DOI: 10.2298/CSIS110907042S

Context-sensitive Access Control Model for Business Processes

Goran Sladić, Branko Milosavljević, and Zora Konjović

Faculty of Technical Sciences, University of Novi Sad, Trg D. Obradovića 6
21000 Novi Sad, Serbia
{sladicg, mbranko, ftn_zora}@uns.ac.rs

Abstract. This paper focuses on problems of access control for business processes. The subject of the paper is a specification of the Context-sensitive access control model for business processes (COBAC). In order to efficiently define and enforce access control for different business processes, the COBAC model is based on the RBAC (Role-based Access Control) model which is extended with the following entities: *context*, *business process*, *activity* and *resource category*. By using a context-sensitive access control it is possible to define more complex access control policies whose implementation by existing access control models for business processes is not possible or is very complicated. The COBAC's context model can describe rich context information and can be easily extended for specific cases. The introduction of business process and activity entities has facilitated the definition of access control policies for business processes. The categorization of resources enables the definition of access control policies for whole resource categories, and thus, potentially, reduces the number of policies which need to be defined. The COBAC model is applicable in different business information systems, and supports the definition of access control policies for both simple and complex business processes. The model is verified by a case study on a real business process.

Keywords: access control, RBAC, context-sensitive, workflow, business process.

1. Introduction

Davenport and Short [14] define a business process as a set of logically related tasks performed to achieve a defined business outcome. The Workflow Reference Model [26] extends the above definition by introducing the concept of a role, stating that a business process is a set of one or more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure which defines functional roles and relationships.

Workflow Management Coalition defines a workflow as a computerized facilitation or automation of a business process, in whole or in part [26]. In computer science literature, business process and workflow terms are often used interchangeably; in this paper we will use them as synonyms. Workflow management system (WfMS) is a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic [26]. Workflow management systems (WfMS) are widely used in enterprises to automate and facilitate their business processes. Because of the advantages of the workflow technology, many companies build their business on a WfMS platform. At the same time, the security is becoming more conspicuous. Secure workflows require that only authorized users should be able to perform specific tasks. The set of authorized users is usually defined by the security policy of the workflow.

In the Role-based Access Control (RBAC) model, access to resources of a system is based on the role of a user in the system [17]. The basic RBAC model comprises the following entities: *users*, *roles*, *permissions* and *sessions*, where permissions are composed of *operations* applied to *objects*. In RBAC, permissions are associated to roles, and users are made members of roles [17]. A user's interaction with the system is modeled by a session, where a user activates a subset of the roles to which she/he is assigned. This greatly simplifies management of access rights, so the RBAC model has generated great interest in the security community. It is customary to use the *role hierarchy* to aggregate permissions, i.e. a role is assumed to inherit the permissions assigned to its parent roles in the hierarchy. In addition, the role hierarchy also determines the roles that are available to a user, i.e. a user assigned to a particular role can also activate any subordinate roles in the hierarchy.

The RBAC model has been widely applied to both commercial and research workflow systems. In order to meet workflow access control requirements, the RBAC model is extended with some workflow specific entities. Usually, a specific role is assigned to each task in a workflow. Thus, users can perform tasks based on the privileges possessed by their own role or roles they inherit. However, there are still many problems in describing complex workflow access control authorization and constraints. One of the major problems is how to express business character and phase authorization constraints in carrying out the task between the roles and users. Traditional access control models, such as RBAC are passive access control. They do not take into account contextual information, such as processed data, location or time for making access decisions. Consequently, these models are inadequate for specifying the access control needs of many complex real-world workflows. As context data get involved, the access decision no longer depends on user credentials only, it also depends on the state of the system's environment and the system itself. For example, a doctor, during some task, can access to a patient record only if she/he treats the patient; a judge can change a judgment only if she/he is involved in the given trial proceedings and the trial proceedings is not finished. A possible solution for such and similar cases is to extend the RBAC model to met the required access control requirements. The extensions can be specific to cover just certain requirements, or they can be general to satisfy a wider range of access control requirements. In the first case, extensions include only some contextual information, while in the second case, extensions include wider range of context information in the access control decision.

The following features of a workflow access model should be included, in addition to standard ones, to successfully address different access control requirements:

- it should support access control at a task level,
- a user can have different privileges for different workflow instances,
- during a time/workflow execution, a role can have different permissions,
- during a time/workflow execution, different roles can be assigned or unassigned to a user, and
- assignment of roles to user and/or assignment of permissions to roles may depend on a context state (context information).

In this paper we propose the *Context-sensitive access control model for business processes* (COBAC) that includes listed features. This model is based on the RBAC model which is extended with the following entities: *business process*, *activity*, *context*, and *resource category*. Our model is based on RBAC because it simplifies role management and is the de facto access control model for commercial organizations. By using entities *business process* and *activities* in the model it is possible to efficiently define and enforce access control for different *business processes* (workflows). Since it is possible that access control can be affected by different factors from the environment and the system, the COBAC model explicitly introduces the notion of *context*. *Categorization* of the resources enables the definition of access control policies for the whole resource category, and thus, potentially, reduces the number of policies which need to be defined.

The COBAC model is organized into two components: the *Core* component defines core aspects of the model, and the *Constraint* component introduces different constraints including static and dynamic separation of duties. In this paper we present the *Core* component of the COBAC model.

The proof of concept for the proposed system is carried out through the implementation of a representative real-world case study – *Employment procedure for the Professor position at Faculty of Technical Sciences, University of Novi Sad*. While analyzing this business process we have identified some of the security requirements that cannot be implemented using the standard RBAC model since implementation of those requirements depends on the entities that are not part of RBAC. We have shown that it is possible to implement those requirements using the COBAC model.

The rest of the paper is structured as follows. Section 2 reviews the related work. Section 3 presents an overview of the *Core* component of the model. The access control enforcement is presented in Section 4. A case study is presented in Section 5. Section 6 concludes the paper and outlines further research directions.

2. Related Work

In this section two groups of research are presented. The topic of the first group is research on RBAC-based access control models adopted for use in workflow systems. The second group reviews latest results on context-sensitive access control.

Many models for workflow access control are based on the RBAC model extended with the entity representing the workflow task.

Russello et al. [38] present Workflow-Based Access Control (WBAC), an access control mechanism that adapts the access rights of subjects to the actual tasks that they have to fulfill. The requirements of entities' duties are expressed by means of workflows. WBAC ensures that entities can access the resources associated to a workflow task but only while such a task is active. The paper [54] introduces the TRBAC (Task-Role-Based Access Control) model which is based on RBAC and TBAC (Task-Based Access Control) [46] models. The TRBAC model is constructed by adding "Task" to the RBAC model. It's central idea is that the user has a relationship with a permission through a role and a task. Permissions are assigned to tasks and tasks are assigned to roles. This model supports the task classification. The task can be classified according to the organization structure and characteristics of access control in enterprise environment. They can be divided into two groups according to whether they belong to the workflow. Also, permissions can be divided into two groups: permissions on workflow tasks and permissions on workflow task instances. A similar model is presented by Oh and Park [35]. Their model is based on the classification of job functions. Three different types of tasks are identified: *workflow tasks* used for workflow oriented job functions, *non workflow tasks* used for non workflow oriented functions, and *supervision tasks* used for supervision job functions. Identically as the previous model, permissions are assigned to users through roles and tasks. Yao et al. [53] are also using tasks to associate roles and permissions. The unit of task defines the permission granularity. The authors propose constraints on user, role, task and session so that an access control configuration will not result in the leakage of a right to an unauthorized principal. A RBAC based workflow access control model in which tasks and permissions are assigned to the roles is presented in [8]. In the paper [45] the notion of an *activity* represents the basic extension of RBAC in order to provide efficient workflow access control. The activity is defined as a 5-tuple $(p\_set, r\_set, ant\_set, cur\_sta, a\_type)$ where p_set is the set of permissions which could be performed in the activity, r_set is the set of roles permitted to perform this activity, ant_set is the set of its direct ancestor activities that satisfy sequential relation in any workflow. cur_sta is the current state of activity, and a_type is the type of activity. To decouple the workflow access control model from the workflow model, the authors in [52] propose the Service-Oriented Workflow Access Control (SOWAC) model. In the SOWAC model, a service is the abstraction of a task and the unit for applying access control. Rather than associating roles to tasks directly, SOWAC binds them through a service as an interface.

Although the previously described access control models are extended for the use in workflow systems, they are not able to efficiently address all access control requirements that often occur in workflow systems, including those defined in Section 1. Therefore, the following researches, beside the notion of the *task* introduce some other concepts in their access control models to address some specific access control requirements.

A RBAC-based authorization workflow model which has a level request is proposed in [55]. The concept of the privilege in this model has the attributes of the time domain and the security level. It means that only in a certain period of time, a user with the security level greater than or equal to the security level of the privilege can implement some kind of operation. The proposed model could be used in the field that has the higher security requirements like military. Wainer et al. [50] propose a workflow access control model based on RBAC model extended with *case* and *organization unit* entities and appropriate relations. The entity *case* is added to be able to refer to an instance of a process. Within organizations, and thus in workflow applications, the concept of a hierarchy of people/organizations is prevalent. While workflow systems as a rule include some form of organizational modeling capabilities, RBAC by itself does not have such a hierarchy modeled. Therefore, the RBAC model is extended with the *organization unit* entity. The importance of using an organization structure in workflow access control models is also noticed in [51]. Shafiq et al. [40] propose an architecture for the adaptive real-time workflow-based collaborative system. The authors indicate that a key security requirement for a real-time workflow system is to provide the right data to the right person at the right time. They use the Generalized Temporal Role-Based Access Control (GTRBAC) model [31] to capture the real-time dependencies of such workflow applications. Importance of the time factor in workflow access control models is also noticed in [21] [27] [56]. Leitner et al. [32] propose an extended RBAC model for adaptive workflow systems (AW-RBAC). This model enables workflow systems to enforce access control under all different kinds of workflow changes. Specifically, the AW-RBAC model enables the definition of permissions in order to perform authorized control and data flow changes, administrative modifications and adaptations at the service level of a workflow system. In addition, access rights based on AW-RBAC can be specified in fine granularities based on the definition of constraints. Another research on access control for flexible workflows has been presented in [28]. It analyses how to realize the dynamic authorization relations of users, roles and permissions.

