Administration of an RBAC system

Fredj Dridi, Björn Muschall and Günther Pernul
Department of Information Systems
University of Regensburg
Universitätsstrasse 31, D-93053 Regensburg, Germany
Email: {fredj.dridi,bjoern.muschall,guenther.pernul}@wiwi.uni-regensburg.de

Abstract— **Recently RBAC (role-based access controls) was found to be among the most attractive solutions for providing acess control in web-based e-commerce and e-government applications. Usually, such systems involve a huge number of heterogeneous users working with the systems under different rights and obligations. In an RBAC authorization and access control system the users are assigned to roles which are derived from the organizational structure. Because of the huge amount of users and the diversity of their requirments the administration of a RBAC system becomes crucial. Our group is involved in the European funded Webocracy project in which we have designed and implemented an RBAC system based on the Core RBAC model as defined in a proposed NIST standard. Based on the functional specification of the proposed NIST standard we specified administration requirements for managing roles, users and permissions we specified. In this paper we will present an administration console, which we designed to implement this requirements.**

I. Introduction and motivation

RBAC models have matured to the point where they are now being prescribed as a generalized approach to access control. For instance, recently RBAC was found to be among the most attractive solution for providing access control in web-based e-commerce and e-government applications [1]. Such an e-government system is the Webocrat-System, which is designed and implemented as part of the Webocracy project [2], [3]. Webocracy is an ongoing research project funded by the European Union under contract IST-1999-20364. Its full title is "Web Technologies Supporting Direct Participation in Democratic Process" and is an example of innovative use of state-of-the-art web technologies in order to support direct participation of citizens in democratic processes. The goal of the Webocracy project is to use new technologies to provide citizens, businesses, and government agencies with more convenient access to government information and services, to improve the quality of the services and to provide greater opportunities to participate in democratic institutions and processes. The Webocrat system is modularly designed and composed of several modules implementing the following functions: discussion management, publishing on the Web, opinion polling, reporting, intelligent retrieval of information, and knowledge management. An integral module within the e-government system Webocrat is CSAP (Communciation, Security, Authentication and Privacy). The design and implementation of CSAP is described in [4]. The purpose of CSAP is to provide the other modules with appropriate security services in order to reduce or eliminate the potential damage that may be produced when security violations are exploited. Part of these services is the authorization and access control facility of CSAP. Currently we have provided RBAC-based authorization, however, CSAP is designed in a way that existing components can be replaced by new ones (plug-and-play architecture). In this work we are concerned with the administration of the RBAC-based access control and authorization facility. We will further describe an administration tool with graphical user interface, called CAC (CSAP Administration Console), that has been build to administrate the CSAP system. We will concentrate on the RBAC related administrative requirements for the implementation of the CAC.

The remainder of this work is structured as follows: Section 2 presents the RBAC models given by the proposed NIST standard [1] and explains the functional requirements of the Core RBAC model, which forms the basis of our implementation. Section 3 introduces CSAP, its architecture and how applications can use the provided security services. In Section 4 we will describe RBAC related administrative requirements, which are important for the implementation of the administration console. A description of the CAC follows. In the remainder of Section 4 we describe how we meet the discussed administrative requirements.

II. The NIST RBAC-Model

In [1] a standard for role-based access control is proposed, which is organized into the RBAC Reference Model and the RBAC System and Administrative Functional Specification. The reference model defines the scope of features that comprise the standard and provides a consistent vocabulary in support of the specification. The reference model is furthermore divided into several submodels as depicted in figure 1, which comprises different sets of functionality and are described as follows:

Core RBAC embodies the essential aspects of RBAC that users are assigned to roles, permissions are assigned to roles and users acquire permissions by being members of roles. RBAC comprises five basic element sets as depicted in figure 2: Users are active elements that can be human beings as well as software processes, agents, etc. Roles correspond to fields of activities of human beeings in an organisational context. These activities are bound to the corresponding permissions to carry out these activities. The permissions assigned to a role should be the minimum set required for fulfilling all necessary

