# Kroki Administration Subsystem Based on RBAC Standard and Aspects

Sebastijan Kaplar, Milorad Filipović, Gordana Milosavljević, Goran Sladić

Faculty of Technical Sciences, University of Novi Sad, Serbia
{kaplar, mfili, grist, sladicg}@uns.ac.rs

***Abstract*— This paper presents administration subsystem that was developed to enable dynamic customization of enterprise applications specified by our Kroki tool. Kroki is a tool for participative development of enterprise applications based on executable mockups. The administration subsystem is based on standard RBAC model for access control. It enables users to perform previously authorized tasks by dynamically adjusting their actions. Available actions are determined based on operations, relationships and constraints, according to RBAC. Every user role can access only specific parts of the application that its role is entitled to, with the use of menu adjusted especially for its needs. Dynamic adjustments of available actions are implemented by our runtime engines based on aspect-oriented approach.**

## I. INTRODUCTION

Enterprise applications usually have a large number of users with different roles and responsibilities. For example, a worker in a warehouse can access only information about his warehouse. His superior can access information about all warehouses in the application. Financial director has the rights to access all available subsystems.

Enterprise applications should be dynamically adjusted to support specific needs of every user. In order to achieve this, we need an administration subsystem that allows specification of a working environment for each user (menu structure, user rights, etc.). Also, we need the architecture of the enterprise applications that can adapt to the mentioned specification on the fly.

This paper presents the administration subsystem that was being developed as a part of our Kroki tool [5, 7, 8] and generic engines that allow run-time enterprise application adaptation. Kroki (*croquis* – sketch) is an open-source tool for participatory development of enterprise applications based on executable mockups (see Section 3 for details).

The administration subsystem is based on RBAC (Role Based Access Control) standard for access control. RBAC introduces user roles as an additional layer of indirection between users and permissions. User roles can be created, modified, or deleted based on the enterprise system requirements, without the need to individually manage privileges for every user. RBAC-based systems enable users to perform previously authorized tasks, by dynamically adjusting their actions. Actions are determined based on operations, relationships and constraints [1].

The paper is organized as follows. Section 2 reviews the related work. Section 3 gives a short overview of our Kroki tool and describes the administration subsystem implementation. Section 4 presents an example of dynamic adjustments. Section 5 concludes the paper.

## II. RELATED WORK

The paper [1] is one of the early papers that present a family of RBAC models. These models are provided as a common frame of reference for other research and development in the area, and they are used as a starting point for our work.

The paper [2] presents modeling and metamodeling of access control policies. The described process spans three meta-levels. At level M2, the policy metamodel is defined. Using the policy metamodel, different policy models can be applied at level M1, such as RBAC. PolicyDSL is used at level M0 for specifying actual access control policies in a particular system, and policy model is used for parameterizing the syntax of PolicyDSL.

In [3] the work is focused only on specifying the static structure of RBAC, and utilizes standardized modeling language (UML) and also integrates the policy specification activity with UML design modeling activities. Also, the task of capturing RBAC policies in reusable patterns is described.

In [4] SecureUML is introduced. SecureUML extends RBAC in order to add constraints on system states that are associated with a UML model. Rules are allowed to be restricted with OCL (Object Constraint Language) conditions. Also, action hierarchies allow forming of higher-level actions.

## III. THE KROKI ADMINISTRATION SUBSYSTEM

The administration subsystem is developed as a part of our Kroki tool (Figure 1). Kroki is a tool designed for development of business applications based on executable mockups. A mockup is a sketch of an application UI (User Interface).

Kroki uses mockups as a basis for automatic execution and code generation for the enterprise applications. An advantage of mockup tools is in their ease of use, which allows end users to participate actively in the development process using simple and intuitive notation [9]. Besides mockups, Kroki also supports "classical" way of application modeling, using its lightweight UML editor [1].