Another research direction is focused on covering wider range of access control requirements (usually by extending the RBAC model with the notion of *context*). Various definitions of the context have been proposed in literature [1] [15] [19] [39] [47]. Broadly, the notion of context relates to the characterization of environment conditions that are relevant for performing appropriate actions in the computing domain. Probably the most widely accepted definition has been given in [1] [15]:

"*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*".

In order to enable the fine grained access control many context-sensitive access control models have been proposed. The paper [25] describes a context-sensitive access control model which consists of a context model, a context-sensitive policy model and a context-sensitive request model. In the paper [3], the Conditional Role Based Access Control (C-RBAC) model is presented. This model relies on RBAC model and extends the notion of role by incorporating attributes, and bases on the notion of system context. Covington et al. [12] introduce the notion of environmental role, and provide a uniform access control framework that can be used to secure context-sensitive applications. Georgiadis et al. [22] discuss the integration of contextual information with team-based and role-based access control. Filho and Martin [18] showed the use of context information and its quality indicators to grant access permissions to resources. The article [34] presents a context-dependant authorization model for collaborative access control. The context model is based on the context parameter which represents a certain property of the environment, capable of influencing or controlling when and how a collaborative authorization policy is enforced. The context condition is a Boolean expression that compares the current value of a context parameter with either a predefined specific value, or the current value of another context parameter, while the context constraint is defined as a logical conjunction of context conditions. Koufi et al. [30] propose the role-based workflow authorization model that provides the flexibility for granting (revoking) fine-grained, context-dependent roles to (from) users dynamically at workflow run time to ensure a tight matching of roles to actual need. To encapsulate sets of permissions that take into account the domain-dependent context associated with each task, the concept of *contextual role* is defined. The contextual role concept is also used as a mechanism that associates users with contexts in a similar manner as the role concept is used as an intermediary between users and permissions. The authors use contextual rule-based approach that enables automatic role changes on the occurrences of specific events. The preliminary research results on context and privacy-aware access control model for network monitoring workflows have been reported in [36]. The presented model takes full advantage of the integration of contextual properties and thus allows covering the definition of both simple and complex business processes, as well as describing the rich contextual categorization of network resources. Many authors [6] [24] [29] [41] propose a context-based access control model for web services. Their approach grants and adapts permissions to users based on a set of contextual information collected from the environment of the system. The use of the context-sensitive access control for controlling XML documents is presented in the papers [7] [42] [43]. In the papers [11] [16] [33] [37] the context-awareness is identified as a crucial principle for the design of efficient access control models for the pervasive computing. The influence of temporal constraints to access control is probably the most thoroughly analyzed in [4] [31], while the influence of geospatial constraints is presented in [5] [13].

By analyzing the previously mentioned access control models for workflows, we noticed that most of them are based on the RBAC model which is extended with some other concepts specific for business processes. Although the literature recognizes a significant number of context-sensitive access control models, most workflow access control models only partially support context-sensitive access control, or do not support it at all. Most of models do not wholly support all features, given in Section 1, identified as important for workflow access control systems. COBAC, the model presented in this paper, has the following notable improvements over the aforementioned workflow access control models:

- It is based on the RBAC model which is extended with the concept of business process and activity. By using the business process and the activity concepts it is possible to bind certain access control segments with activities instead of binding them with the current session, and thus provide efficient definition of different policies and constraints in different business systems. This ensures the independence of these segments from the number of sessions in which activities are executed. Since a business process can be viewed as an executing sequence of activities in our model, the roles are assigned to activities that they can execute, while the permissions (to execute operations on resources) needed for activity execution are assigned to the activity. Thus, efficient and fine-grained control of least privileges is achieved.
- The policies can be defined at the level of each specific process instance (activity instance) and thus satisfy different protection requirements that different process instances may have. On the other hand, requiring the specification of authorizations for each single process instance would make the authorization specification task too complex, therefore the COBAC model supports policy definition at the process (activity) definition level. The policies defined at the process definition level will apply to all process instances.
- Since the real-world business processes may have sophisticated access control requirements that depend on different factors from the environment, we extend the assignment relations in the COBAC model with the context-dependant condition and thus provide support for the context-sensitive access control. Also, we extend the notion of the role with the context information. In this way, the context can affects all segments of access control model.
- The proposed context model is developed using ontologies in order to describe rich context information, to allow the semantic interoperability between different context-aware systems and to be easily extended for specific use cases.
- Resources in the COBAC model can be hierarchically organized thus lowering the number of policies that need to be defined.

3. COBAC Model

An overview of the basic concepts and their relations is presented in Figure 1. The solid lines are used to represent relations between the concepts, while the dashed lines are used to represent the influence of the context on the relations/concepts. The given model is based on the standard RBAC model which is extended with the following concepts: *business process*, *activity*, *context* and *resource category*. In the proposed model the definition of the session is taken from the standard RBAC model [17].

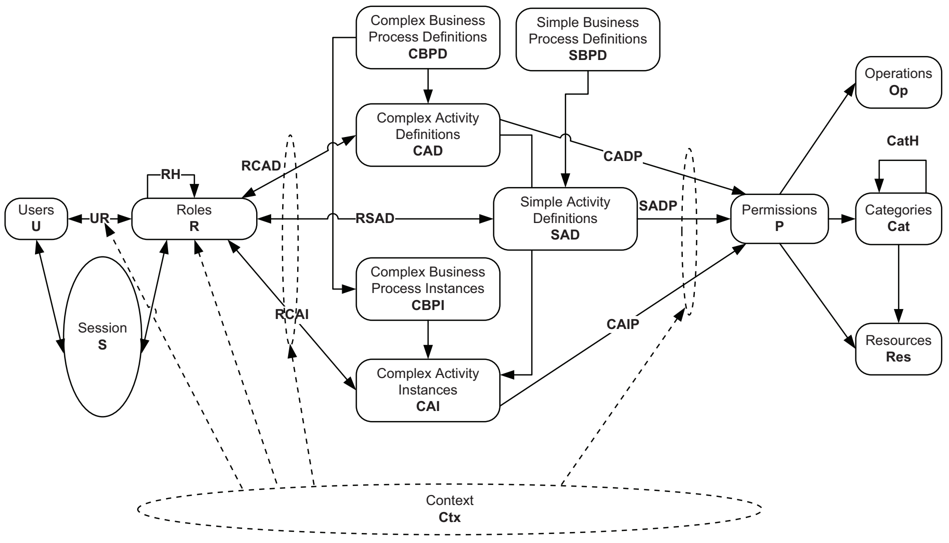
Complex Business Process Definitions CBPD
Simple Business Process Definitions SBPD
Operations Op
Complex Activity Definitions CAD
CatH
RCAD
CADP
RH
Users U
UR
Roles R
RSAD
Simple Activity Definitions SAD
SADP
Permissions P
Categories Cat
Session S
Complex Business Process Instances CBPI
RCAI
CAIP
Resources Res
Complex Activity Instances CAI
Context Ctx

Fig. 1. The COBAC model

The reason for introducing *business process* and *activity* is to achieve the efficient use of the RBAC model in business processes, because certain access control aspects need to be defined for a particular process (definition or instance) or it's activities (definition or instance). There are cases when a business process can be fully executed within a single user's session, but there are also cases where the process execution is distributed across multiple users' sessions. Therefore, it is necessary that an access control mechanism supports both of these cases.

We propose two types of business processes in the COBAC model: a *complex business process* and a *simple business process*. A complex business process is defined as proposed in [26] [14]. It consists of a set of activities among which there is a proper order relation. Since organizations usually carry out a lot of "relatively simple" tasks which do not require a workflow for their execution we introduce a simple business process to represent them (e.g. create a notification or create a report). A simple business process consists of only one activity. In the COBAC model activities are assigned to roles which are authorized to execute them. Although the use of two types of business processes may be inappropriate from the information security perspective since it can be viewed as unnecessary complexity, we decided to use this approach because of the model efficiency and the practical model implementation. Since simple business processes comprise a single activity, they do not require a workflow engine for execution and access control in this case can be optimized by omitting some steps and constraints verification which are necessary when the access control is performed for complex business processes.

Access control policies in the COBAC model can be defined for definitions and instances of business processes. If a policy is associated to a process definition, it will be applied to all instances of that process, while policies associated to an instance will be applied only to that particular instance.

By analyzing different real-world business processes, we concluded that in the case of complex business processes it is necessary to support the process definition - CBPD (i.e. activity definition - CAD) and the process instance - CBPI (i.e. activity instance - CAI) level of policies, while in the case of simple business processes it is sufficient to support only policies at the process definition - SBPD (i.e. activity definition - SAD) level.

During an activity execution it may be necessary to access some resources. In order to execute an activity, proper permissions, required for the resources being accessed, must be assigned to that activity.

Since access control may be influenced by some other factors from a system and an environment, the COBAC model also includes the notion of a *context*. By using context it is possible to define a context dependent constraint on assignment relations and on a role activation. Thus, an assignment relation with such constraint will be established only if the context dependent constraint is fulfilled. A similar case applies to a role activation.

Electronic resource (primarily documents) categorization is a crucial and well-proven instrument for organizing large volumes of resources/information. By grouping too many discrete items into understandable categories, users can quickly eliminate what is irrelevant or not interesting, and just pay attention to what matters most [9] [10] [20] [44]. Therefore the COBAC model explicitly introduces *resource category* to support efficient access control administration by defining permissions not only at the level of resources but also on the level of categories (i.e. permission can be defined for the *Sentence* category, or for the *Sentence for Minor* subcategory, or for the concrete judgment document). The COBAC model is independent of the categorization process. Resources may be classified according to their subjects or according to other metadata attributes (such as resource type, author, printing year etc.) using manual, automatic or hybrid categorization process [10] [44]. The concrete installation can be adjusted to the existing resource categorization in the information system.