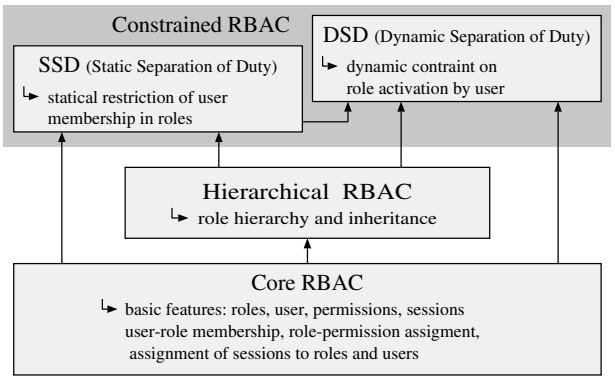

Fig. 1. Overview of different RBAC models

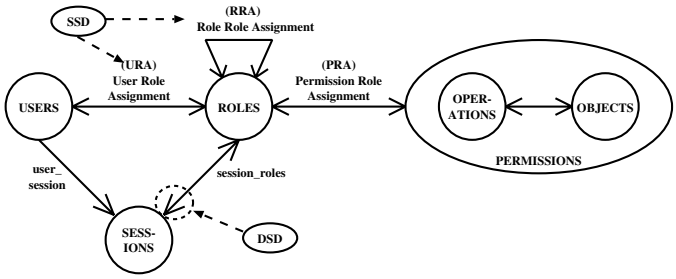

Fig. 2. The elements of the NIST RBAC reference models

activities. A permission concerns the combination of a certain operation that can be executed on a certain object. A session concerns the context in which a sequence of activities are excecuted. There must be at minimum one activated role, of which the assigned permissions can be used during the session. A user can be assigned several roles, to which the assigned permissions may not be used at the same time. The session enables to seperate role assignment from the context of execution and use of the assigned role's permissions. Objects and Operations can be arbitrary reflect system objects and methods at different levels of granularity.

Hierarchical RBAC enlarges core RBAC by a partial order between roles, which describes an inheritance relation, whereby senior roles acquire the permissions of their juniors, and junior roles acquire the user membership of their seniors. Roles can have overlapping capabilities. There may be a number of general permissions that are performed by a large number of users and consequently may be assigned to a more general role that is inherited by these users.

Constraint RBAC assumes that there are, contrary to the core RBAC, existing relations or existing exclusions between some fields of activity and allows to define separation of duty relations to enforce conflict of interest policies. Constraint RBAC let impose restrictions statically (SSD, static seperation of duty) or dynamically (DSD, dynamic separation of duty). SSD-relations can be defined to avoid a user to be assigned to mutual exclusive roles and consequently accumulate more permissions than the minimal permissions just necessary to carry out the duties corresponding to the fields of activity. DSD-relations take effect in respect to the activation of roles. For example, it can be avoided, that the role of a superuser can be activated by more than one user at the same time.

The RBAC functional specification specifies administrative operations for the creation, maintenance and review of RBAC element sets and the relations between those elements. Furthermore the RBAC functional specification defines functional requirements for system level functionality in support of session attribute management and an access control decision process. The functional requirements are organized corresponding to the submodels of the RBAC Reference Model. Concerning the Core RBAC model the requirements comprise administrative commands, system functions, review functions and more advanced review functions, which are all depicted in figure 3. As we will describe later on, the whole extent of the depicted functions is necessary to administrate RBAC. Moreover for some administrative tasks the different functions must be used in combination. For example, if you want to deassign a user from a role (*DeassignUser*) you must first find out, which users are still assigned (*AssignedUser*) to select one of the resulting set. The opposite way, to assign a user (*AssignUser*) that is not yet assigned must also be possible. Although the existing functions seem to be sufficient, we found out that some practical useful functions are missing. During the implementation of CSAP and CAC we added the new functions of significant practical value. (see Section IV-B).

