Security challenges in adaptive e-Health processes

Michael Predeschly, Peter Dadam, Hilmar Acker

Institute DBIS, University Ulm
E-mail: firstname.lastname@uni-ulm.de

Abstract. E-health scenarios demand system-based support of process-oriented information systems. As most of the processes in this domain have to be flexibly adapted to meet exceptional or unforeseen situations, flexible process-oriented information systems (POIS) are needed which support ad-hoc deviations at the process instance level. However, e-health scenarios are also very sensitive with regard to privacy issues. Therefore, an adequate access rights management is essential as well. The paper addresses challenges which occur when flexible POIS and adequate rights management have to be put together.

1 Introduction

The personnel in clinical domains have to deal with a large number of different processes. Due to high workloads, exceptional situations, frequently changing diagnostics, treatments, and accounting procedures the risk of errors is pretty high. For that reason it is widely acknowledged that adequate process-oriented information systems (POIS) could help to improve this situation [1]. However, clinical processes are not static by nature. Therefore, POIS have to be flexible, i.e. to allow ad-hoc deviations at the process-instance level [2]. While good progress has been made in understanding how to build such powerful process management systems [3], little attention has been paid so far how an adequate access rights management system (RMS) for such systems should look like. In "classical" information systems, a few administrators are authorized to assign access rights and privileges to users. However, in flexible POIS even end-users may be authorized to perform modifications to processes at the instance level, i.e. to insert, delete, or move process steps. In this context the question arises, for example, what kind of rights should be granted to an end-user to be able to insert a new step and to assign access and execution rights to this newly inserted step without being too restrictive at the one side or to completely undermine the RMS at the other side. Some of the issues related to access rights management in conjunction with flexible POIS shall be discussed in the following.

2 Challenges and Problems

Contemporary RMS have been designed to enable or to restrict access of users to functions and data in "classical" information systems. In such information systems users may come and go but the information systems themselves, i.e. the functions they are providing, are rather static. If these information systems are implemented in the traditional, monolithic fashion, access rights management is typically handled in a centralized way. Thus such systems have an RMS component which manages which users have which kind of permission to execute which application functions and/or which kind of access to data has been granted to them.

These RMS maintain some kind of rights matrix which associates subjects (users) and objects (functions, data) with access rights. If many subjects (s) and objects (o) have to be handled, this rights matrix becomes very large if it is directly stored as full-fledged $s \times o$ matrix. Therefore, typically only one dimension of this rights matrix is physically stored as a so called access control list (ACL). This ACL represents for every object the list of users which have access to it along with the corresponding permissions. The other dimension, namely the association between users and the objects they are allowed to access is either not supported at all or must be computed by inspecting all ACLs.

In environments with high security requirements, one wants to ensure that the security rules are always obeyed. Thus one wants to enforce *mandatory access control*. In such environments, typically the assignment of access rights to users follows the need-to-know principle. In addition, constraints like separation of duties or the enforcement of the four-eyes-principle also have to be supported. To implement such principles, one usually assigns security levels or permissions to users and respective qualifications to the objects. Only if a user's permissions matches the respective requirements of the object, the access is granted. In order to enforce mandatory access control, the RMS is typically implemented as an "active" component sometimes called a reference monitor [4] which controls the access to functions and objects.

All these aspects are well known and the alternative approaches to implement them are pretty well understood in the area of "classical" monolithically implemented information systems. This picture changes significantly, however, when flexible POIS have to be realized. Firstly, such systems are typically no longer implemented in a monolithic fashion. Instead, the information system consists of separate, individually invokable application functions ("services") which interact with each other according to a process schema which is executed by a process engine. In principle, each process schema constitutes an own schema-specific RMS which regulates which users (based on their roles, organizational units they belong to, etc.) can execute which process steps. If the processes are static, i.e. all process instances execute according to the process schema without any deviations, then there is not much difference to the "classical" monolithic information systems, at first glance. However, in many cases these application functions come from different (and often even foreign) sources and have not been designed to cooperate with a centralized RMS or with a reference monitor.