The COBAC's UML model is presented in Figures 2 and 3. The role assignment relation is represented with the `UserRoleAssignment` class. If the context condition (modeled with the `ContextCondition` class) is associated with `UserRoleAssignment`, then a role will be assigned to a user only if the condition is satisfied. The definitions of complex business processes are represented by the `ComplexBPDef` class while complex process instances are represented by the `ComplexBPInst` class. The class `ComplexActivityDef` models complex activity definitions, while the class `ComplexActivityInst` models complex activity instances. `SympleBPDef` and `SimpleActivityDef` define simple business processes and simple activities. The assignments of activities to roles are modeled with the `RoleActivityAssignment` specializations. Those assignment relations will be established only if the associated context conditions are satisfied. The `Category` class represents categories of resources that are modeled with the `Resource` class. Operations executed on resources are defined using the `Operation` class. A permission to execute a certain operation can be defined at a resource and category level. The model defines permissions (`Permission`) for resources (`ResourcePermission`) and categories (`CategoryPermission`). A permission defined for a category applies to all category's resources and subcategories. The specializations of the `PermissionAssignment` class represent assignments of privileges to activities. The assignment of privileges to complex activities is defined using the `CAPermissionAssignment` specializations (`CADPermissionAssignment` and `CAIPermissionAssignment`), while the `SAPermissionAssignment` class models assignment of permissions to simple activities.

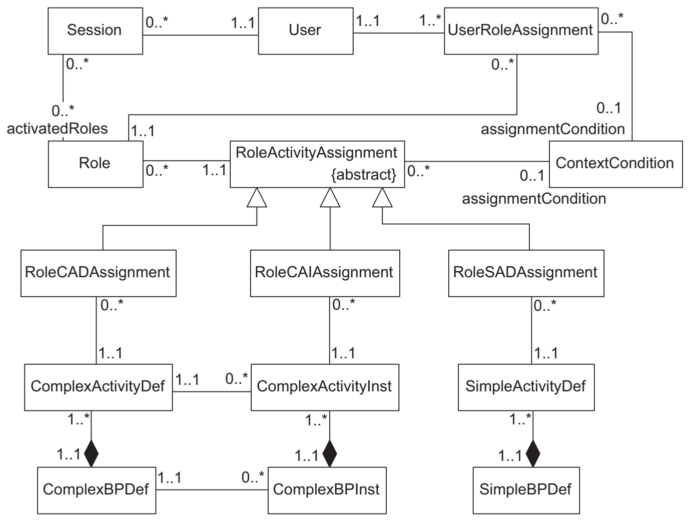

Fig. 2. User-role-activity assignment

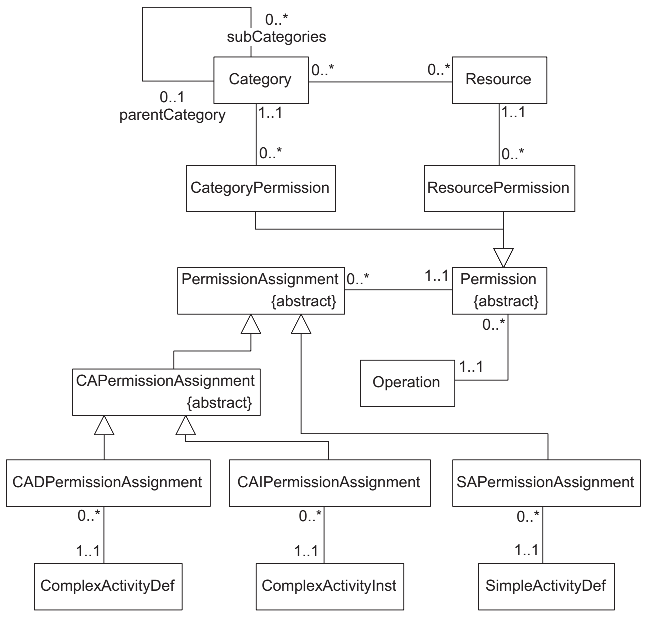

Fig. 3. Activity-permission assignment

In the rest of this section the complete description of the aforementioned models concepts is given. The basic terms used in the rest of the section are:

– U - set of users
– R - set of roles
– S - set of users sessions
– $CBPD$ - set of complex business process definitions
– CAD - set of complex activities definitions
– $CBPI$ - set of complex business processinstances
– CAI - set of complex activities instances
– $SBPD$ - set of simple business process definitions
– SAD - set of simple activities definitions
– P - set of permissions
– Op - set of operations
– Res - set of resources
– Cat - set of categories
– C - set of constraints
– Ctx - context
– CC - set of context conditions

3.1. The Model of Context

We chose the Web Ontology Language (OWL) [49] for the context modeling due to several reasons. First, it enables a formal representation of context and supports rich representation of different contextual information. Second, it allows the necessary semantic interoperability between different context-aware systems and also enables easy adjustment of the context model for use in different systems. Finally, it provides a high degree of inference making by providing additional vocabulary along with a formal semantics to define classes, properties, relations and axioms.

Basic entities of the context ontology are presented in Figure 4. This model defines *domain ontology* for a context in business systems. The *domain ontology* models only basic concepts and relations among them. When the COBAC model is used in the concrete case, the *domain ontology* model should be extended with the new concepts and relations which are specific for that case.

Two basic classes of the context model (see Figure 4) are *ContextFact* and *ContextExpression*. The class *ContextFact* represents a basic context fact, while the class *ContextExpression* represents the context expression. Different context facts can be classified as *ContextFact* specializations.

The *Actor* class is used for representing different actors of events. Different activities are modeled by the *Action* class, and resources are modeled by the *Resource* class. Location is described by the *Location* class. The *Time* class is used for modeling time factor in the model. Different purposes in this model are represented by *Purpose* class, while means are described by the *Means* class.

The context expression (class *ContextExpression*) represents a semantic binding of previously listed concepts, and it is based on 7 semantic dimensions (relations). Each aforementioned context fact creates one semantic dimension. Actually, the context expression describes events that took place and the conditions under which these events occurred. We extend five semantic dimensions defined in [2] [48] (*"who"*, *"what"*, *"where"*, *"when"* and *"how"*) with the notion *"why"* which defines the purpose, and with the notion *"related"* which defines the relation between different context expressions. We also define two specializations of the *"what"* concept. The specialization *"what action"* is used for defining the *"what"* relation with an action, while the specialization *"what resource"* is used for defining the *"what"* relation with a resource.

The context model relations are presented in Listing 1, while the context expression definition is presented in Listing 2. In our definition, the context expression must contain at least one *"who"* and *"what"* relation. We add this restriction because it is necessary that the context expression contains the information who/what did something, and what he/it did, in order to describe an event.

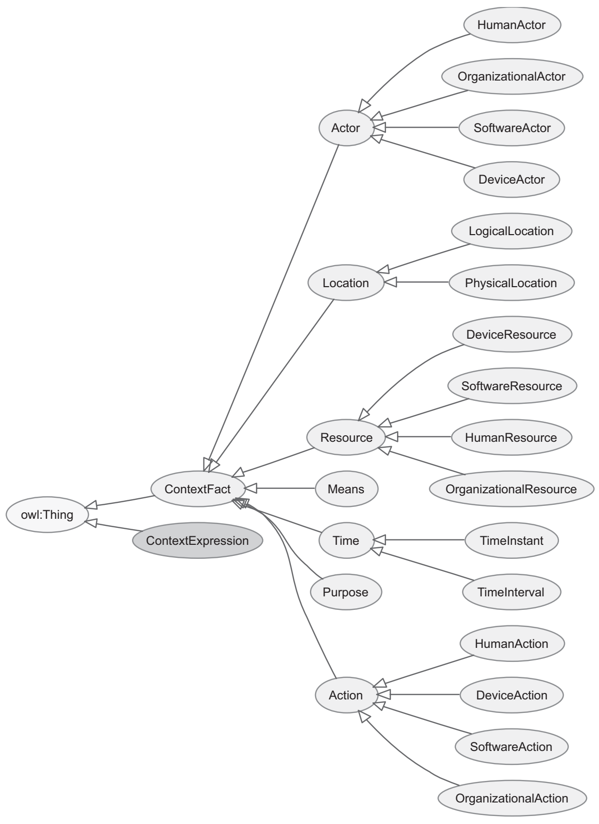

Fig. 4. The context model

```
ctx:hasWhoPart          a owl:ObjectProperty.
ctx:hasWhatPart         a owl:ObjectProperty.
ctx:hasWhatActionPart   a owl:ObjectProperty; rdfs:subPropertyOf ctx:hasWhatPart.
ctx:hasWhatResourcePart a owl:ObjectProperty; rdfs:subPropertyOf ctx:hasWhatPart.
ctx:hasWhenPart         a owl:ObjectProperty.
ctx:hasWherePart        a owl:ObjectProperty.
ctx:hasWhyPart          a owl:ObjectProperty.
ctx:hasHowPart          a owl:ObjectProperty.
ctx:isRelatedTo         a owl:ObjectProperty,
                          owl:SymmetricProperty;
```

Listing 1. The context model relations

```
ctx:ContextExpression a owl:Class;
owl:equivalentClass[a owl:Class; owl:intersectionOf(
[a owl:Restriction; owl:allValuesFrom ctx:Actor; owl:onProperty ctx:hasWhoPart]
[a owl:Restriction; owl:someValuesFrom ctx:Actor; owl:onProperty ctx:hasWhoPart]
[a owl:Restriction; owl:allValuesFrom ctx:Action;
                    owl:onProperty ctx:hasWhatActionPart]
[a owl:Restriction; owl:allValuesFrom ctx:Resource;
```

```
                    owl:onProperty ctx:hasWhatResourcePart]
[a owl:Restriction; owl:onProperty ctx:hasWhatPart; owl:someValuesFrom
                     [a owl:Class; owl:unionOf (ctx:Action ctx:Resource)]]
[a owl:Restriction; owl:someValuesFrom ctx:Event; owl:onProperty ctx:hasWhatPart]
[a owl:Restriction; owl:allValuesFrom ctx:Event; owl:onProperty ctx:hasWhatPart]
[a owl:Restriction; owl:allValuesFrom ctx:Means; owl:onProperty ctx:hasHowPart]
[a owl:Restriction; owl:allValuesFrom ctx:Time; owl:onProperty ctx:hasWhenPart]
[a owl:Restriction; owl:allValuesFrom ctx:Location; owl:onProperty ctx:hasWherePart]
[a owl:Restriction; owl:allValuesFrom ctx:Purpose; owl:onProperty ctx:hasWhyPart]
[a owl:Restriction; owl:allValuesFrom ctx:ContextExpression;
                    owl:onProperty ctx:isRelatedTo] )].
```

Listing 2. The context expression definition

Context Condition We define the context condition as a logical expression which may consist of queries for searching context ontology (like SPARQL), context functions, logical operators $\{\neg, \wedge, \vee\}$ and comparison operators $\{<, \leq, >, \geq, =, \neq\}$. The context functions are used to retrieve some current information from a system, like *who is current user*, or *what activity is being currently executed*. The context expression defined in the EBNF notation is presented in Listing 3.

```
ContextCondition ::=  Query
       | Function
       | ContextCondition BinaryOperator ContextCondition
       | UnaryLogicalOperator ContextCondition
       | "("ContextCondition")"
       | "TRUE"
       | "FALSE"
       | String
       | Number
BinaryOperator ::= ComparsionOperator|LogicalBinaryOperator
ComparsionOperator ::= "<"|"<="|">"|">="|"=="|"!="
LogicalBinaryOperator ::= "&&"|"||"
UnaryLogicalOperator ::= "!"
Query ::= "QUERY {" (String|Function){String|Function} "}"
Function ::= "$$"Name"("[Param{","Param}]")$$"
Param ::= String|Number|Function|Query
```

Listing 3. EBNF notation for the context condition

The function $evalCond : CC \rightarrow \{\top, \bot\}$ verifies if the given context condition is satisfied.