Mockups execution is performed by two aspect-oriented (AOP) engines for the web and desktop applications. A basis for execution is application repository that contains configuration files for the current Kroki project. Configuration files are generated from the current model using Kroki generators (Figure 2). Although the engines could take this data directly from the

Kroki model, we choose XML files as an intermediate step in order to provide independent functioning of the specified applications, when deployed.

The administration subsystem is getting information about specified application elements (forms, panels, reports etc.) from the application repository. These elements are observed as resources in access control model.

Configuration files created by the administration subsystem are also delivered to the application repository. They are loaded during the engines start up and used as a source for dynamic adjustment of the application based on user rights.

Dynamic access to the user rights and their activation in appropriate moments is achieved with the use of AOP. Kroki engines use AOP techniques in order to enable easier integration of cross-cutting concerns imposed by its tools. Also, AOP enables easier integration of the engines with hand-written code which is necessary for implementation of complex business transactions and reports.

A tool for creating personalized menus is also a part of the administration subsystem. The tool helps in the process of creating custom menus, defined for every user role. Custom menu allows personalization of users working environment.

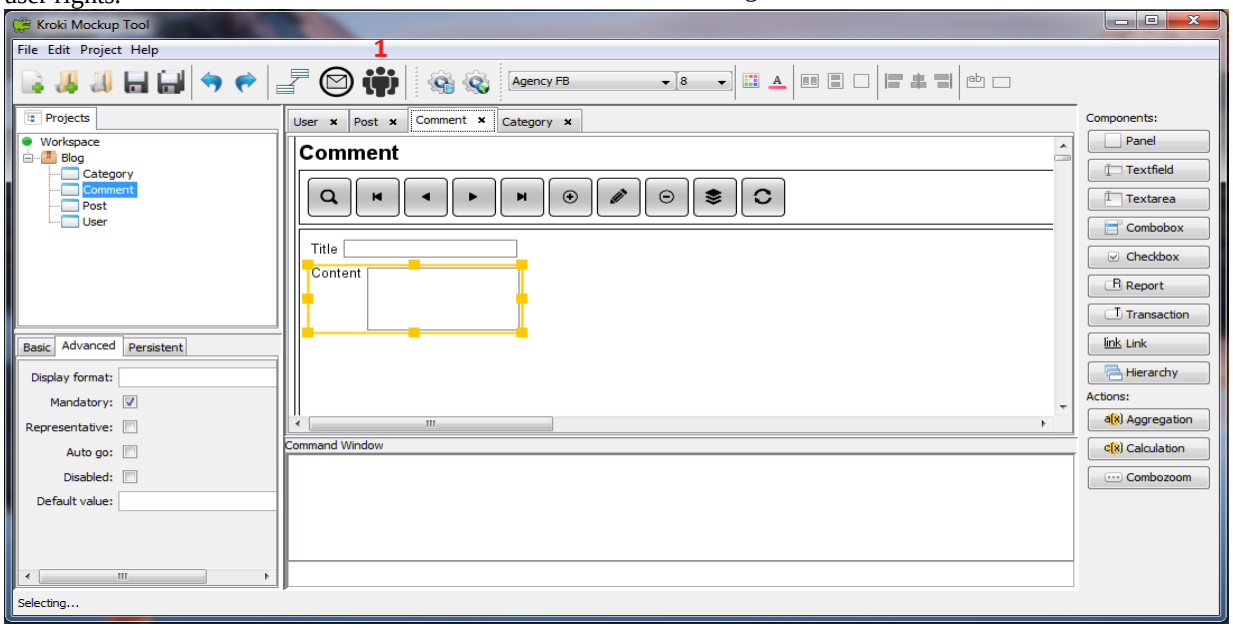



Figure 1. Kroki mockup tool. Icon for activation of the administration subsystem is marked with 1.