The situation becomes even worse, if we have to deal with flexible POIS, i.e. systems which may deviate from the pre-planned execution sequence by inserting new process steps, deleting process steps, or moving process steps to another place in the process schema. Furthermore it is possible to individually change the pre-planned actor assignments at the instance level. In this case every process instance now constitutes an independent instance-specific RMS (see Fig. 1. Instance rights can diverge from the schema rights

).

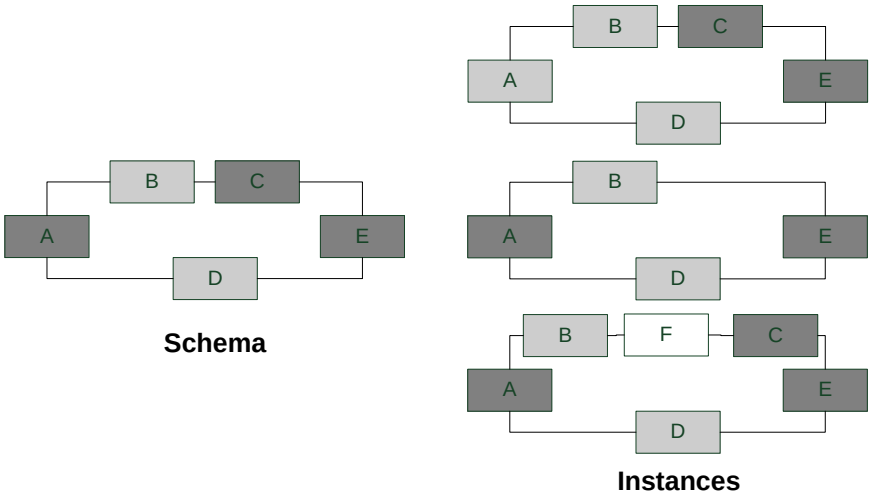

Fig. 1. Instance rights can diverge from the schema rights

Another problem is the granularity of rights used in contemporary POIS. Usually, actor assignment expressions are used to describe for a given step who is authorized to execute this step. The expression `orgunit = "radiology" AND role = "doctor"` would mean that the actor (user) has to be a doctor from the radiology department. As mentioned above, application functions are associated with these steps. Thus when executing this process step the actor is allowed to use the complete functionality this application provides. This would cause no problem if these application functions would be limited to the task to be performed. We will address this issue in the following section.

The rest of the paper is organized as follows: Section 3 discusses challenges of RMS in the context of component-oriented composition of POIS. Section 4 deals with constraints. In section 5 storage-aspects of rights information are treated and section 6 deals with the administration of RMS. Then we want to give a short introduction in the solutions we want to integrate in today's POIS. This solution is introduced in section 7. Finally, section 8 gives a short conclusion of the paper.

3 Components

As mentioned above, actor assignment expressions are used in POIS to decide which users are authorized to execute a certain process step.

Assuming the user's task is to view a certain document and, therefore, a general purpose text processor like Word for Windows, for example, is associated as application function. Once invoked, such a program would in many cases not only allow to read a document, but also allow to modify it, to store it, to print it, or to even invoke some macros which may do very strange things. This is no specific problem of such a text processor but is rather typical for most application functions or components. They are usually implemented in a generic way to support a broad variety of application domains. As a consequence the problem arises how to constrain such application functions accordingly.

To deal with this kind of problem component systems like .NET [5] offer the possibility to implement some component specific access rights management. This is done by special annotations in the source code which are used by the runtime environment of the component to check whether the user has the required privileges.

Using this approach the implementer of the component decides at the source code level which privileges are required for a user to invoke certain functions of the component. For POIS which may integrate a large variety of different application functions coming from different components, this approach is problematic for at least two reasons: Firstly, every component implementer introduces names for privileges independent from others (name inflation problem), and secondly, changes to the RMS will require changes at the source code level of the affected components in many cases.