From the set CC there are two specific conditions: σ ($\sigma = TRUE$) and $\bar{\sigma}$ ($\bar{\sigma} = FALSE$). The condition σ is the condition which is always satisfied, i.e. $evalCond(\sigma) = \top$. The condition $\bar{\sigma}$ is the condition opposite to σ, i.e. it is never satisfied, $evalCond(\bar{\sigma}) = \bot$.

The context conditions are used for the assignment relations (a proper assignment relation will be established if the assigned context condition is satisfied) and for roles definition in order to enable/disable roles. A detailed explanation of the use of the context condition is given in the following sections.

Variability of Context Condition From the viewpoint of the context condition variability we identify three types of the context condition:

– *slowly varying* (*session safe*) context conditions are those conditions whose result is unchangeable during the user session.
– *frequently varying* (*activity safe*) context conditions are those conditions whose result can be changed during the user session, but it is unchangeable during the execution of the activity.
– *constantly varying* context conditions are those conditions whose result can be changed during the user session or during the execution of the activity.

3.2. The model of Business Process

The i-th complex business process definition ($cbpd_i$) is defined as the tuple: $cbpd_i = (CAD_i, F_{BPD_i})$, where:

– CAD_i - The set of complex activity definitions of $cbpd_i$, i.e. $CAD_i = \{cad_{i1}, cad_{i2}, \cdots, cad_{in}\}$, where cad_{ij} is the definition of the j-th complex activity of the i-th complex business process, $1 \leq j \leq n$, and n is a number of activity definitions of $cbpd_i$.
– F_{BPD_i} - The flow control relation which defines the execution order of activities from the set CAD_i.

The given model specifies three types of complex activities:

– *start* - The first (starting) activity of a complex business process. Invoking this activity will initiate a new instance of a complex business process. Each complex business process definition must have exactly one activity of the start type.
– *end* - The last (ending) activity of a complex business process. Invoking this activity will end the current instance of a complex business process.
– *regular* - The regular activity which represents a part of the business logic implemented by the business process.

The function $typeOf : CAD_i \rightarrow \{start, regular, end\}$ determines the type of the complex activity.

The function $instanceOf_{BP} : CBPI \rightarrow CBPD$ determines the definition of the complex business process instance.

A similar function, $instanceOf_A : CAI \rightarrow CAD$, is used in the case of determining the definition of the complex activity instance.

The function $activityDefOf : CAD \rightarrow CBPD$ determines the belonging of CADs to CBPDs:

$$activityDefOf(cad) = cbpd_i \mid cbpd_i = (CAD_i, F_{BPD_i}) \wedge cad \in CAD_i$$

The k-th instance of the i-th complex business process ($cbpi_i^k$) is defined as a tuple: $cbpi_i^k = (CAI_i^k, F_{BPI_i^k})$, where:

– $instanceOf_{BP}(cbpi_i^k) = cbpd_i$.
– CAI_i^k - The set of complex activity instances of $cbpi_i^k$., i.e. $CAI_i^k = \{cai_{i1}^k, cai_{i2}^k, \cdot, cai_{in}^k\}$, where $instanceOf_A(cai_{ij}^k) = cad_{ij}$

– $F_{BPI_i^k}$ - The flow control relation which defines execution order of activities from the set CAI_i^k.

Similarly to the $activityDefOf$ function, the function $activityInstOf : CAI \rightarrow CBPI$ determines the belonging of CAIs to CBPIs:

$$activityInstOf(cai) = cbpi_i \mid cbpi_i = (CAI_i, F_{BPI_i}) \wedge cai \in CAI_i$$

As noted before, the i-th simple business process (spd_i) consists of a simple activity (sad_i), e.g. $spd_i = (sad_i)$.

3.3. The Model of User

Let U be the set of all users in the system, and S be the set of users' sessions. The predicate $usersSession(u, s)$ verifies if the session s ($s \in S$) belongs to the user u ($u \in U$).

Various previous researches have identified that access control may depend on certain user's properties. In the case of RBAC-based models, assignment of roles to a user can depend on user's properties. The COBAC model supports this requirement by using context constraints in user-role assignment relation, where context constraints include some user's properties.

3.4. The Model of Role

Similar to [4] [31], roles can be in one of the three states:

– *disabled* - A user can not activate such roles in her/his current session. *Disabled* roles can change their state to *enabled*.
– *enabled* - A user can activate such roles in her/his current session. An *enabled* role can switch to the *disabled* or *active* state.
– *active* - This state applies to each user individually. If a user activates a role in her/his session, then that role changes state from *enabled* to *active*, but only for that user. By deactivating a role, it changes the state from *active* to *enabled* only if the role was active in **exactly one** session of the user, otherwise the role will continue to be in the *active* state. From the *active* state, a role can also change the state to *disabled*.

Whether a role will be in the *enabled* or *disabled* state depends on the current context state. Therefore, we define the role as a tuple : (rn, sc, sct) where:

– rn - The role name,
– sc - The context condition for enabling/disabling the role (*state condition*), and
– sct - The context condition type (*state condition type*). Possible values are from the set $\{dc, ec\}$. It defines what the condition applies to: the condition for disabling a role (dc) or the condition for enabling a role (ec).

Each rn can exist in only one tuple:

$$\forall t_1 = (rn_1, sc_1, sct_1), \forall t_2 = (rn_2, sc_2, sct_2) :$$
$$rn_1 = rn_2 \Rightarrow sc_1 = sc_2 \wedge sct_1 = sct_2 \wedge t_1 = t_2$$

If the condition sc is satisfied the role will be in the *disabled* state if $sct = dc$, or in the *enabled* state if $sct = ec$. If the condition is not satisfied the role will be in the opposite state then defined by sct.

In the specific case, the sc condition is defined as the context condition which is always satisfied: (rn, σ, ec). This actually means that there is no context influence to disabling/enabling a role. A role is always *enabled*.

Although, it is possible to use the negation of the condition to accomplish role enabling/disabling effect, we used the context condition type to increase clarity of the specific role definition by explicitly stating if condition is used for enabling or disabling of the role.

To determine the role state we introduce the following predicates:

– $disabled(r)$ - Verify if the role is disabled,
– $enabled(r)$ - Verify if the role is enabled, and
– $active(r, u)$ - Verify if the role r is active for the user u.

The predicate $disabled(r)$ is true if the role r is disabled at the current context state:

$$disabled(r) \Leftarrow (r.sct = dc \wedge evalCond(r.sc)) \vee (r.sct = ec \wedge \neg evalCond(r.sc))$$

The predicate $enabled(r)$ is true if the role r is enabled at the current context state:

$$enabled(r) \Leftarrow (r.sct = ec \wedge evalCond(r.sc)) \vee (r.sct = dc \wedge \neg evalCond(r.sc))$$

The predicate $active(r, u)$ is true if there is at least one session of u in which the role r is active:

$$active(r, u) \Leftarrow usersSession(u, s) \wedge activeInSession(s, r)$$

where the predicate $activeInSession(s, r)$ verifies whether the role r is active in the session s.

3.5. User-Role Assignment

The proposed model supports two ways of assigning roles to users: *static* and *dynamic*.

In the case of the *static* type, a user is assigned roles for which there are assignment relations between them and the user. The $sRoleAssign(r, u, cc)$ relation defines *static* assignment of the role r to the user u if the context condition cc is satisfied.

Some research identify cases when a predefined role-user relation is not satisfactory, and it is necessary to assign certain roles only to the users who satisfy certain conditions. This model of role assignment in the COBAC model is supported by the *dynamic* type of assignment. In this case a role can be assigned to any user if the assignment condition of the role is met.

The relation $roleCondAssign(r, rac)$ defines the *role assignment condition* ($rac \in CC$) which should be fulfilled in order to assign the role r to a user. This condition (rac) may contain any context data including some user's properties. This way a user-role assignment depends on the user's properties is supported. The dynamic role assignment is defined with the following predicate:

$$dRoleAssign(r) \Leftarrow roleCondAssign(r, rac) \wedge evalCond(rac)$$

If the predicate $dRoleAssign(r)$ is satisfied, the role r will be assigned to the current user whose roles are loaded.

The predicate $isRoleAssigned(u, r)$ verifies if the role r is assigned to the user r, regardless of the assignment type:

$$\begin{aligned} isRoleAssigned(r, u) \Leftarrow & dRoleAssign(r) \vee \\ & (sRoleAssign(r, u, cc) \wedge evalCond(cc)) \end{aligned}$$

If $cc = \sigma$, it is a *context free static user-role assignment* and otherwise it is a *context influenced static user-role assignment* ($cc \neq \sigma$).

3.6. Role Hierarchy

The role hierarchy (RH) is defined as a partial order over the set of roles R ($RH \subseteq R \times R$), i.e. as an inheritance relation, marked as $\succeq$, where if the role r_1 inherits the role r_2 then stands $r_1 \succeq r_2$.