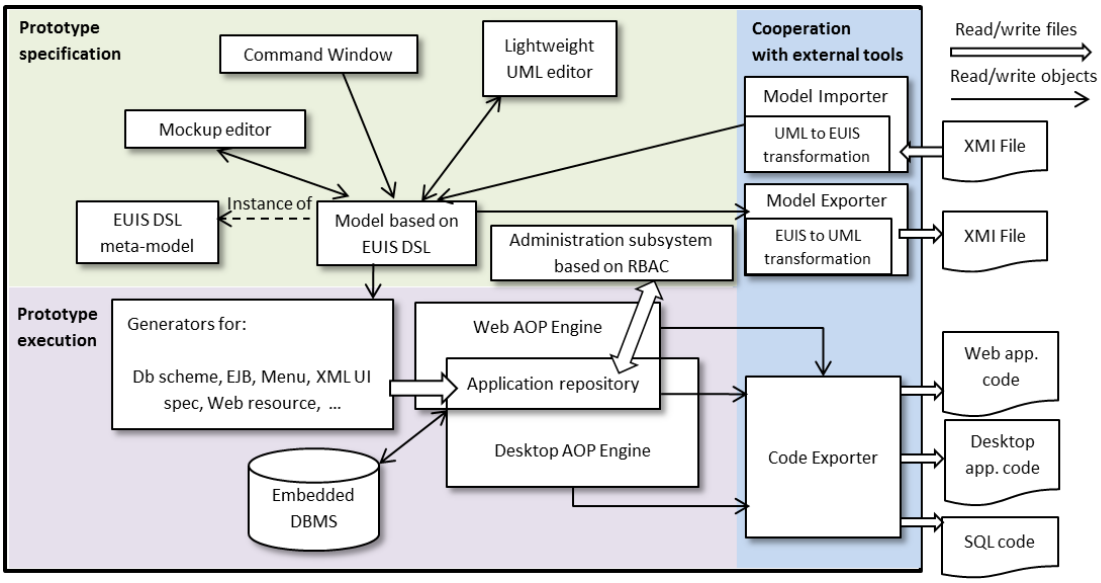



Figure 2. Kroki architecture

## A. *RBAC*

The administration subsystem is based on RBAC, which is one of the most common access control models. RBAC is largely spread in business systems with a large number of users. The reason behind it is in the introduction of user roles. User role is an additional layer of indirection between users and permissions allowing the grouping of users and privileges in logical units at the higher level of abstraction. This enables simplified

specification of the user rights. RBAC model [3] is shown in Figure 3.

Basic entities defined with RBAC model shown in Figure 3 are sets of users, roles, objects, operations, and permissions. A role is a job function performed by the user. Object is an entity that contains or receives information. Permission is an approval to perform an operation (for example add, modify, remove) on the object. Permission is defined abstractly and implemented as a pair of permissions (objects, operations).

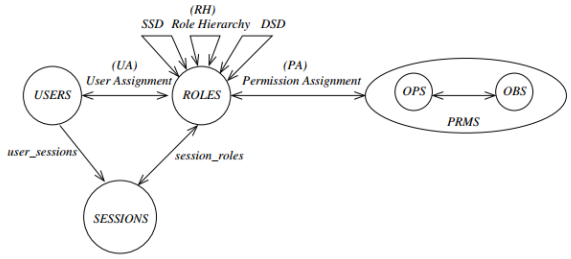



Figure 3. RBAC Model

The permission for execution of a certain operation is defined as a basic concept in RBAC. Associating permissions with roles simplifies their management. Users do not have direct permissions; they are obtained through roles.

Figure 4 shows the administration subsystem's UML model. In Figure 4, classes (entities) and their relationships in the administration subsystem are shown. One of the features of the model are resources, represented by the class *Resource*. Resources represent application forms sketched using Kroki's mockup editor. Storing data into resources is performed automatically through forms previously sketched in Kroki. Figure 4 shows that for each resource multiple permissions can be defined (*Permission* class) while each permission can have only one resource that results in one-to-many relationship.