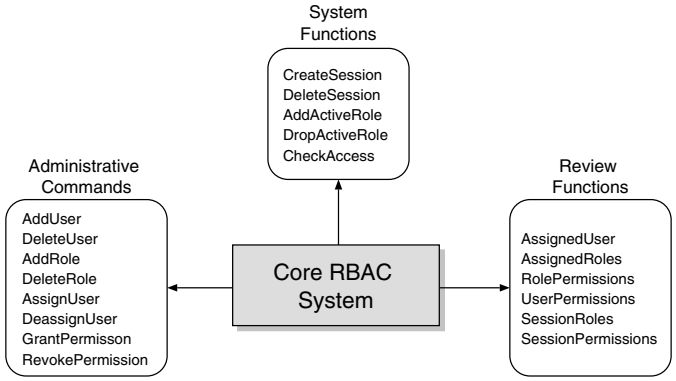

Fig. 3. Functional requirements of core RBAC

III. CSAP

The webocrat system involves a large number of different users (citizens, politicians, government employees, business members, etc.), who want to share and grant access to a huge number of security objects (i.e. in Webocracy documents) in a controlled way. Some documents may contain sensitive information and consequently must not be disclosed to every user. Basic security requirements have been determined: confidentiality, integrity, availability, authenticity of data, non-

repudiation of messages, proof of originality, proof of identification. Security services addressing the above mentioned security objectives are essential issues within the Webocrat system. The purpose of CSAP is to form the basis for trust for the whole Webocrat system offering practical and consistent security by providing security services for:

- Identification and authentication - is needed in order to get and verify the identity of the users attempting to access the system or its modules. CSAP authentication mechanism range from simple authentication by password to more sophisticated models based on signed certificates.
- Access control and authorization - is needed in order to verify what actions the users are able to perform and what information the users are able to access. The access control mechanism is implemented based on a role-based access control model.
- Auditing - is needed to log all security relevant actions in order to make users accountable for their transactions. Audit trails are used to find patterns of abnormal use which are often a sign of (attempted) compromise.
- Session management - is needed to keep track of global security information used throughout the Webocrat system and to authenticate users only once during a session.

During the design of CSAP the overall goal was to develop a general, flexible and extensible architecture in which applications can selectively and dynamically access security services (see [4]). In figure 4 the conceptual architecture in conjunction with a web-based use-scenario is described. CSAP should be able to anticipate new security requirements and allow the integration of new security services or the enhancement of the existing ones without the need of modification, recompilation, or even notification of client code unless there would be a significant change in the specification. Furthermore it was required that the way how the security information is stored has to remain configurable and storage mediums and mechanisms have to be substitutable. To meet these primary requirements the overal conceptual architecture has to provide some generic interfaces (API), facilities for medium-independent data storage, as well as corresponding configuration tools.

Within the Webocracy project CSAP will be used by several user partners (city councils, public administrations) with different requirements for the implementations of the offered services. For example, the authentication service can be implemented using passwords, using PKI or a different scheme, while the authorization service can be implemented based on RBAC or a different scheme. As a result the need raised to design a system implementing different security services that are completely substitutable, as demonstrated in figure 4. In this context we speak about "pluggable" security services.

IV. Administration of RBAC

From a technical point of view, RBACs flexibility results from the fact, that users can be easily de-assigned from

