that it is also allowed to read another form of the value, and also that it is allowed to check the existence of the attribute. Note that positive rights prevail on negative rights, i.e., positive rights are on bottom of the scale in Fig. 4. This is because of the closed policy adopted. Taking into account this prevailment scale, the following set of access rights is retained:
$Val_{\texttt{generate expertise},\ Att_1} = 2$, $Val_{\texttt{generate expertise},\ Att_2} = 1$,
$Val_{\texttt{request expertise},\ Att_1} = 2$, $Exist_{\texttt{request expertise},\ Att_3} = 1$,
$Exist_{\texttt{generate expertise},\ Att_4} = 0$, $Exist_{\texttt{generate expertise},\ Att_5} = 0$,
$Exist_{Activity,\ Attribute} = 0$, $\forall$ *Activity* $\in$ ObjSet $\setminus$ {`request expertise`, `generate expertise`}, *Attribute* $\in$ attributeSet(*Activity*)

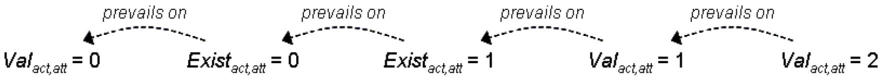

Fig. 4. Prevailment of Access Rights

AC rights being clearly defined, we present now a mechanism consisting of two functions that respectively return 1) whether an attribute is associated with an object or not, 2) the exact value or an abstraction of the value of an attribute.

Definition 3 (Existence/Value of Attribute). *Let (obj, att) (obj $\in$ ObjSet, att $\in$ attributeSet(obj)) be an attribute associated with an object. Let Val be a function on ObjSet $\times$ AttSet, Val: ObjSet $\times$ AttSet $\mapsto$ Dom_{AttSet} $\cup$ $\{Undefined\}$. Val reflects for each (obj, att) $\in$ ObjSet $\times$ AttSet its current value from domain Dom_{AttSet} <u>or</u> the value "Undefined" if att has not been written yet. Let FunctionSet be the set of functions that can be applied on the value of an attribute in order to provide another form of this value. For defining the specific function that can be applied on a specific attribute, we need the function:*

fa: ObjSet $\times$ AttSet $\mapsto$ FunctionSet $\cup$ $\{Undefined\}$ which maps each couple (obj, att) $\in$ ObjSet $\times$ AttSet to a specific function from FunctionSet <u>or</u> to "Undefined" if att $\notin$ attributeSet(obj) or no function is defined.

Then, f returns either the name of attribute att within object obj, <u>or</u> "Undefined"; h determines either the value <u>or</u> another form of the value of attribute att within object obj, <u>or</u> "Undefined". Formally:

f*: ObjSet* $\times$ *AttSet* $\mapsto$ *AttSet* $\cup$ $\{Undefined\}$

$$with\ f(obj, att) := \begin{cases} att & if \quad Exist_{obj,att} = 1 \wedge att \in attributeSet(obj) \\ Undefined & otherwise \end{cases}$$

h*: ObjSet* $\times$ *AttSet* $\mapsto$ $Dom_{AttSet} \ \cup \ Dom_{FunctionSet} \ \cup \ \{Undefined\}$

$$with\ h(obj, att) := \begin{cases} Undefined & if \quad Val_{obj,att} = 0 \\ fa(obj, att)(Val(obj, att)) & if \quad Val_{obj,att} = 1 \\ Val(obj, att) & if \quad Val_{obj,att} = 2 \end{cases}$$

$$Dom_{AttSet} = \bigcup\nolimits_{att \in AttSet} \quad Dom_{att}$$

$$Dom_{FunctionSet} = \bigcup\nolimits_{fct \in FunctionSet} \quad Dom_{fct}$$

If we go back to our example, applying Def. 3 would lead to the following existence/value of the different attributes:

$h(\texttt{generate expertise}, Att_1) = Val(\texttt{generate expertise}, Att_1)$
$f(\texttt{generate expertise}, Att_1) = Att_1$
$h(\texttt{generate expertise}, Att_2) = fa(\texttt{generate expertise}, Att_2)$
$(Val(\texttt{generate expertise}, Att_2))$
$f(\texttt{generate expertise}, Att_2) = Att_2$
$h(\texttt{request expertise}, Att_1) = Val(\texttt{request expertise}, Att_1)$
$f(\texttt{request expertise}, Att_1) = Att_1$
$h(\texttt{request expertise}, Att_3) = Undefined$
$f(\texttt{request expertise}, Att_3) = Att_3$
$h(\texttt{Activity}, Attribute) = f(\texttt{Activity}, Attribute) = Undefined$
for all other combinations of activities and attributes

The result of applying Def. 3 on our CR process, taking into account specific access rights assigned to role "engineer", is illustrated in Fig. 5.