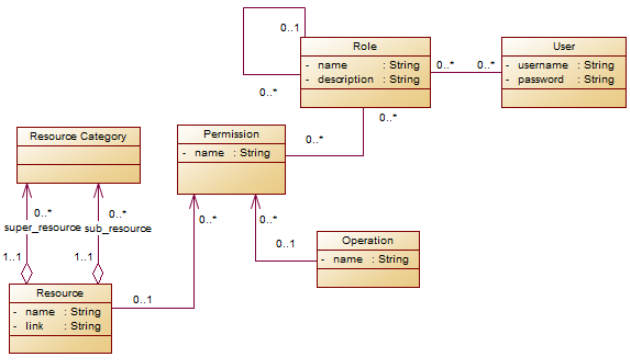


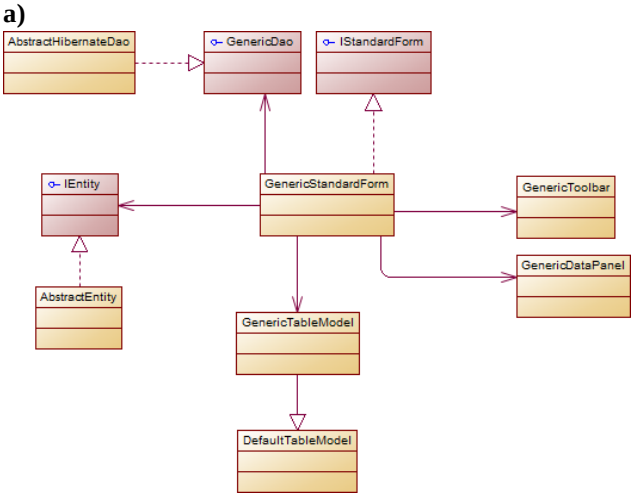

Figure 4. Administration subsystem model

Also, there is a categorization of the resources. Permission has one resource and one allowed operation (class *Operation*) on that particular resource, for example: add, modify, delete. For one resource, many permissions with different operations may exist and when a new operation is introduced it can add new permissions with the newly introduced operation on the existing resource. Roles (class *Role*) associate permissions with users. Figure 4 shows that one role can have more permissions, and one permission can have multiple roles that result in many-to-many relationship. Identical is the relationship between roles and users (class *User*).

## B. *Implementation*

The administration subsystem is implemented as a three-tiered desktop application. A presentation part is implemented in Java using the Swing GUI library. The middle tier, responsible for business logic, uses Hibernate library for persistence.

**a)**




**b)**

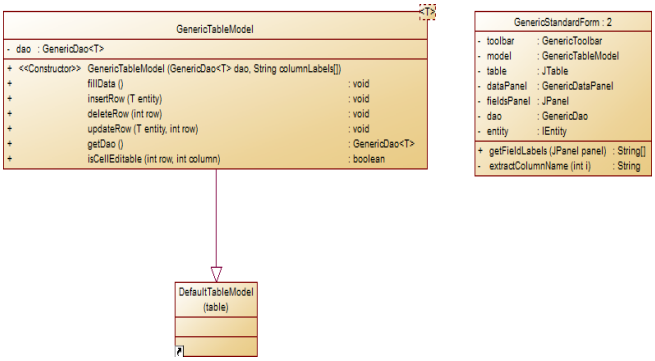



Figure 5. a) A sketch of a simple framework used for implementation of the administration subsystem application b) Details for *GenericStandardForm* and *GenericTableModel* classes

In order to support faster and easier development of the administration subsystem UI, we have developed a simple framework presented in Figure 5. Classes *GenericStandardForm* and *GenericTableModel* are used for data presentation and manipulation of all persistent classes in the middle-tier. Examples of administration subsystems forms developed with this framework are presented in Figure 6, 7, and 8. The framework code is available as a part of the Kroki administration subsystem at [11]

## C. *Integration with Kroki*

After an enterprise application is sketched using the mockup editor and/or the lightweight UML editor, it can be executed using Kroki's desktop or web engine. If user rights and customized menus are not specified, the application has a default menu that provides activation of all developed application forms. This is suitable for the development phase and requirements elicitation based on

prototypes, but before deployment, the application must be customized to support every user role in the enterprise.