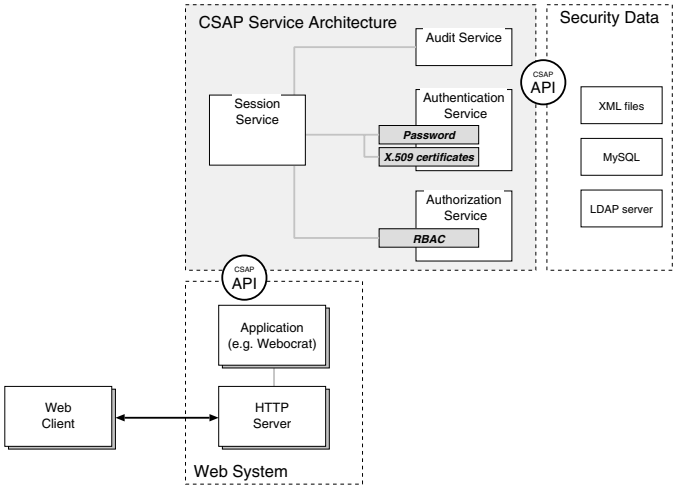

Fig. 4. CSAP service architecture

roles while roles can be granted or withdrawn permissions without interfering with the assignment of users. From an organizational point of view roles reflect and generalize various types of job functions (secretary, billing clerk, manager), which greatly simplifies security administration. For example, when a user moves to a new job function, only the user's membership in roles must be modified while in the absence of roles (in traditional discretionary access control systems) the permissions have to be individually searched and revoked.

Therefore RBAC proved to be suitable for large enterprise-wide multi-domain systems with a number of roles up to hundreds and a higher number of users and permissions up to ten thousands. This means that administration of RBAC becomes more important and that managing this amount of roles, users and permissions has to be done carefully and prudently to ensure that security policies remain to be the intended ones.

In this section we describe administrative issues that form our functional requirements for the implementation of CAC and some extension of the authorization facility of CSAP, which implements the NIST functional requirements. Later on we describe the implementation of CAC and how we meet the given functional requirements.

A. CAC's requirements

As mentioned in Section II RBAC elements are users, roles, permissions, objects, operations and sessions (we will use the term elements later on in this text). With regard to the management of these RBAC element sets the NIST standard requires only functions for creation and deletion of users and roles as part of the administrative commands, depicted in figure 3, which are likewise part of the RBAC functional specification. With regard to the management of the user role assigment and the permission role assignment (depicted in figure 2) the NIST standard only requires operations to assign and deassign permissions and users to a role. Assigning roles to permissions starting with permissions is not included.

Moreover, review functions (see figure 3) are essential for the use of administrative commands. Before deassigning a user from a role whith the administrative command *DeassignUser*, the review function *AssignedUser* must be called, but there is no review function *NotAssignedUser* which could be called before assigning a user to a role.

After several discussions with our end-user partners in the Webocracy project, we figured out some requirements to meet for the administration of a Core RBAC system like the access control component of CSAP, which the required functionality in the functional specification of the NIST standard are not sufficient for. These RBAC related requirements are represented in the following and arranged into three groups. The first group regards to the managment of the element sets while the second and third group regard to two aspects of the management of the corresponding relations.

1. Although the remaining elements (permissions, objects and operations) are given by the system for which the RBAC system is employed for, there must be some facility to define them explicitly within the RBAC system. Consequently it must be possible to create and delete all five element sets not only users and roles. The NIST proposed standard only requires to create and delete users and roles.
2. Administration of element sets does not only comprise the creation of elements but also some functionality to change (the attributes of) just existing elements. Therefore update operations for each element are required.
3. Contrary to the create function that does not require an element to be existent, the delete and update operation require to be applied to an existing element. This element is assumed to be selected of an increasing amount of defined elements first. Since RBAC elements does not only possess an unique identifier (ID, name) but some more attributes, which may not be unique, there must be some selection facility that allows selection by an arbitrary number (one or more) of these attributes in order to modify or delete selected elements later on.
4. Before creating a new element, the administrator has to specify the element's attributes in a GUI mask. It is necessary to validate user input in the sense, that user input is complete and does not lead to a conflict with a still existing element. A conflicting element must be found first.

In accordance with figure 2 there are relations to be managed between four of the element sets: users, roles, permissions and sessions. Each relation is symmetrical and involves two elements that can be of the cardinality n:m. Identifying an existing assignment (instance of that relation) means to start with a given element object (e.g. a certain role), to ask for all elements that are assigned to it (e.g. one or more users) and to select one of the resulting set of assigned elements. We will later use the terms user-view, permission-view and role-view. If we consider for example the relationship user-role assignment starting with a user, we also can say that we consider the user-role assignment from user view. There are following reasons to consider more relations from both possible views than done in the NIST functional requirements. In that sense the Core RBAC review functions are not sufficient und must be completed:

5. It must be possible to know in advance, which roles are affected if a permission is modified. Consequently it must be possible to select assigned roles from permission-view.
6. It must be possible to determine if a user is logged into the system, when his attributes must be changed or when the user may be deleted. To be logged in means, that there must be at minimum one assigned session. The concerned user must be logged off first to avoid the lost update problem.
7. It must be possible to withdraw a users membership in a role, that may just be activated and used by the affected user. To examine this case means to evaluate if a role is activated within a certain session.