Similar problems arise when using security frameworks like JAAS [20]. JAAS is based on PAM (*Pluggable Authentication Module* [21]) and mainly supports configurable user authentication. Thus like in .NET the method used for authenticating users must not be hard coded in the source code. Instead, the concrete authentication mechanism (e.g. by password, fingerprint or even retina scan) is implemented by individual plugins. Which concrete plugin should be used for identifying the user can be chosen in configuration files by the administrator. On the other hand JAAS also extends the Java 2 security policies so that not only the code origin but also the current user can be taken into account when the security manager needs to decide whether access from the sandbox to system resources is granted or not [19]. Although not intended to, this can be used to secure internal application functions. But therefore the developer has to encapsulate the different functions in separate classes implementing special interfaces. This leads to name inflation problems and to a quite bad system design as well.

Therefore, such approaches are not suitable for POIS. Instead a (logically) centralized approach is needed. Today, this works only in environments where all components come from the same vendor. A general vendor-independent approach is still missing.

Web services are a variant of this component oriented software development and, thus, have similar problems, in principle. So far, the main focus of web services (WS)

is on establishing trustful communication and on Quality-of-Service (QoS) aspects, however. Standards such as WS-Security [6] and its extension WS-Trust [7] are addressing such issues. A centralized approach to method-based access rights management is only discussed in [8]. [8] suggests a layer model. Rights at the bottom layer are associated with the functions provided by the WS. A set of functions can be bundled into rights packages, so-called "keys", and passed on to the next higher level. There these keys can be combined to new keys and propagated to next higher level and so on. However, changes performed at lower levels have significant effects on the roles defined in higher levels. Changes to the rights at higher levels usually lead to major changes of keys at lower levels.

A problem, especially in the clinical domain, may be that the permission for a given user to invoke a certain application function may be limited to a certain type of process and perhaps even to a certain place within this process. E.g., the function to administer an examination may only be allowed within treatment and diagnostic processes but not stand-alone. To read and modify a document (using a certain application function) may be allowed during some creation phase but may be forbidden after the document has been published. In addition, the permission to execute this function may be even dependent on the data or document to be accessed. In total, this means that a rather fine-granular access rights management is required.

4 Storage Aspects

As described above, ACLs are typically used to store (parts of) the rights matrix in a compact manner. Unfortunately, this approach is not very satisfying in the POIS context, because efficient access along both dimensions is necessary. Whenever responsibilities of users are changed, users leave the organization or certain structural changes in the organization happen, one has to determine which objects (and thus their ACLs) are affected. Without adequate support of the user $\rightarrow$ object dimension this will result in an exhaustive search through all ACLs. And there may be many of them! Because fine-granular access rights management is needed (see above), it's not sufficient to store just one ACL per object (application function). One has also to discriminate in which process schema and at what position this application function is used. This, in turn, increases the number of ACLs significantly because a component may easily have hundreds of occurrences of this kind.

The situation worsens when flexibility comes into the game, i.e. new process steps are inserted at process instance level. Modifying process instance by inserting a new step, for example, means to dynamically create a new, individual process schema. This, in turn, leads to a new ACL because access rights have to associated with this step. As a consequence, one has to maintain a very dynamic and potentially large rights matrix. Solutions have to be found for both, efficient access along both dimensions and for adequate storage representations. Efficient access becomes a very important factor here, because the number of users which can modify rights increases in such flexible environments significantly (see introduction) and thus the number of

accesses to the RMS will very likely significantly increase as well.

A real world example of this problem can be found in the EU project Webocrat [9], where an e-government solution is being developed. In small installations, such as single municipalities, efficient access to the rights matrix is not a big problem. However, in large application domains like a whole county with a lot of users who interact with the RMS, the realization of efficient access is a big problem. One approach under investigation here is whether some kind of intelligent caching can help to improve the situation [10].

5 Constraints

In addition to simple rights, like execute permissions, an RMS has usually also to obey several kinds of dependencies or constraints. *Separation of duties*, where the execution of a certain step by a certain user restricts the set of actors which are allowed to execute subsequent steps, is an example for such a constraint. Another example is *binding*, where a set of process steps has to be executed by the same actor. This means that once the first step of this set is executed by a certain actor, "binding" for the other steps to this set is fixed.