The predicate $canObtainRole(u, r)$ verifies if the role r can be assigned to the user u in the presence of the role hierarchy:

$$\begin{aligned} canObtainRole(u, r_j) \Leftarrow & isRoleAssigned(u, r_j) \vee \\ & ((r_i \succeq r_j) \wedge canObtainRole(u, r_i)) \end{aligned}$$

The function $usersRoles : U \rightarrow 2^R$ determines all roles assigned to a user in the presence of the role hierarchy:

$$usersRoles(u) = \{r \mid canObtainRole(u, r)\}$$

3.7. Role-Activity Assignment

It is possible to assign a privilege to each role from the set of roles R to execute certain business process activities. Since the COBAC model supports privileges for definitions and instances of complex activities and for definitions of simple activities, we introduce the following three role-activity assignment relations:

– The expression $rCADAssign(r, cad, cc)$ defines a relation which allows the role $r \in R$ to execute the complex activity definition $cad \in CAD$ if the context condition $cc \in CC$ is fulfilled.
– The expression $rCAIAssign(r, cai, cc)$ defines a relation which allows the role $r \in R$ to execute the complex activity instance $cai \in CAI$ if the context condition $cc \in CC$ is fulfilled. Also, the type of the cai activity can not be *start*, because it is meaningless to assign a privilege for an instance of the *start* activity ($cai \in CAI \wedge instanceOf_A(cai) = cad \wedge typeOf(cad) \neq start$). The *start* type of activity is used only to initiate a new instance of the complex business process.
– The expression $rSADAssign(r, sad, cc)$ defines a relation which allows the role $r \in R$ to execute the simple activity $sad \in SAD$ if the context condition $cc \in CC$ is fulfilled.

Depending if $cc = \sigma$ there are *context free static role-activity assignment* and *context influenced static role-activity assignment* ($cc \neq \sigma$).

Verification if the role r is allowed to execute the complex activity instance cai ($cai \in CAI$) is defined with the following predicate:

$$canExecCAI(r, cai) \Leftarrow (rCAIAssign(r, cai, cc) \wedge evalCond(cc)) \vee$$
$$(rCADAssign(r, cad, cc) \wedge evalCond(cc) \wedge instanceOf_A(cai) = cad)$$

In a similar manner, the predicates for verification whether the role r is allowed to execute a complex activity definition or a simple activity definition are defined as follows:

$$canExecCAD(r, cad) \Leftarrow rCADAssign(r, cad, cc) \wedge evalCond(cc)$$
$$canExecSAD(r, sad) \Leftarrow rSADAssign(r, sad, cc) \wedge evalCond(cc)$$

A certain role can create a new complex business process instance if the role has permission to execute the *start* type activity of the process. The function $assignedCompProcToStartR \rightarrow 2^{CAD}$ determines all *start* type activities assigned to the given role:

$$assignedCompProcToStart(r) =$$
$$\{cad \mid canExecCAD(r, cad) \wedge typeOf(cad) = start\}$$

The function $assignedCompActToExec : R \rightarrow 2^{CAI}$ determines all complex activity instances which the role can execute. The role can execute the complex activity instance if the role has a permission to execute it (the predicate $canExecCAI(r, cai)$) and if the complex activity instance is next in the order of execution. The order of execution is verified with the $nextToExec$ predicate. This actually provides an order (sequence) of events, i.e. activities are executed in the order defined by a business process to which they belong to:

$$assignedCompActToExec(r) =$$
$$\{cai \mid nextToExec(cai) \wedge canExecCAI(r, cai)\}$$

The set of simple activities that the role can execute is the return value of the function $assignedSimpActToExecR \rightarrow 2^{SAD}$:

$$assignedSimpActToExec(r) = \{sad \mid canExecSAD(r, sad)\}$$

3.8. The Model of Permission

In order to improve the COBAC administration, resources can be organized into categories, where each resource can be classified into more categories. The relation $assignCat(res, cat)$ classifies the resource $res \in Res$ into the $cat \in Cat$ category.

Like roles, resource categories can also be hierarchically organized. The resource hierarchy is defined as a partial order over the categories set Cat ($CatH \subseteq Cat \times Cat$), i.e. as inheritance relation, marked as $\succeq_{cat}$, where if the category cat_i inherits the category cat_j then stands $cat_i \succeq_{cat} cat_j$. All resources that belong to the category cat_i also belong to the category cat_j. The predicate $belongsToCat$ verifies whether the resource res belongs to the category cat:

$$\begin{aligned} belongsToCat(res, cat_j) \Leftarrow & assignCat(res, cat_j) \vee \\ & (cat_i \succeq_{cat} cat_j \wedge belongsToCat(res, cat_i) \end{aligned}$$

A permission to execute a certain operation can be defined for resources and for categories. A permission defined for a category applies to all the category's resources and subcategories. The resource permission is defined as: $rp = (res, op), res \in Res, op \in Op$, while the category permission is defined as: $cp = (cat, op), cat \in Cat, op \in Op$. Let RP be the set of all permissions defined for resources and CP be the set of all permissions defined for categories than: $P = RP \cup CP$.

In order to successfully execute a business process activity it is necessary to assign it the required permissions. In the COBAC model, the activity-permission assignment is specified with the following relations:

- $cadPermissionAssign(cad, p, cc)$ - The permission $p \in P$ is assigned to the complex activity definition $cad \in CAD$ if the context condition $cc \in CC$ is fulfilled,
- $caiPermissionAssign(cai, p, cc)$ - The permission $p \in P$ is assigned to the complex activity instance $cai \in CAI$ if the context condition $cc \in CC$ is fulfilled, and
- $sadPermissionAssign(sad, p, cc)$ - The permission $p \in P$ is assigned to the simple activity definition $sad \in SAD$ if the context condition $cc \in CC$ is fulfilled.

Similary to the role-activity assignment, if $cc = \sigma$ it is a *context free activity-permission assignment* and if $cc \neq \sigma$ it is a *context influenced activity-permission assignment*.

The functions which determine the privileges (defined for the corresponding operation) associated to a complex activity definition, complex activity instance or simple activity definition are defined as follows:

$$cadPermission : CAD \times Op \rightarrow 2^P$$
$$cadPermission(cad, op) = \{p \mid p.op = op \land cadPermissionAssign(cad, p, cc)\}$$

$$caiPermission : CAI \times Op \rightarrow 2^P$$
$$caiPermission(cai, op) = \{p \mid p.op = op \land caiPermissionAssign(cai, p, cc)\}$$

$$sadPermission : SAD \times Op \rightarrow 2^P$$
$$sadPermission(sad, op) = \{p \mid p.op = op \land sadPermissionAssign(sad, p, cc)\}$$

The permission p_1 is *contained* in the permission p_2 ($p_1 \sqsubseteq p_2$) if:

- p_1 and p_2 are defined for the same operation and resource ($p_1 = p_2$), or
- p_1 and p_2 are defined for the same operation and the p_1 category is a sub-category of the p_2 category, or
- p_1 and p_2 are defined for the same operation and the p_1 resource belongs to p_2 category.

This can formally be noted as follows:

$$\begin{aligned} p_1 \sqsubseteq p_2 \Rightarrow & (p_1 \in RP \land p_2 \in RP \land p_1.op = p_2.op \land p_1.res = p_2.res) \lor \\ & (p_1 \in CP \land P_2 \in CP \land p_1.op = p_2.op \land p_1.cat \succeq_{cat} p_2.cat) \lor \\ & (p_1 \in RP \land p_2 \in CP \land p_1.op = p_2.op \land belongsToCat(p_1.res, p_2.cat)) \end{aligned}$$

If an activity possesses the permission p_2, and if that activity needs the permission p_1 to execute a certain operation on a resource, than the activity will be able to execute the operation only if $p_1 \sqsubseteq p_2$.

4. Access Control Enforcement

In this section, access control enforcement defined by the COBAC model without constraints enforcement is presented. The process of the access control enforcement is performed through four phases:

- role activation,
- creation of activity (task) list that a user can execute,
- verification if the user is allowed to perform the activity in the moment when she/he initiate execution of the given activity, and
- verification if it is allowed to access the required resources during the execution of the activity.

In order to define access control enforcement in the consistent way we assume that the context conditions specified in the user-role assignment relations and the context conditions used for enabling/disabling roles are *slowly varying*, while the conditions in the role-activity relations and in the activity-permission relations are *frequently varying*.

Based on the previous assumptions, roles assigned to a user in here/his session are **not changed** during the session duration, but it is possible that privileges, to execute certain activities, assigned to those role are **changed**.

4.1. Roles Activation

The first step in the roles activation is loading of the roles assigned to the user, and then, from the set of the loaded roles, the *disabled* roles are removed. The algorithm for the activation of roles in this phase can be changed in order to meet the requirements of different systems. Algorithm 1 uses simple way of role activation where all roles assigned to the user are activated.

Algorithm 1 An example of role activation

```
NAME: ActivateRoles
INPUT: $u \in U$ - user
$s \in S$ - session of $u$
OUTPUT: $ARS$ - activated roles
$URS$ := usersRoles($u$)
for each $r \in URS$ do
  if disabled($r$) then
    removeFromSet($r$, $URS$)
$ARS$ := activateAllRolesForSession($URS$, $s$)
```

4.2. Activity List Creation

The creation of the activity list for the user is the process of selection of activities which are allowed to be executed by the user. The created activity list is represented as the tuple $(SCPS, ECAS, ESAS)$ where:

– $SCPS \subseteq CAD \wedge \forall cad \in SCPS, typeOf(cad) = start$ (*Start Complex Process Set*) represents the set of the complex activity definitions of the “start” type which user can execute, i.e. the user can create a new instance of the complex business process.
– $ECAS \subseteq CAI$ (*Execute Complex Activity Set*) represent the complex activity instances set which user can execute.
– $ESAS \subseteq SAD$ (*Execute Simple Activity Set*) represent the simple activities set which user can execute.