After launching, the administration subsystem is supplied with an XML file that has a list of developed resources (forms, reports, etc.) provided by the Kroki tool in the application repository (Figure 3). Kroki is "aware" of the administration subsystem's existence, but the reverse does not apply, in order to achieve a higher level of independence of the administration subsystem. Once the enterprise application is deployed, administrator should manage users and user rights using only the administration subsystem, with no need to access other Kroki's tools.

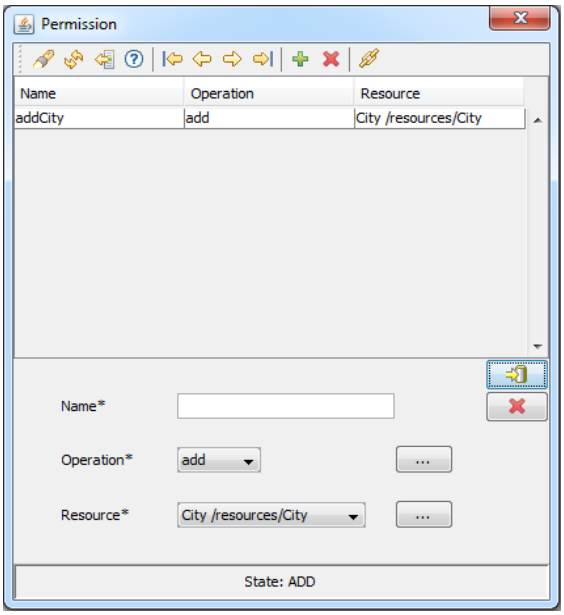



Figure 6. Permissions form

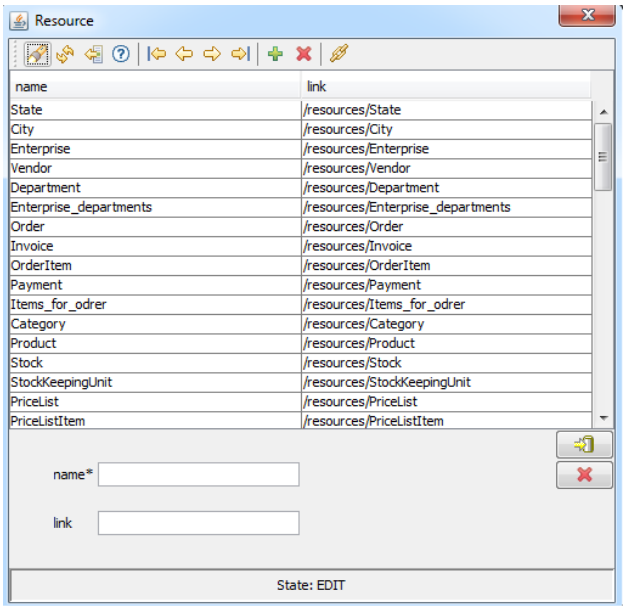



Figure 7. Resources form

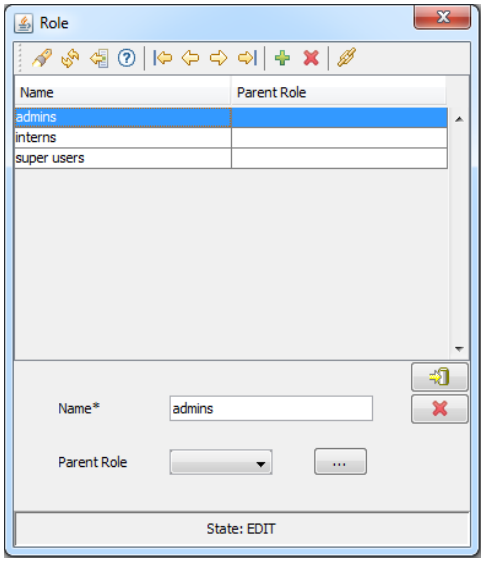



Figure 8. Roles form

The administration subsystem is storing user rights information and menu specifications in its own database. This information is provided to web and desktop engines in order to perform dynamic adjustments during run-time.