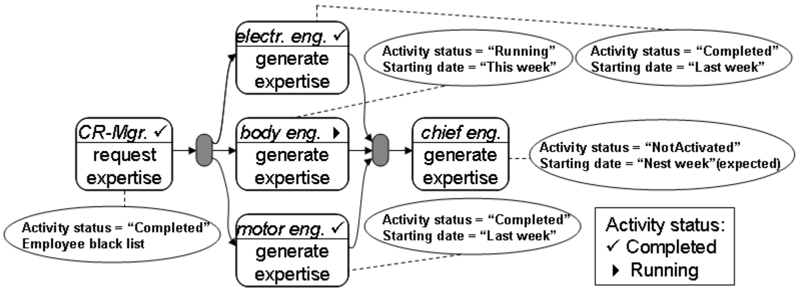
electr. eng. ✓
generate expertise
Activity status = "Running"
Starting date = "This week"
Activity status = "Completed"
Starting date = "Last week"
CR-Mgr. ✓
request expertise
body eng. ▶
generate expertise
chief eng.
generate expertise
Activity status = "NotActivated"
Starting date = "Nest week"(expected)
Activity status = "Completed"
Employee black list
motor eng. ✓
generate expertise
Activity status = "Completed"
Starting date = "Last week"
Activity status:
✓ Completed
▶ Running

Fig. 5. View on CR Process Provided to Role "Engineer"

5.2 Extended AC Model - Users Playing Multiple Roles

In this section, we recognize and we point out the fact that one user may play more than one role leading to inconsistencies between the AC rights associated with each of the different roles. As an example, a user may play both roles “manager” and “engineer”. On the one hand, engineers may not be given access to private information. On the other hand, managers may need to access private documents, and access to such information may be given to them. In this context, a number of conflict resolution policies are discussed in the literature [9, 10, 12, 13]. None of these policies represents “the perfect solution”. Whichever policy we take, we will always find one situation for which it does not fit. [9] states some problems of the different policies in conjunction with specific scenarios.

Interestingly, conflicts may result either from explicitly defining negative AC rights, or from applying the closed policy. In the latter case, a simple solution approach may be to neglect negative AC rights deriving from the used policy. Conflict resolution policies should be applied in the former case. For lack of space, we will abstain from discussing this matter here.

5.3 Extended AC Model - Compact Definition of AC rights

So far, we have expressed that a certain attribute is allowed to be accessed (or not) within a certain object, particularly a certain activity. However, we must also be able to state, for instance, within which processes this is allowed, i.e., what is the *context* of the AC to be defined. Candidates for the context are: the entire process visualization component (All), a group of process models, a particular process model, a group of process instances related to a particular process model, and a process instance.

The example elaborated in Sect. 5.1 presents a set of AC rights defined on a specific process model: CR_M. We may think of the following representation: (CR_M, $Val_{\texttt{generate expertise},\ Att_1} = 2$) stating that the value of Att_1 from activity `generate expertise` is allowed to be read within process model CR_M.

Suppose that AC rights are defined on a set of process models (e.g., M_1, M_2, M_3). This would lead to a set of couples: (M_1, $Val_{\texttt{generate expertise},\ Att_1} = 2$), ($M_2$, $Val_{\texttt{generate expertise},\ Att_1} = 2$), ($M_3$, $Val_{\texttt{generate expertise},\ Att_1} = 2$). Hence, we recognize the need for abstraction at the objects’ level in order to compact the definition of AC rights reducing the redundancy as much as possible. Therefore, one feasible way is to organize objects hierarchically (cf. Fig. 6): “All” at the top level, “Group of process models” at the next level down, “Process model” at the level just after, etc., and to propagate AC rights top-down. This allows us to meet the AC rights usability and maintainability requirement (cf. R3 in Table 1). Going back to our example, a group of process models $G_M = \{M_1, M_2, M_3\}$ would be defined, and the set of three couples would be reduced to the following couple: (G_M, $Val_{\texttt{generate expertise},\ Att_1} = 2$).

This approach would also simplify the definition of exceptions. As an example, it would be easy to express that no restrictions exist at all regarding accesses within any of the defined processes except the following: no accesses are allowed

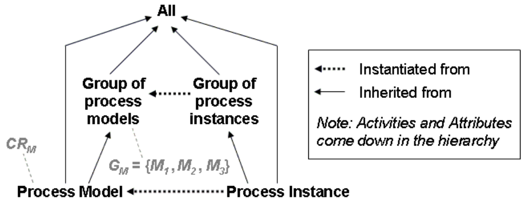
All
Group of process models
Group of process instances
CR_M
$G_M = \{M_1, M_2, M_3\}$
Process Model
Process Instance
Instantiated from
Inherited from
Note: Activities and Attributes come down in the hierarchy

Fig. 6. Objects' Hierarchy

to activity `approve CR` within the CR process model. This would be reduced to: $(All,\ Val_{\texttt{All},\ All} = 2)$ (i.e., access is given to everything in order to bypass the closed policy), and $(CR_M,\ Exist_{\texttt{approve CR},\ All} = 0)$ (i.e., access is retrieved from `approve CR` within CR_M).