Deleting an assignment between certain elements (e.g. a role and a permission) assumes the assignment to be already existent. From view of a certain role an assigned permission is part of the already assigned elements. To assign a permission to a role and so establishing a new relation instance between those elements requires the permission to be part of the not yet assigned elements (the complementary set). This means, that the review functions of the Core RBAC model must be completed with review functions, which return the complementary set of a considered element. Moreover the administration console must reflect this fact in the user interface, which is shown in figure 5. More precisely the following requirements are valuable:

8. To grant one or more permissions to a certain role, which is/are not known precisely, there must be a function that lists all not yet granted permissions and enables the administrator to perform a selection.
9. If a certain permission is given, that may be assigned to one or more roles, which the administrator does not know precisely, there must be a function that lists all not yet assigned roles which are candidates for the assignment of this permission.
10. From view of a certain user, it is comfortable to give a list of potential roles for a membership. From those candidates the administrator can select one or more roles and assign them to the user.

B. Design and implementation issues

In the previous section we described the RBAC related requirements, which form CAC's functional requirements. There are some more requirements to meet, which primarly concern to the main design decision to be made for CAC's software architecture:

- **a.** In its actual state of development the access control component of CSAP implements only the core RBAC model of the proposed NIST standard. The extension to other RBAC model such as the hierarchical or constraint

RBAC is planned and must be anticipated in the design of the administration console. Therefore an extensible and flexible design must be able to realize additional views of the user interfaces that reflect the new functionality and can administer hierarchical relations and constraints.

b. Similar to the approaches mentioned in section V the functionality of the administration console must be protected by some authentication mechanism and access control functionality. Access to the console is only given to administrative users which have been granted the adequate authorizations to perform some administrative functions. There are several administrative roles with different scopes of administrative rights.

c. A simply understandable, clear and consistently subdivided user user interface is required. Consistent support for navigation through the console's functionality is needed in the same way as error messages and status information must be visible consistently.

d. Since CSAP is used by several web-based modules in the Webocrat system, it was quite obvious to require a web-based design for the administration console and so integrate it with the other applications. This offers the advantage of providing remote access to several administrative users. Of course such security administration functionality is highly sensitive, which is why our CAC modul has to be secured by means of cryptographic techniques like SSL (Secure Socket Layer).

In figure 4 the integration of CSAP into web-based applications is demonstrated. In accordance with the requirement *d* the administration console is implemented as a web-based application based on technologies like JSP and servlets [5], [6]. CAC is a client of CSAP as depicted in figure 4.

To meet the requirement *b* we decided to use the functionality of CSAP to realize access control for the administration console. Consequently - on one hand - the CAC represents a client depending on CSAP functionality for authentication and access control as the other applications using CSAP do. This case is called "administrating RBAC by means of RBAC". On the other hand, CAC has a special purpose that the CSAP system depends on, which is to administer the persistent data of CSAP. Consequently CAC needs some more administrative functionality to be provided by CSAP, that goes beyond the functionality needed by other clients. Since the persistent data of CSAP are hold transparent to clients and the currently employed implementation of the data layer depends on the current configuration, the CAC has to avoid operating directly on the data layer.

Since CSAP is implemented in java, we used java related web-techniques like servlets, java beans, tag-handler and JSP. Requiring a high degree of flexibility and the ability to anticipate enlargements to the underlaying CSAP module, the architecture of CAC was designed with a special variant of the architectural design pattern MVC (model view controller) called Model 2. Model 2 allows an application to be scaled up using EJB in conjunction with servlets and JSP and is part of a larger catalog of integrating design patterns described in the Blue Prints [7]. Some more of those patterns described in the catalogue are used by the console.