### *D. The Web and Desktop Engines*

The web and desktop engines are performing dynamic adjustments using aspects. Kroki engines use aspect-oriented programming techniques to enable easier integration of cross-cutting concerns imposed by its tools.

The web engine is developed using restlets, so all of the web classes extend RestletResource class and are located in the resources package. Every resource class has `prepareContent` method that is invoked when a client request is sent to a particular resource and can be used to attach aspect functionalities. Restlet resources use map called `dataModel` to pass arbitrary data to HTML templates, so once attached to `prepareContent`, aspect can get access to the resource object and modify its `dataModel`. `DataModel` is wrapped into HTML elements using Freemarker templates (Figure 9).

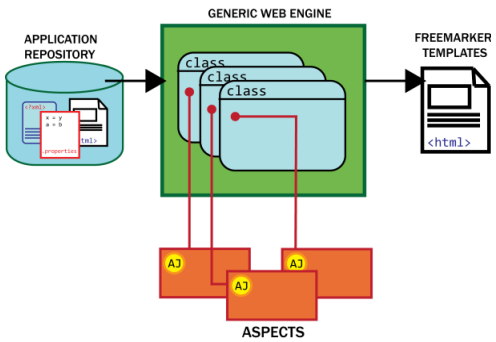



Figure 9. Web engine architecture

Listing 1 shows an aspect that modifies the main menu based on customization specified by the administration subsystem. It is activated after user has logged in and before menu is created in the application.

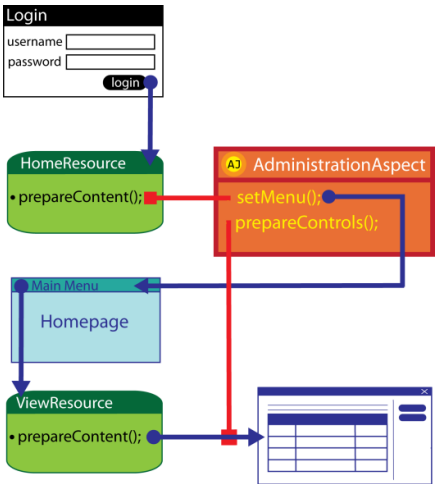



Figure 10 Aspects can change the content before pages are rendered

Since the generic web engine is designed as a single-page AJAX web application, once the user is logged in, the interaction takes place on the home page and restlet resource in charge of this page. So, in order to modify the main menu creation process, we need to attach our aspect to `prepareContent` method of `HomeResource` class. Freemarker template looks up main menu list by the name `main_menu`, so it will be the name by which we will put our modified menu into `dataModel` (Figure 10). Listing 1 represent basic steps described above.

```
public aspect MainMenuAspect {
//Create the pointcut that intercepts PrepareContent
//method in Home resource and obtain home resource object
public pointcut setMenu(HomeResource homeResource) :
        call(public void HomeResource.prepareContent()) &&
        this(homeResource);


        after (HomeResource homeResource):
                setMenu(homeResource) {
        User user = SessionAspect.getCurrentUser();
        //Obtain main menu list from AppCache
        ArrayList<AdaptMenu> menus =
                AppCache.getInstance().getMenuList();

        List<UserRoles> roles = …;
        //Obtain user roles through query
        if (roles.size() == 0) {
                //Put default main menu to data model
                homeResource.addToDataModel("menu", menus);
         else {
                //Retrieve and put modified
                //main menu to data model
                AdaptMenu modified_menu = …;
                homeResource.addToDataModel("menu",
                modified_menu);
                .
                .
                .
```

Listing 1. Aspect for menu loading

Similar activities are performed for the application forms in order to dynamically adjust their toolbar according to the specified user rights. More details about Kroki engines can be found in [10].