Algorithm 2 describes the process of the activity list creation for the given user. The creation of the $SCPS$ set includes loading of all complex activity definitions of the “start” type which are assigned to user’s active roles. The function $assignedCompProcToStart(r)$ determines all the *start* type activities which are assigned to the given role. The set $ECAS$ is formed from all activity instances which are assigned to the user’s active roles and which are next in

Algorithm 2 Activity list creation

```
NAME: CreateActivityList
INPUT: $u \in U$ - user
$ARS \subseteq R$ - active roles of user in session
OUTPUT: $(SCPS, ECAS, ESAS)$ - user's actitivty list

{Create Start Complex Process Set (SCPS)}
$SCPS$ := $\bigcup_{r \in ARS}$ assignedCompProcToStart($r$)

{Create Execute Complex Activity Set (ECAS)}
$ECAS$ := $\bigcup_{r \in ARS}$ assignedCompActToExec($r$)

{Create Execute Simple Activity Set (ESAS)}
$EAS$ := $\bigcup_{r \in ARS}$ assignedSimpActToExec($r$)
```

order to be executed. The function $assignedCompActToExec(r)$ determines all the complex activity instances which the role r can execute. The set $SCPS$ is formed from simple activities which are assigned to the users active roles. The set of simple activities that the role can execute is return value of the function $assignedSimpActToExec$.

4.3. Verification of Privilege to Execute Action

When a user from the activity list selects an activity to execute, it is necessary to verify once more whether the user is allowed to execute that activity. The reason for this verification is the existence of the *frequently varying* context conditions which may be changed between the creation of activity list and the execution of the activity.

Verification whether it is allowed to execute a complex activity instance is presented in Algorithm 3. The user can execute the activity instance if she/he has an active role with a privilege to execute that activity (the predicate $canExecCAI$). The similar algorithm is used for verification whether it is allowed to execute simple activities or to create a new process instance.

Algorithm 3 Verification if it is allowed to execute an activity instance

```
NAME: ExecOfComplexActivityAllowed
INPUT: $cai \in CAI$ - complex activity instance to execute
$u \in U$ - user
$ARS \subseteq R$ - active roles of user in session
OUTPUT: $result$ - result, can user execute taks

$result$ := false
for each $r \in ARS$ do
  if canExecCAI($r$, $cai$) then
    return $result$ := true
return $result$
```

4.4. Verification of Permission to Access to Resource

An activity can access to resources only if it has necessary permissions. Verification if the complex activity instance cai can execute the operation op on the resource res is presented in Algorithm 4. The activity instance cai can execute the operation op on the resource res if the permission that allows execution is **contained** in the permission assigned to the instance ($caiPermission$) or its definition ($cadPermission$). The similar algorithm is used for simple activities.

Algorithm 4 Permission verification

```
NAME: CanExecOperation
INPUT: $cai \in CAI$ - complex activity instance
$op \in Oper$ - operation
$res \in Res$ - resource
OUTPUT: $result$ - result, can operation be executed on resource

$result$ := false
$p$ := ($res$, $op$)
$RPS$ := caiPermission($cai$, p.op) ∪ cadPermission($instanceOf_A(cai)$, p.op)
for each $rp \in RPS$ do
  if $p \sqsubseteq rp \wedge evalCond(rp.cc)$ then
    return $result$ := true
return $result$
```

5. A Case Study

The verification of the COBAC model is carried out through the analysis and implementation of the security segment of the workflow-based information system. The model is tested on the real-world business process - *Employment procedure in the Professor position at Faculty of Technical Sciences, University of Novi Sad*. This process is implemented using a document-oriented workflow system.

The process includes cases where user's privileges depend on a current task/workflow instance being executed, and cases where access control decision depends on the elements outside standard access control entities, precisely it depends on the content of a document being involved in the executed workflow instance. We decided to choose this process as a case study because it has security requirements that can not be fully implemented with existing workflow access control models, but it is possible to implement them using COBAC. Therefore, security requirements of this process are suitable to demonstrate full capacity of the COBAC model.

The simplified version of the given process is presented in Figure 5. The whole process initiates a certain *Department* which requests a new employment in the Professor position. Then *Dean* approves or disapproves the *Department's* request. If the request is approved, *Dean* forms the commission whose task is to select the best candidate for the job position. After this, *Human Resources* publishes the job vacancy in the papers, and accepts candidates' applications.

Commission analyzes accepted applications, ranks all candidates, and suggests the most suitable one. At the end, *Dean* creates a job contract for the candidate chosen by *Commission*.

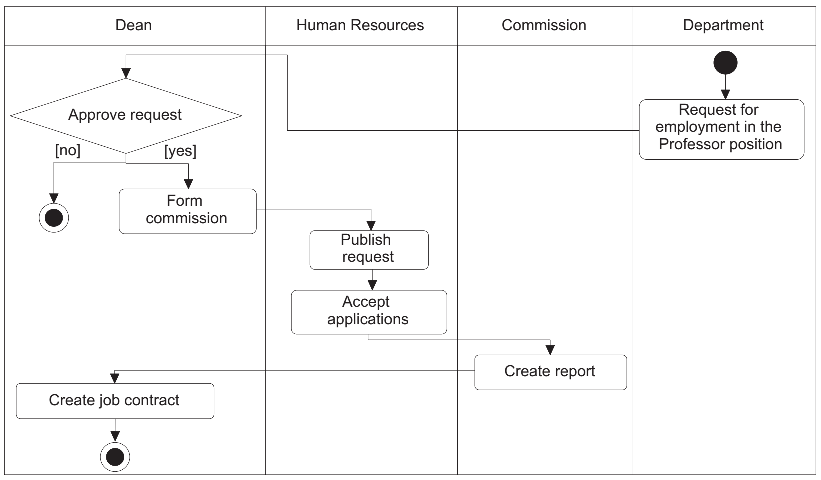

Fig. 5. The simplified version of the employment procedure process

Due to paper limitation and for better clarity, without the loss of generality, we will present two representative access control requirements and show how to implement them by COBAC.

The first requirement presented in this case study demonstrates the purpose of extending RBAC with the notion of activities. It proves that COBAC prototype can be used for "simple" access control requirements when context condition is not required.

The second requirement demonstrates the use of context conditions. It represents the cases when certain roles are dynamically assigned to users based on the information contained in documents that participate in the workflow. Also, these roles will have the permissions to execute certain activities only for the specific process instances depending on users who acquire these roles. To the best of our knowledge this requirement cannot be implemented by any of the workflow access control models presented in Section 2 or its implementation can be overly complicated.

The proof of some other COBAC concepts can be found in [23]. The article [23] demonstrated the use of the COBAC prototype implementation for enforcing access control in workflow systems supporting judicial processes.

Requirement 1 *In order to form the commission, the* Dean *role must be assigned the privilege to execute the activity* Form commission*. Also, this activity* *must be assigned the permission to create, read and modify the document that represents the formal act of forming the commission.*

Requirement 2 *The role* Commission Member *is inappropriate to be statically assigned to the users, but it is necessary to assign this role to the users selected as the commission members by the dean. The role* Commission Member *must be assigned the privilege to execute the activity* Create report *with the condition that commission members can execute this activity only for the process instance for which they are selected. In order to create a report, the activity* Create report *must have permissions to read received applications for this job position and to create, read and modify the document that represents the commission report.*

By analyzing the roles that appear in the process we notice that it is not necessary to introduce a context condition for their enablement/disablement. Therefore, all roles are always enabled: $sc = \sigma \wedge sct = ec$. For this case study we used the role activation algorithm presented in Section 4.1, i.e. all roles assigned to the user are activated when a user logs in to the application.

Requirement 1 represents an example where a context condition is not required for the implementation. Listing 4 shows the COBAC access control policy which represents the implementation of Requirement 1. The *Dean* role has a privilege to execute the activity definition *Form commission*. This activity has the permissions to read, create and modify all documents that belong to the *Formed commission* category.

```
/* role-activity assignment */
$CAD_1$ - activity definition set for the given process
$cad \in CAD_1 \wedge cad =$ Formed commission
$r \in R \wedge r.rn =$Dean
$cc \in CC \wedge cc = \sigma$
rcdActivityAssign($r$, $cad$, $cc$)

/* activity-permission assignment */
$cat \in Cat \wedge cat =$ Formed commission
$op_r \in Op \wedge op_r =$ read document
$op_c \in Op \wedge op_c =$ create documnet
$op_m \in Op \wedge op_m =$ modify document
$p_r \in CP \wedge p_r = (cat, op_r)$
$p_c \in CP \wedge p_c = (cat, op_c)$
$p_m \in CP \wedge p_u = (cat, op_m)$
cadPermissionAssign($cad$, $p_r$, $cc$)
cadPermissionAssign($cad$, $p_c$, $cc$)
cadPermissionAssign($cad$, $p_m$, $cc$)
```

Listing 4. Access control policy for Requirement 1

Requirement 2 represents an example where it is needed to use a context condition in the requirement implementation. In order to define the required context conditions we extend the context model presented in Section 3.1 with information about the process and users (commission members).

The users are represented with the class *User* (see Listing 5). This class is defined as a subclass of the *HumanResource* and *HumanActor* classes. A user is assigned an ID using the *hasUID* relation.

```
@prefix ctx:  <http://informatika.ftn.uns.ac.rs/cobac/context.owl#>.
@prefix ep:   <http://informatika.ftn.uns.ac.rs/cobac/employ-prof-context.owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.

ep:User a owl:Class; rdfs:subClassOf ctx:HumanResource, ctx:HumanActor.
ep:hasUID a owl:DatatypeProperty; rdfs:domain ep:User; rdfs:range xsd:string.
```

Listing 5. The context model of a user

The given process is modeled by the *EPProces* class which is a subclass of the *SoftwareActor* and *SoftwareResource* classes (see Listing 6). A process is assigned an ID using the *hasPID* relation.

```
@prefix ctx:  <http://informatika.ftn.uns.ac.rs/cobac/context.owl#>.
@prefix ep:   <http://informatika.ftn.uns.ac.rs/cobac/employ-prof-context.owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.

ep:EPProces a owl:Class; rdfs:subClassOf ctx:SoftwareActor, ctx:SoftwareResource.
ep:hasPID a owl:DatatypeProperty; rdfs:domain ep:EPProces; rdfs:range xsd:string.
```

Listing 6. The context model of a process

Besides the aforementioned context entities, we also define a specialization of the context expression - *CommissionForProcess* (see Listing 7). This specialization actually contains information who are the commission members for the given process instance. To define this class we introduce the relations *commissionMemberIs* and *forProcess* (see Listing 7).

```
@prefix ctx:  <http://informatika.ftn.uns.ac.rs/cobac/context.owl#>.
@prefix ep:   <http://informatika.ftn.uns.ac.rs/cobac/ employ-prof-context.owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.

ep:commissionMemberIs a owl:ObjectProperty; rdfs:subPropertyOf ctx:hasWhoPart.
ep:forProcess a owl:ObjectProperty; rdfs:subPropertyOf ctx:hasWhatResourcePart.

ep:CommissionForProcess a owl:Class;
     rdfs:subClassOf ctx:ContextExpression;
     owl:equivalentClass[a owl:Class;
     owl:intersectionOf(
           [a owl:Restriction; owl:allValuesFrom ep:User;
              owl:onProperty ep:commissionMemberIs]
           [a owl:Restriction; owl:someValuesFrom ep:User;
              owl:onProperty ep:commissionMemberIs]
           [a owl:Restriction; owl:allValuesFrom ep:EPProces;
              owl:onProperty ep:forProcess]
           [a owl:Restriction; owl:someValuesFrom ep:EPProces;
              owl:onProperty ep:forProcess])].
```

Listing 7. The *CommissionForProcess* context expression

The implementation of Requirement 2 using the COBAC model demonstrates one of the key features of COBAC - role assignment relations that are context-dependent and evaluated at run-time separately for each process instance. This feature facilitates the efficient implementation of workflow systems where access control rights depend on context (in this case, data stored in documents being handled by the workflow). Requirement 2 is implemented with the COBAC policy presented in Listing 8.