Such constraints may also span multiple processes. This poses new challenges in case of flexible processes. If process steps are inserted, deleted, or moved to another position at the process instance level, it must be immediately (and efficiently) checked that no such constraint is violated. The constraints specified at build time must be also valid at runtime. New constraints have to be integrated and need to be checked if they stay in conflict to the yet specified ones. Another problem occurs in the context of process modifications: when inserting new process steps the user is only allowed to assign a subset of the services available in the system. For example an employee adds a new step in a process. If he is not responsible for financial transactions assigning financial services should not be allowed in order to prevent e.g. financial transactions from the companies account to the account of the employee.

This kind of problems is partly addressed by W-RBAC (Workflow - Role Based Access Control) [11]. In W0-RBAC constraints like separation of duties as well as binding across instance boundaries can be defined. The extension W1-RBAC addresses the selective replacement and adaptation of conditions to correct modeling errors at runtime or to handle unforeseen cases. How to support these constraints at the process instance level, especially in the context of flexible POIS is not addressed and will be a major challenge.

Time-based restrictions of rights are considered in GTRBAC (Generalized Temporal Role Based Access control) [12][13]. The system provides a comprehensive set of time-based conditions which cover arbitrary periodic time periods as well as single events. Conditions can be related to fixed points in time or to events. The approach is general and not related to processes. The application of these concepts to POIS and to perform the necessary checking at the process instance level is a non-trivial task, especially when flexibility comes into the game.

6 Administration

As can be seen from the discussion above, the definition of appropriate types and levels of rights and constraints on the one side and their association with users and application functions (and their different occurrences) on the other side is a non trivial task. Typically, some kind of structuring or leveling is used to make this task manageable. On the users' side usually organizational units (departments, projects,…) and roles are used to form groups of users having the same rights to perform certain tasks. On the objects' side usually hierarchical structures are used to form appropriate groups.

Systems which address the users' aspects are ARBAC97 [14] and ARBAC02 [15] (Administrative Role Based Access Control) and the extension RBACAM (RBAC Administration Model) [16]. ARBAC97 distinguishes between three elements of descriptions: users, roles, and permissions. Users can be associated with roles, roles can be associated with other roles (in order to form hierarchies), and roles can be associated with permissions. ARBAC02 extends this approach by introducing organizational structures and hierarchies of permissions. RBACAM (RBAC administration model) complements ARBACxx by adding verification facilities to check the modeled rights. These approaches point in the right direction. However, it remains to be elaborated how these concepts can be applied to flexible POIS and how the support of constraints can be integrated in such an RMS.

Using hierarchical structuring of objects in order to simplify access rights management is a well known principle and used, for example, in the directory structure of file systems. One single entry at the parent directory level is sufficient to set the access rights for all files in that directory. SAM Jupiter [17] simply applies this principle to data objects, with finer granularity. The grouping is done here according to responsibilities. It is rather unclear, however, to what extend this approach can be utilized to simplify access rights management in POIS.

Another important aspect are organizational changes and their impact on access rights management. When departments are closed, outsourced, or combined with other departments, actor assignment expressions at the process schema as well as at the process instance level have to be adjusted accordingly. A detailed discussion of this problem can be found in [18].

All approaches introduced here require a deep understanding of the RMS and knowledge from the user which privileges and permissions exists and how they have to be associated with users, roles, assignment expressions, and application functions such that they show the desired effects. With the increased number of persons in flexible POIS who interact with the RMS, user interface aspects become a very important issue. Only with a proper model, with few and easy to understand basic concepts, one has a chance to enable end-users to deal with access rights managements in a safe way.

7 Short introduction to ARMS

All challenges presented so far have no adequate solution in today's RMS. We propose an integration of all these topics in one approach we call it ARMS (adaptive rights management system). Here, we want to introduce the concepts we have considered so far.