### *E. Menu Specification*

The administration subsystem enables specification of customized menu for every user role. Custom menu allows personalization of users working environment.

Menus in administration subsystem are based on the composite design pattern (Figure 11).

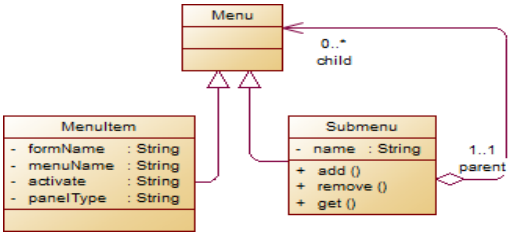



Figure 11. Menu structure within administration subsystem

The menus are stored in an XML file and deployed to the application repository. The tool for specification of menus is shown in Figure 12.

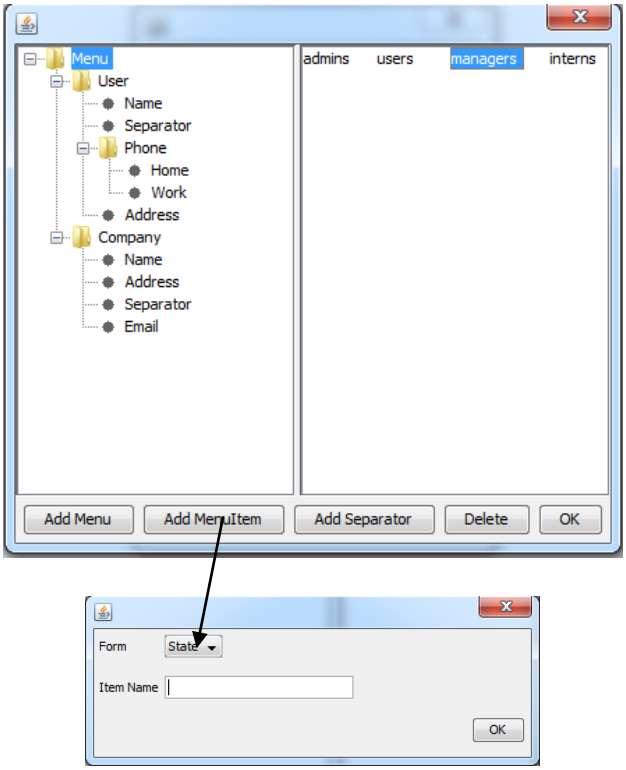



Figure 12. The tool for menus specification

## IV. AN EXAMPLE

This section shows an example of the run-time adaptation of a business application according to the user roles.

Figure 13 and Figure 14 depict the same form in the web application. The depicted form is used by two groups of users. The first group of users is allowed to add, modify and remove data (see Figure 13), while the second group of users is allowed only to view data in that particular form (see Figure 14).

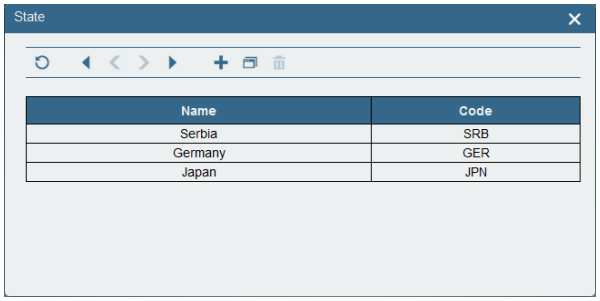



Figure 13. User form with add/modify/delete permissions

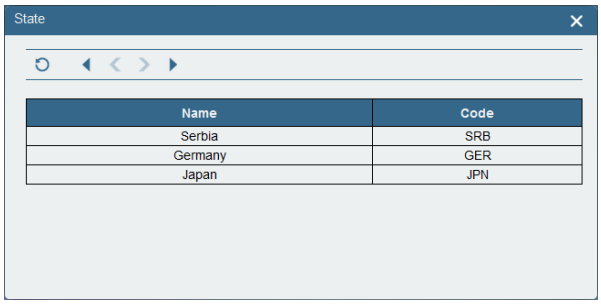



Figure 14. User form with view permission

Similarly Figures 15 and 16 show the same menu adapted for two different users. Available menu items are determined by user roles. In the case presented in Figure 15. the user is entitled to all submenus in the given menu, while in the case presented in Figure 16 the user is allowed to access only a subset of menu items.