Since the commission members for each process instance are users selected by the dean for that specific process instance, the role *Commission Member* is inappropriate to be statically assigned to the users. Therefore the context condition for the *Commission Member* role assignment must be defined. The role *Commission Member* will be assigned to all users who satisfies the context condition cc_r. This context condition, beside necessary namespaces, defines a query which verifies whether there is an instance of the *CommissionForProcess* context expression, in context data, which assigns the current user as a commission member for any process instance. The identifier of the current user is retrieved using the context function *$$currentUserID()$$*.

The user who possesses the *Commission Member* role cannot execute the activity *Create report* for all process instances, but only for process instances for which she/he has been appointed as a commission member. The privilege to execute the instance of the activity *Create report* is assigned to the role *Commission Member* if the context condition cc_{ra} is satisfied. This condition requires that a user with the role *Commission Member* must be a commission member for the current process instance. It defines a query which verifies whether context data contains the *CommissionForProcess* instance which assigns the current user as a commission member for the current process instance. The query of the condition cc_{ra} contains context functions *$$currentUserID()$$* and *$$currentProcessID()$$*. The last one returns the identifier of the current process instance.

The permissions to read received applications for this process instance and permissions to read, create and modify the *Commission report* document are assigned to the instance of the activity *Create report* (see Listing 8).

The appropriate *CommissionForProcess* instance will be inserted into context data upon completion of the *Form commission* activity. After the condition is inserted, it will not be changed until current process instance is completed, when it will be deleted. Therefore we can assume that the both context conditions are at least *activity safe* because their result is unchangeable during the execution of the activity *Create report*.

```
/* dynamic user-role assignment */
$CAI_1^k$ - activity instances of the k-th process instance
$cai \in CAI_1^k \wedge cai =$ Create report

$r \in R \wedge r.rn =$Commission Member
$cc_r \in CC$

$cc_r$ = QUERY {
     PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
     PREFIX ep:  <http://informatika.ftn.uns.ac.rs/cobac/
                  employ-prof-context.owl#>.
     ASK ?X
      WHERE { ?X rdf:type ep:CommissionForProcess.
              ?X ep:commissionMemberIs ?Y.
              ?Y ep:hasUID $$currentUserID()$$ }
     }
```

```
roleCondAssign($r$, $cc_r$)

/* role-activity assignment */
$cc_{ra}$ $\in$ $CC$
$cc_{ra}$ $=$ QUERY {
      PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
      PREFIX ep:  <http://informatika.ftn.uns.ac.rs/cobac/employ-prof-context.owl#>.
      ASK ?X
       WHERE {?X rdf:type ep:CommissionForProcess.
               ?X ep:commissionMemberIs ?Y.
               ?Y ep:hasUID $$currentUserID()$$
               ?X ep:forProcess ?Z.
               ?Z ep:hasPID $$currentProcessID()$$ }
      }
rciActivityAssign($r$, $cai$, $cc_{ra}$)

/* activity-permission assignment */
$doc_1$ $\in$ $Res$ $\wedge$ $doc_1 =$ Job application (for the k-th process instance)
$doc_2$ $\in$ $Res$ $\wedge$ $doc_2 =$ Commission report (for the k-th process instance)
$op_r$ $\in$ $Op$ $\wedge$ $op_r =$ read document
$op_c$ $\in$ $Op$ $\wedge$ $op_c =$ create document
$op_m$ $\in$ $Op$ $\wedge$ $op_m =$ modify document
$p_{1r}$ $\in$ $RP$ $\wedge$ $p_{1r} = (doc_1,\ op_r)$
$p_{2r}$ $\in$ $RP$ $\wedge$ $p_{2r} = (doc_2,\ op_r)$
$p_{2c}$ $\in$ $RP$ $\wedge$ $p_{2c} = (doc_2,\ op_c)$
$p_{2m}$ $\in$ $RP$ $\wedge$ $p_{2m} = (doc_2,\ op_m)$
$cc_p$ $\in$ $CC$ $\wedge$ $cc = \sigma$
caiPermissionAssign($cai$, $p_{1r}$, $cc_p$)
caiPermissionAssign($cai$, $p_{2r}$, $cc_p$)
caiPermissionAssign($cai$, $p_{2c}$, $cc_p$)
caiPermissionAssign($cai$, $p_{2m}$, $cc_p$)
```

Listing 8. Access control policy for Requirement 2

Since this case study covers only one business process, the activity list in the prototype application will contain only activity instances of the given process which are allowed to be executed at the given time by the logged in user. The activity list is created as presented in Section 4.2.

6. Conclusion

Beside standard features, the workflows access control should support task level access control with possibility to assign different roles to users and different permissions to roles during process execution. In some cases, access control policies should be defined on a process definition (task definition) level, while in other cases they should be defined on a process instance (task instance) level. Also, access control may depend on different context factors, which may vary form process to process. In this paper we present the *Context-sensitive access control model for business processes* (COBAC) that supports these requirements.

The COBAC model is the RBAC-based model which is extended for use in workflow-oriented business systems. Beside RBAC entities, we introduce the following entities: *business process*, *activities*, *context* and *resource categories*. Since the model is based on RBAC, all advantages of the RBAC model are used in COBAC. By introducing the *business process* and *activities* entities it is possible to define access control policies for business processes more efficiently.

The *context* is used to handle access control requirements which depend on factors from the system and from the system's environment. The *categorization* of the resources enables the definition of access control policies for whole resource category, and thus, potentially, reduces the number of policies which need to be defined.

The most notable features of the COBAC model are:

- It is based on the RBAC model which is extended with the concept of business process and activity in order to bind certain access control segments with activities instead of binding them with the current session, and thus provide efficient definition of different policies and constraints in different business systems. This ensures the independence of these segments from the number of sessions in which activities are executed. Also the business process and the activity provide a mechanism for the fine-grained control of least privileges.
- The policies can be defined at the level of each specific process instance (activity instance) and at the process (activity) definition level. The policies defined at the process definition level will apply to all process instances.
- During the time/workflow execution, different roles can be assigned/ unassigned to a user and roles can have different permissions.
- The assignment relations in the COBAC model are extended with the context-dependant condition and thus provide support for the context-sensitive access control. Also, the notion of the role is extended with the context information. The proposed context model is developed using ontologies.
- Resources in the COBAC model can be hierarchically organized thus lowering the number of policies that need to be defined.

The COBAC model is verified on a real workflow system based on the exchange of documents. The presented verification represents the proof of the proposed model practical value.

Our experience in using the COBAC prototype in the case study presented in Section 5 and for the judicial process presented in [23] showed that defining contexts using ontologies can be difficult for the most information security officers due to lack of experience in the use of ontologies. Therefore research on a DSL (domain specific language) for describing context in order to enable easy context definition is in progress. The proposed COBAC model does not consider the quality of the context information, like accuracy, reliability, etc. This information can be very important for access control in certain cases. A further direction in the development of the COBAC model includes the use of the context quality in access control enforcement. Since there is a large number of interorganizational business processes today, the distributed version of COBAC is another further research direction. Also, we plan to investigate how policies can be represented using XACML (eXtensible Access Control Markup Language) and executed using XACML implementations.

References

1. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. pp. 304–307. Springer-Verlag, London, UK (1999)
2. Abowd, G.D., Mynatt, E.D., Rodden, T.: The human experience. IEEE Pervasive Computing 1(1), 48–57 (2002)
3. Bao, Y., Song, J., Wang, D., Shen, D., Yu, G.: A role and context based access control model with UML. In: International Conference for Young Computer Scientists. vol. 0, pp. 1175–1180. IEEE Computer Society, Los Alamitos, CA, USA (2008)
4. Bertino, E., Bonatti, P.A., Ferrari, E.: TRBAC: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. 4(3), 191–233 (2001)
5. Bertino, E., Catania, B., Damiani, M.L., Perlasca, P.: GEO-RBAC: a spatially aware RBAC. In: SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies. pp. 29–37. ACM, New York, NY, USA (2005)
6. Bhatti, R., Bertino, E., Ghafoor, A.: A trust-based context-aware access control model for web-services. Distributed and Parallel Databases 18(1), 83–105 (2005)
7. Bhatti, R., Bertino, E., Ghafoor, A., Joshi, J.B.: XML-based specification for web services document security. Computer 37(4), 41–49 (2004)
8. Botha, R.A., Eloff, J.H.P.: Access control in document-centric workflow systems – an agent-based approach. Computers & Security 20(6), 525 – 532 (2001)
9. Buettcher, S., Clarke, C., Cormack, G.: Information Retrieval: Implementing and Evaluating Search Engines. MIT Press (2010)
10. Cohen, H., Lefebvre, C.: Handbook of Categorization in Cognitive Science. Elsevier (2005)
11. Corradi, A., Montanari, R., Tibaldi, D.: Context-based access control for ubiquitous service provisioning. Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC) 1, 444–451 (2004)
12. Covington, M.J., Long, W., Srinivasan, S., Dev, A.K., Ahamad, M., Abowd, G.D.: Securing context-aware applications using environment roles. In: Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 10–20. ACM, New York, NY, USA (2001)
13. Damiani, M.L., Bertino, E., Catania, B., Perlasca, P.: GEO-RBAC: A spatially aware RBAC. ACM Trans. Inf. Syst. Secur. 10(1), 2 (2007)
14. Davenport, T.H., Short, J.E.: The new industrial engineering: Information technology and business process redesign. Sloan Management Review 31(4), 11–27 (1990)
15. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5(1), 4–7 (2001)
16. Emami, S.S., Amini, M., Zokaei, S.: A context-aware access control model for pervasive computing environments. Proceedings of the IEEE International Conference on Intelligent Pervasive Computing (IPC) 0, 51–56 (2007)
17. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security (TISSEC) 4(3), 224–274 (2001)
18. Filho, J.B., Martin, H.: Using context quality indicators for improving context-based access control in pervasive environments. In: EUC '08: Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. pp. 285–290. IEEE Computer Society, Washington, DC, USA (2008)