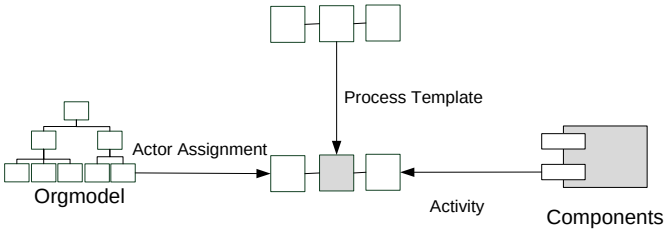

Figure 1: security policies in process oriented information systems

In RMS for process oriented information systems like WRBAC [11] or in IBMs MQ Workflow the arrangement of security policies is managed as in **Figure 1**. The process is created and (as already mentioned above) an actor assignment is the basis for the access rights management. The first problem about this solution is that the actor assignment doesn't have to be equivalent with the rights specified at a process step. The second one is that an actor assignment is done at build time of the process and cannot be changed at runtime. This means the rights information in such systems is very static. One exception is the concept of W1-RBAC in WRBAC that makes it possible to modify and change access rules at runtime.

Therefore ARMS shall be different from other systems and divide the access rights management in two phases. Phase one is the build time phase. In this phase a lot of validation of the modeled rights are carried out to support the administrator or process modeler. This helps to deal with defined constraints or detecting actor assignments that do not fit the rights basis.

For example if there is a constraint that a service only can be used by actors out of the group of permanent employees and if a process modeler makes actor assignments that comprises a person that do not fit this constraint ARMS gives the instruction to change the assignment towards a correct one. ARMS therefore resolves the assignment and tells the user which actors mustn't be included or must be included because of other constraints. This could be the case if an application should always be supervised by a special employee, for example.

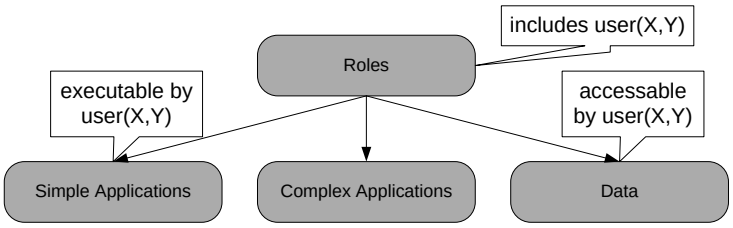

Figure 2: different positions for rights validation at build time

There are many other constraints on groups (users that have to be integrated in groups with a special kind of user) or data (data is accessible to users or not). In ARMS all these could be efficient checked at build time. We need these validations at build time to simplify the work of the process designer (see Figure 3) and avoid unnecessary runtime errors.

Phase two is the runtime phase. When thinking of flexible POIS supporting deviations at the process instance level, these changes are often done by normal employees. Therefore these checks are absolutely required to support the untrained user in the interaction with the RMS. This leads to a great usability and security problem which we want to avoid with these validation so the user cannot make "wrong" decisions.

So at process runtime extensive checks have to be done and although these checks need much time to be validated they must not slow down process execution. This will be solved through the calculation of special checksums on the constraints that can be checked faster.

The question therefore is which user has the right to perform which modification on the process? Solutions for this problem are a clear separation of concerns, which guarantees that no access rule is violated. In ARMS we want to realize this with a combination of roles of authorized persons and the abilities stored in an organization model. Users are assigned to roles which have different rights like in other systems. Additional to these roles the user has an ability field assigned to him where special abilities can be stored to extend a role.

For realizing all the different security features mentioned so far just preventing the execution of a service for a certain user is not enough. For some of them the application even must have the opportunity to interact with the ARMS – therefore we have divided the rights that can be specified for a component in several layers (see **Figure 3**) according to the functionalities of the service and the kind of security which should be expressed.