Figure 15. Menu with all the submenus shown

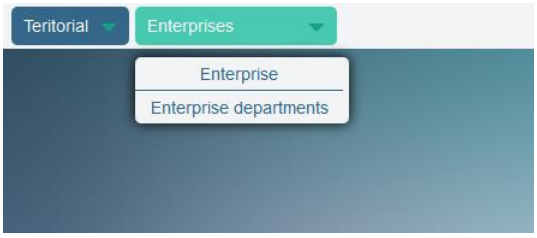



Figure 16. Menu with certain submenu restrictions

## V. CONCLUSION

The paper presented the Kroki administration subsystem and generic AOP engines that enable enterprise application execution. The administration subsystem is based on RBAC standard for access control. RBAC-based systems enable users to perform previously authorized tasks, by dynamically adjusting the availability of actions.

Dynamic application adjustment is based on the application repository that contains configuration files and generic AOP engines. Configuration files are generated from the current Kroki model and the administration subsystem and used as a specification for application dynamic behavior.

Dynamic access to the user rights and their activation in appropriate moments is achieved with the use of AOP. AOP techniques enabled easier integration of cross-cutting concerns imposed by different Kroki tools and easier integration of the engines with hand-written code. Also, AOP allowed clear separation of duty and insight into expandable and easily readable code. Thanks to that, integration of the administration subsystem with Kroki performed seamlessly without the need to change the original tool.

## REFERENCES

[1] R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. *Role-Based Access Control Models, IEEE Computer (IEEE Press) Vol. 29, Issue 2, pp. 38 - 47.*

[2] B. Trninić, G. Sladić, G. Milosavljević, B. Milosavljević, Z. Konjović, „*PolicyDSL: Towards Generic Access Control Management Based on a Policy Metamodel*“, SoMeT 2013, Budapest, Hungary.

[3] D. Kim, I. Ray, R. France, N. Li. *Modeling Role-Based Access Control Using Parameterized UML Model, FASE 2004. LNCS, vol. 2984, pp. 180-193*

[4] T. Lodderstedt, D. Basin, J. Doser, “SecureUML: A UML-Based Modeling Language for Model-Driven Security,” Proceedings of the 5th International Conference on The Unified Modeling Language, Dresden, Germany, September 30. - October 4. pp. 426-441, 2002.

[5] G. Milosavljević, M. Filipović, V. Marsenić, I. Dejanović. *Kroki: Interactive Development Of Business Application Based on Mockups*, Software Methodologies, Tools and Techniques 2013, Budapest, Hungary, pp. 235-242

[6] M. Filipović, *Adaptive Architecture Of Web Application Based on Aspects*, Master thesis, University of Novi Sad, 2011

[7] Kroki, www.kroki-mde.net

[8] Kroki demo, http://youtu.be/r2eQrl11bzA

[9] J. M. Rivero, J. Grigera, G. Rossi, E. Robles Luna, N. Koch, “Improving Agility in Model-Driven Web Engineering”, CAiSE Forum 2011, pp.163-170, 2011

[10] M. Filipović, S. Kaplar, R. Vaderna, Ž. Ivković, G. Milosavljević, I. Dejanović, Aspect-Oriented Engines for Kroki Models Execution, submitted to ICIST 2015, Kopaonik, Serbia

[11] Kroki Administration Subsystem Source, https://github.com/KROKIteam/KROKI-mockup-tool/tree/master/Kroki-Administration