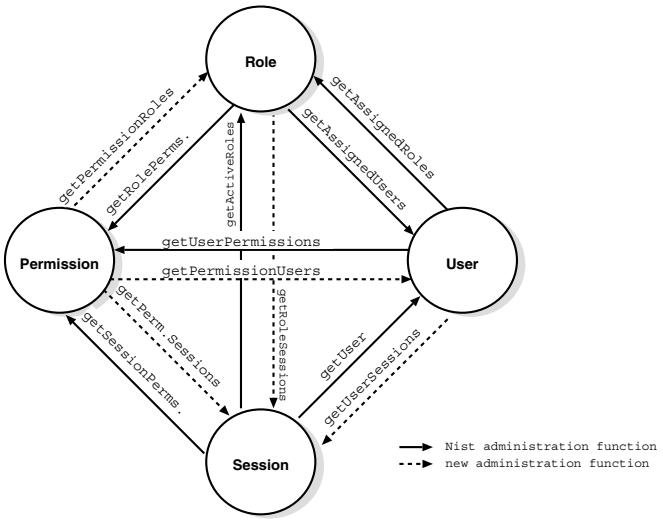

Fig. 6. Administration functions

Figure 5 shows the structure of CAC. On the left side of CAC a navigation menu is shown. The result of every menu selection is displayed on the right side of CAC. In this example permissions can be assigned or deassigned to/from a role which is first selected. In this case the relation called permission-role assignment (compare figure 2) is considered from role-view. After selecting a role the list of the already assigned permissions and the list of the not assigned permissions are presented. It is up to the administrator to select a certain permission from one list and assign it to the opposite one. The management of the other relations (e.g. user-role assignment) is done in a similar way. All relevant relations can be considered from both views respectively, e.g. the relation user-role assignment from role or user view.

In section IV-A we described some requirements regarding to the management of the element relations. To meet this requirements (especially the reqiurements 5,6,7) we consequently decided to implement all missing functions, to support both views of each relation. For example, for the relation role-permission assignment we implemented *getRolePermissions* as well as *getPermissionRoles*. In figure 6, the functions marked with a dashed line are not defined in the NIST RBAC models and additionally implemented by CSAP.

To meet the requirements 8,9,10, some operations are necessary, which returns the complementary sets to the respectively assigned elements. For example, considering the relation role-permission assignment from role-view, the assigned permissions are given by the function *getRolePermissions*, which is defined in the NIST RBAC model. The complementary set containing the not assigned permissions is given by an additionally implemented function, called *notAssignedPermissions*.

V. Related Work

There are some existing RBAC Systems, that come with an administrative user interface, which we examined and

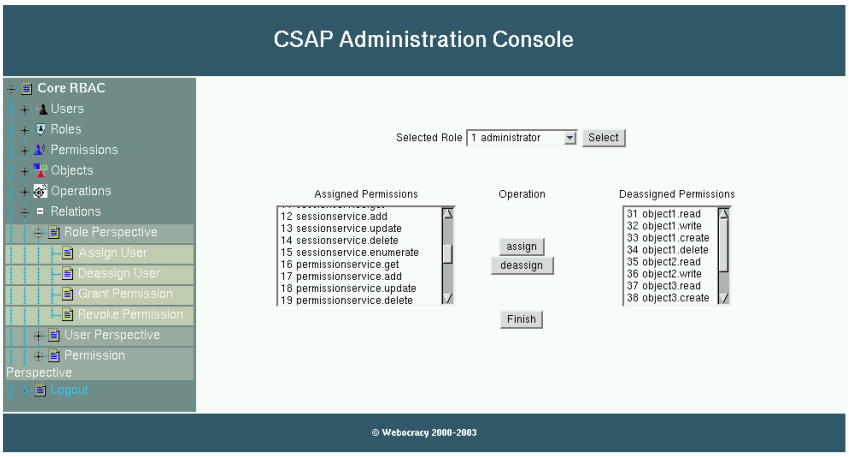

Fig. 5. Screenshot: permission role assignment from role-view

compared to get some incentives for our design of the user interface and to identify some more requirements for our implementation. In RBAC/Web [8] Sandhu and al. implemented a web-based reference implementation that comprises CGI scripts connected to some binary kernel. Consequently RBAC/Web provides a user interface based on HTML pages that only requires a standard browser to be viewed. Contrary to the web-based approach the solaris administration console [9] concerns a user interface that is based on swing, which is a graphical library implemented in Java and shipped as part of the JDK.