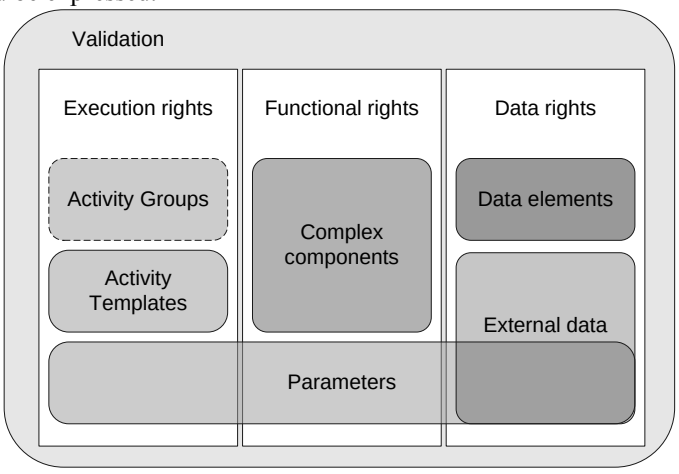

Figure 3: Layer architecture for component integration into ARMS

The problem of the components is that they may come from different sources and need to be integrated in a complex environment with many boundary conditions. The main problem therefore is that the developer of the application should as far as possible not be engaged with the integration of his component into ARMS. So our multi layer environment makes it possible to keep the developer away from many of the access rights management activities.

At the execution rights layer simple components (i.e. ones which only provide the necessary functionality for the respective process step) can be integrated in ARMS by using Activity Templates (AT). An AT represents exactly one service function in a POIS. In such an AT the default rights information is stored and ARMS uses this information when performing its checks. Therefore the component developer doesn't have to integrate any special, security related functionality. Activity Groups are only a hierarchisation of ATs to simplify administration.

However components often not only provide the necessary functionality. In fact they often try to support different scenarios (like Word described above). Such components we call complex components, because the integration might not be done without any further work of the component developer.

One can distinguish two kinds of complex components. The first one supports parameterization so the needed functionality can be chosen before the call is actually made. In ARMS we support such components also by using ATs. Therefore the number of ATs for one component is not limited. Functional parameters can be hard coded in an AT which leads to at least one AT per component specification. With this no security related application logic by the developer is needed.

The second kind of complex components offers a lot of functionality when the component is running. But they not support to limit the available functions by parameterization of the component call. In order to integrate such components into ARMS it is necessary that the developer does some extra work. At the moment we are working on an interface to make this integration also simple for the developer.

The integration of data in the system is mainly done by parameterization of applications (see **Figure 4**). The scope of a single parameter is narrowed or exclusive selected to a special value to customize the data that can be accessed.

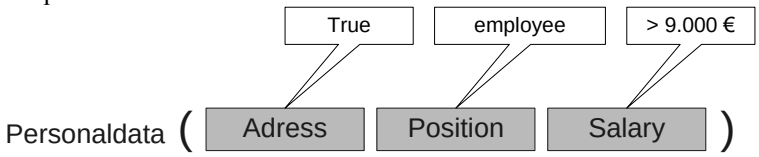

Figure 4: parameterization of methods to integrate data in the RMS

Through this customization the data from foreign sources can be encapsulated and integrated in the RMS. Alternatively the data can be integrated into the POIS through data elements. Therefore the data is copied to this data element and needs to be synchronized with the external data structure.

Additionally to the integration of the applications is the number of potentially used applications in a POIS. In a new system there probably are only a few applications installed and used. During the lifecycle of the system the number of components increases not as the number of stored processes schemas and instances from these schemata's increases. As mentioned above this data cannot be stored in ACLs or

Capability lists so we have to split the classical matrix into instance specific matrices which makes it possible to handle the stored information. These instance matrices only store the process instance relevant data. Only components and users which are assigned to the instance are mentioned in this table.

As a result we can say ARMS shall become a prototype for adaptive POIS-RMS which needs to address all here presented challenges. We therefore have to extend our current approach and to finish the implementation of our current concepts.

8 Conclusion

The discussion above has shown that access rights management is a complex task with many facets and many open questions in the context of flexible POIS have still be to solved. We then give a short introduction in the construction of a solution we want to implement. The main purpose of this paper was to raise awareness of these issues and show that solutions for these problems must be found in the near future.

We are working on these solutions in the context of ARMS and will focus our work on the component integration and the efficient storage with included validation of the rights information. When these challenges are mastered we can go into further details of the administration aspects.