19. de Freitas Bulcao Neto, R., da Graca Campos Pimentel, M.: Toward a domain-independent semantic model for context-aware computing. In: Proceedings of the 3rd Latin American Web Congress (LA-WEB). pp. 61–70. IEEE Computer Society, Washington, DC, USA (2005)
20. Frey, T., Gelhausen, M., Saake, G.: Categorization of concerns: a categorical program comprehension model. In: Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools. pp. 73–82. ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2089155.2089171
21. Gao, L., Zhang, L., Xu, L.: Access control scheme for workflow. In: ICCET '09: Proceedings of the 2009 International Conference on Computer Engineering and Technology. pp. 215–217. IEEE Computer Society, Washington, DC, USA (2009)
22. Georgiadis, C.K., Mavridis, I., Pangalos, G., Thomas, R.K.: Flexible team-based access control using contexts. In: SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies. pp. 21–27. ACM, New York, NY, USA (2001)
23. Gostojic, S., Sladic, G., Milosavljevic, B., Konjovic, Z.: Context-sensitive access control model for government services. Journal of Organizational Computing and Electronic Commerce 22(2), 184–213
24. Haibo, S., Fan, H.: A context-aware role-based access control model for web services. Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE) 0, 220–223 (2005)
25. Han, W., Zhang, J., Yao, X.: Context-sensitive access control model and implementation. Proceedings of the 5th International Conference on Computer and Information Technology (CIT) 0, 757–763 (2005)
26. Hollingsworth, D.: Workflow management coalition the workflow reference model. Workflow Management Coalition, Technical Report, TCOO-1003 (1995)
27. Irwin, K., Yu, T., Winsborough, W.H.: Enforcing security properties in task-based systems. In: SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies. pp. 41–50. ACM, New York, NY, USA (2008)
28. Jian-Min, Z., Xiao-Chun, L.: A modified model for flexible workflow access control. In: International Symposium on Computational Intelligence and Design. pp. 279–282. IEEE Computer Society (2011)
29. Kapsalisa, V., Hadellisb, L., Karelisb, D., Koubiasc, S.: A dynamic context-aware access control architecture for e-services. Computers & Security 25(7), 507–521 (2006)
30. Koufi, V., Malamateniou, F., Mytilinaiou, E., Vassilacopoulos, G.: An event-based, role-based authorization model for healthcare workflow systems. In: Szomszor, M., Kostkova, P. (eds.) Electronic Healthcare, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 69, pp. 221–228. Springer (2012)
31. Latif, U., Joshi, J.B.D., Bertino, E., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Trans. on Knowl. and Data Eng. 17(1), 4–23 (2005)
32. Leitner, M., Rinderle-Ma, S., Mangle, r.: Aw-rbac: Access control in adaptive workflow systems. In: International Conference on Availability, Reliability and Security. pp. 27–34. IEEE Computer Societ (2011)
33. Liscano, R., Wang, K.: A context-based delegation access control model for pervasive computing. In: AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops. pp. 44–51. IEEE Computer Society, Washington, DC, USA (2007)

34. Ma, C.h., Lu, G.d., Qiu, J.: An authorization model for collaborative access control. Journal of Zhejiang University SCIENCE C (Comput & Electron) 11(9), 699–717 (2010)
35. Oh, S., Park, S.: Task-role-based access control model. Information Systems 28(6), 533–562 (2003)
36. Papagiannakopoulou, E., Koukovini, M., Lioudakis, G., Garcia-Alfaro, J., Kaklamani, D., Venieris, I.: A contextual privacy-aware access control model for network monitoring workflows: Work in progress. In: Garcia-Alfaro, J., Lafourcade, P. (eds.) Foundations and Practice of Security, Lecture Notes in Computer Science, vol. 6888, pp. 208–217. Springer Berlin / Heidelberg (2012)
37. Pigeot, C.E., Gripay, Y., Scuturici, M., Pierson, J.M.: Context-sensitive security framework for pervasive environments. In: ECUMN '07: Proceedings of the Fourth European Conference on Universal Multiservice Networks. pp. 391–400. IEEE Computer Society, Washington, DC, USA (2007)
38. Russello, G., Dong, C., Dulay, N.: A workflow-based access control framework for e-health applications. In: AINAW '08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops. pp. 111–120. IEEE Computer Society, Washington, DC, USA (2008)
39. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proc of IEEE Workshop on Mobile Computing Systems and Applications. pp. 85–91. IEEE Computer Society, Washington, DC, USA (1994)
40. Shafiq, B., Samuel, A., Ghafoor, H.: A GTRBAC based system for dynamic workflow composition and management. In: Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC). pp. 284–290. IEEE Computer Society, Los Alamitos, CA, USA (2005)
41. Shang, C., Yang, Z., Liu, Q., Zhao, C.: A context based dynamic access control model for web service. In: International Conference on Embedded and Ubiquitous Computing, IEEE/IFIP. vol. 2, pp. 339–343. IEEE Computer Society, Los Alamitos, CA, USA (2008)
42. Sladić, G., Milosavljević, B., , Surla, D., Konjović, Z.: Flexible access control framework for MARC records. The Electronic Library 30(5), 623–652 (2012)
43. Sladić, G., Milosavljević, B., Konjović, Z., Vidaković, M.: Access control framework for XML document collections. Computer Science and Information Systems (ComSIS) 8(3), 591–609 (2011)
44. Soergel, D.: Organizing Information: Principles of Data Base and Retrieval Systems. Academic Press Professional, Inc., San Diego, CA, USA (1985)
45. Sun, Y., Pan, P.: PRES: a practical flexible RBAC workflow system. In: ICEC '05: Proceedings of the 7th international conference on Electronic commerce. pp. 653–658. ACM, New York, NY, USA (2005)
46. Thomas, R.K., Sandhu, R.S.: Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented autorization management. In: Proceedings of the IFIP TC11 WG11.3 11th International Conference on Database Securty XI. pp. 166–181. Chapman & Hall, Ltd., London, UK, UK (1998)
47. Tripathi, A.R., Kulkarni, D., Ahmed, T.: A specification model for context-based collaborative applications. Pervasive Mob. Comput. 1(1), 21–42 (2005)
48. Truong, K.N., Abowd, G.D., , Brotherton, J.A.: Who, what, when, where, how: Design issues of capture & access applications. In: Ubicomp 2001: Ubiquitous Computing. pp. 209–224. Springer-Verlag, New York, NY, USA (2001)
49. W3C: Web Ontology Language (OWL). W3C Recommendation (2004), http://www.w3.org/TR/owl-features/

50. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC - a workflow security model incorporating controlled overriding of constraints. International Journal of Cooperative Information Systems 12(4), 455–485 (2003)
51. Wang, B., Zhang, S.: An organization and task based access control model for workflow system. In: Advances in Web and Network Technologies, and Information Management. pp. 485–490. SpringerLink, Berlin, DE (2007)
52. Xu, W., Wei, J., Liu, Y., Li, J.: SOWAC: A service-oriented workflow access control model. In: COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference. pp. 128–134. IEEE Computer Society, Washington, DC, USA (2004)
53. Yao, L., Kong, X., Xu, Z.: A task-role based access control model with multi-constraints. In: NCM '08: Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management. pp. 137–143. IEEE Computer Society, Washington, DC, USA (2008)
54. Zhang, L., Luo, L., Zhang, L., Geng, T., Yue, Z.: Task-role-based access control in application on MIS. In: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC). pp. 153–159. IEEE Computer Society, Washington, DC, USA (2006)
55. Zhang, W., Zhang, K.: A role-based workflow access control model. In: ETCS '09: Proceedings of the 2009 First International Workshop on Education Technology and Computer Science. pp. 1136–1139. IEEE Computer Society, Washington, DC, USA (2009)
56. Zhao, H., Fang, Z., Xu, P., Zhao, L., Liu, J., Wang, T.: An improved role-based workflow access control model. Information Technology: New Generations, Third International Conference on 0, 551–556 (2008)

Goran Sladić is holding the assistant professor position at the Faculty of Technical Sciences, Novi Sad, Serbia since 2011. Mr. Sladić received his Bachelor degree (2002), Master degree (2006) and and PhD degree (2011) all in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 2002 he is with the Faculty of Technical Science in Novi Sad. His research interests include access control, document management systems, XML technologies, context-aware computing and workflow systems.

Branko Milosavljević is holding the associate professor position at the Faculty of Technical Sciences, Novi Sad, Serbia since 2008. Mr. Milosavljević received his Bachelor degree (1997), Master degree (1999), and PhD degree (2003) all in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 1998 he is with the Faculty of Technical Science in Novi Sad.

Zora Konjović is holding the full professor position at the Faculty of Technical Sciences, Novi Sad, Serbia since 2003. Mrs. Konjović received her Bachelor degree in Mathematics from the University of Novi Sad, Faculty Science in 1973, Master degree (1985) and Ph. D. degree (1992) both in Robotics from the University of Novi Sad, Faculty of Technical Sciences. Since 1973 till 1980 she was with the Faculty of Science in Novi Sad, and since 1980 she is with the Faculty of Technical Sciences, University of Novi Sad.

Received: September 7, 2011; Accepted: April 11, 2013.