Sandhu et al. [10] concern with the topic of administrating RBAC. This work describes how RBAC can be used for managing RBAC itself and is mainly motivated by the goal to decentralize the details of RBAC adminstration without loosing central control over broad policy. It is proposed to use a second level of roles for administrative purposes and so enable to spread administrative responsibility over more administrative roles. This work is build upon the cognitions of the RBAC models given in the proposed NIST standard.

VI. Conclusion

We have presented CSAP, an autonomous software module offering programming interfaces to core security services such as authentication, access control, auditing and security management. It is currently developed as part of the ongoing EU-funded Webocracy project. We further concentrated on the access control facility of CSAP that is an implementation of the Core RBAC model described in [1]. Regarding to the functional specification of the Core RBAC model, we formulated some additional functional requirements that had to be met by the implementation of an administration console. Additional non-functional requirements for the software architecture of CAC had been presented as well. To meet the functional requirements, the required functionality of Core RBAC had to be completed and implemented by the CSAP and CAC.

In future works, the access control facility of CSAP will be extended conforming to the remaining RBAC models as well as CSAP will be extended to include alternative plug-in security services. In addition, the configuration functionalities of CSAP for managing several applications with different storage requirements will be improved. The improvement of the CAC is planned to go along with the extension of CSAP. Therefore we designed the CAC in a flexible, extensible way.

Acknowledgment

We would like to thank our project partners for helpful comments and stimulating discussions. This work is done within the Webocracy Project which is supported by European Commission DG INFSO under the IST programme, contract No. IST-1999-20364. The content of this publication is the sole responsibility of the authors, and in no way represents the view of the European Commission or its services.

References

[1] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and Systems Security*, vol. 4, no. 3, pp. 224–274, August 2001.

[2] F. Dridi, G. Pernul, and T. Sabol, "The Webocracy project: Overview and security aspects," in *Professionelles Wissensmanagement: Erfahrungen und Visionen*, S. et al., Ed. Shaker Verlag, Aachen, 2001, pp. 401–408.

[3] J. Paralic, T. Sabol, and M. Mach, "A system to support e-democracy," in *Proc. of the 1st eGovernment Conference within DEXA2002*, Aix-en-Provence, France, September 2002, pp. 288–291.

[4] F. Dridi, M. Fischer, and G. Pernul, "CSAP – an adaptable security module for the e-government system Webocrat," in *Proc. of the 18th IFIP International Information Security Conference (SEC 2003)*, Athens, Greece, 26-28 May 2003, pp. 301–312.

[5] Sun Microsystems, "JavaServer Pages 2.0 Specification, Proposed Final Draft ," April 2003. [Online]. Available: http://jcp.org/aboutJava/communityprocess/first/ jsr152/index3.html

[6] Sun Microsystems , "Java Servlets," Dezember 2001. [Online]. Available: http://java.sun.com/products/servlet/

[7] J. Singh, Stearns, *Designing Enterprise Applications with the J2EE*. Addison-Wesley, 2002, iSBN 0-201-78790-3.

[8] D. Ferraiolo, J. Barkley, and D. Kuhn, "A role-based access control model and reference implementation within a corporate intranet," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 34–64, February 1999.

[9] "Rbac in the sun solaris 7 operating environment," Mai 2002.

[10] R. S. Sandhu, V. Bhamidipati, E. Coyne, S. Ganta, , and C. Youman, "The ARBAC97 model for role-based administration of roles: preliminary description and outline," in *Proceedings of Second ACM Workshop on Role-Based Access Control*, Fairfax, Virginia, November 6-7 1997.