6 Related Work

The provision of adequate security mechanisms is indispensable for any IS. Particularly, in the context of process-aware IS such as ADEPT [14], approaches have been proposed for dealing in a secure way with specific issues related to process management. As an example, Weber et al. propose an extension to RBAC in order to support process changes safely [15]. Rinderle and Reichert address changes that may occur within organizational structures [16]. They discuss how to support such changes, and how to adapt access rules when the underlying organizational model is changed. However, to our best knowledge, no research work has yet addressed the problem of AC in conjunction with process data integration and process monitoring.

Some of the aspects retained in this paper have already been introduced by others. The fine-grained control was discussed in [7] as one of the collaborative environment factors that determine the usability of a specific AC model. The authors argue that it is not sufficient to define AC rules only for groups of users on clusters of objects. A user might need a specific permission on an instance of an object at a particular point (i.e., time) in the collaboration session. In our approach, we were more explicit when defining AC rights at a fine-grained level: 1) we introduced the *spectrum of confidentiality* concept that would reflect the "specific" permission to grant or to revoke, and 2) we hierarchized objects such that AC rights may be defined in a *compact way* on the *different aspects* of the process model and instances. In [7], no details were given regarding *time* (i.e., a permission is valid only for a specific time space). This is an interesting point to be further investigated. In the context of adaptive process-aware IS, Weber et al. propose the definition of process type dependent AC rights [15]. Change commands that are useful within a particular *context* are only allowed. This idea can be compared to our approach of specifying the context of an AC right. However, both approaches focus on different aims. In [15], more assistance is provided for users when performing a change, whereas in this paper, the context notion is used for defining AC rights in a more focused way.

7 Summary and Outlook

In this paper, we identified an exhaustive list of AC requirements in the context of business process monitoring. We then presented possible solution approaches for major requirements, and we motivated the objects-based approach that we used for proposing a core AC model for process monitoring. Two extensions to this model were also discussed: the first one deals with the problems that may appear when a single user plays more than one role; the second extension introduces the "context" notion and discusses the compact definition of AC rights taking into account a defined objects' hierarchy. Major requirements were addressed using the proposed AC model and its extensions.

In future work, we will address requirements R4-7 (cf. Table 1). Our research work will also include the investigation of advanced issues such as the aggregation and the definition of AC rights on data elements and other process aspects.

References

1. Bobrik, R., Reichert, M., Bauer, T.: Requirements for the visualization of system-spanning business processes. In: Proc. DEXA'05 Workshops, Copenhagen (2005) 948–954
2. Dumas, M., v.d. Aalst, W., t. Hofstede, A.: Process-Aware ISs. Wiley (2005)
3. Rinderle, S., Bobrik, R., Reichert, M., Bauer, T.: Business process visualization - use cases, challenges, solutions. In: Proc. ICEIS'06, Paphos (2006) (accepted for publication)
4. v.d. Aalst, W., van Hee, K.: Workflow Management. MIT Press (2002)
5. Schaad, A., Moffett, J.: A framework for organisational control principles. In: Proc. ACSAC'02, Las Vegas (2002) 229–238
6. Botha, R., Eloff, J.: Separation of duties for access control enforcement in workflow environments. IBM Systems Journal **40** (2001) 666–682
7. Tolone, W., Ahn, G.J., Pai, T.: Access control in collaborative systems. ACM Computing Surveys **37** (2005) 29–41
8. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM ToISS **4** (2001) 224–274
9. di Vimercati, S.D.C., Samarati, P., Jajodia, S.: Policies, models, and languages for access control. In: Proc. Int'l Workshop DNIS'05, Aizu-Wakamatsu (2005) 225–237
10. Jajodia, S., Samarati, P., Sapino, M., Subrahmanian, V.: Flexible support for multiple access control policies. ACM ToDS **26** (2001) 214–260
11. Castano, S., Fugini, M., Martella, G., Samarati, P.: Database Security. Addison Wesley (1995)
12. Fernandez, E., Gudes, E., Song, H.: A model for evaluation and administration of security in object-oriented databases. IEEE ToKDE **6** (1994) 275–292
13. Shen, H., Dewan, P.: Access control for collaborative environments. In: Proc. CSCW'92. (1992) 51–58
14. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
15. Weber, B., Reichert, M., Wild, W., Rinderle, S.: Balancing flexibility and security in adaptive PMSs. In: Proc. CoopIS'05, Agia Napa (2005) 59–76
16. Rinderle, S., Reichert, M.: On the controlled evolution of access rules in information systems. In: Proc. CoopIS'05, Agia Napa (2005) 238–255