References

[1] Dadam P., Reichert M.: Towards a New Dimension in Clinical Information Processing. Proc. MIE2000/GMDS 2000, Hannover, Sept. 2000, IOS Press, 2000, pp. 295-301

[2] Dadam P., Reichert M., Kuhn K.: Clinical Workflows – The Killer Application for Process-oriented Information Systems?, Proc 4th Int'l Conf. on Buisness Information Systems BIS 2000, Poznan, Poland, April 2000, pp. 36-59

[3] Reichert M., Rinderle S., Kreher U., Dadam P.: Adaptive Process Management with ADEPT2. Proc. Int'l Conf. on Data Engineering, ICDE '05, Tokyo, April 2005, Demo Session, pp. 1113-1114

[4] Anderson J. P., Computer security technology study ESD-TR-73-51, Vol2

[5] Microsoft Library: Security in the .NET Framework, 2007, http://msdn2.microsoft.com/en-us/library/fkytk30f.aspx

[6] Oasis WSS: SOAP Message Security http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[7] Oasis WS-Trust 1.3 http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html

[8] Payne C., Thomson D., Bogle J., O'Brien R.: Napoleon: A Recipe for Workflow, Computer Security Application Conference 1999, pp. 134

[9] Dridi F., Pernul B.M. & G.: Administration of an RBAC system, Proceedings of the 37th Hawaii International Conference on System Sciences – 2004

[10] Kern A., Kuhlmann M., Kuropka R., Ruthert A.: A meta model for authorisations in application security systems and their integration into RBAC administration, ACM symposium on Access control models and technologies 2004 New York, Pages: 87 - 96

[11] Wainer J., Barthelmess P., Kumar A.: W-RBAC – A workflow security model incorporating controlled overriding of constraints, International Journal of Cooperative Information Systems 2003, pp. 455-485

[12] Joshi J. B. D., Bertino E., Latif U., Ghafoor A.: Generalized Temporal Role Based Access Contol Model (GTRBAC) Part 1 - Specification and Modeling, Cerias Tech Report 2001-47

[13] Joshi J. B. D., Bertino E., Latif U., Ghafoor A.: Generalized Temporal Role Based Access Contol Model (GTRBAC) Part 2 – Expressiveness and Design Issues, Cerias Tech Report 2003-01

[14] Sandhu R., Bhamidipati V., Munawer Q.: The ARBAC97 Model for Role-Based Administration of Roles, ACM Transactions on Information and System Security, Vol. 2, No. 1, February 1999, Pages 105–135.
[15] Oh S., Sandhu R.: A model for role administration using organization structure, Proceedings of the seventh ACM symposium on Access control models and technologies; Monterey California 2002, pp. 155 - 162
[16] Jiong Q., Chen-hua M., Jian-wei Y., Dong Jin-xiang: Research and Implementation of Role-Based RBAC Administration Model, The Fifth International Conference on Computer and Information Technology (CIT'05), 2005
[17] Kern A., Schaad A., Moffett J.:.An Administration Concept for the Enterprise Role-Based Access Control Model, ACM, SACMAT'03, June 2–3, 2003, Como, Italy., 2003, pp. 3 - 11
[18] Weber B., Reichert M., Wild W., Rinderle S.: Balancing Flexibility and Security in Adaptive Process Management Systems, Proc 13th Int´l Conf. on Cooperative Information Systems, Agia Napa, November 2005, pp. 59-76
[19] Middendorf S., Singer R., Heid J.: Java – Programmierhandbuch und Referenz für die Java-2-Plattform, Standard Edition, 3. edition, 2002, http://www.dpunkt.de/java/Programmieren_mit_Java/Sicherheit/14.html, last visited on January 28, 2008
[20] Lai C., Gong L., Koved L., Nadalin A., Schemers R.: User Authentication and Authorization in the Java Platform, Proc 15th Annual Computer Security Applications Conference, Phoenix, AZ, December 1999, pp. 285-290
[21] Morgan A. G., Kukuk T.: The Linux-PAM System Administrators' Guide, 2008, http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/Linux-PAM_SAG.html, last visited on Mai 8, 2008