JPAAM - pluggable authentication and authorization framework

Marcel Haerry
University of Berne
Software Composition Group

October 2, 2006

Abstract

Many (web) applications share content between several users and different views. To manage this content often a CMS (Content Management System) is used with different levels of access and offer possibilities to edit and change the content. Only a few systems have a security system, which can adapt to changing requirements with another type of *security model*. Therefore the way access is permitted or denied is fixed in the architecture and the evolution progress of the application. In detail this means that the part of authorization and authentication is often hard-wired into the application and bigger changes to the structure in the application are required to implement for example another policy. Typically the actual implementation of the security system fits the current wishes of the users or developers and is a fixed part of the application and therefore not very easy to exchange nor to adapt a new policy. The proposed pluggable authentication and authorization framework (called JPAAM) offers a solution to this problem and allows users to select their *security model* for their needs and gives developers the possibility to develop an application aside the aspect of authorization and authentication. JPAAM provides highly configurable interfaces with which a clear separation of the security system from the application is possible.

Contents

1 Introduction

1.1 Security system

In the common use of language a security system is a burglar alarm, that is a system that warns if someone breaks into your house, building etc. In computer science a security system can be many different things. Often a security system is a system that prevents someone from breaking into a network or a system or protects a system from attacks of malicious users. In the context of application design a security system has several aspects: authorization, authentication, data security, etc. Data security for example can require that the application ensure that all sensitive data is transfered encrypted between the user and the system or that it is even only stored encrypted.[1] In the context of this work the term security system focuses on authorization and authentication in applications. This basically means that if we talk about a security system the part of the application which does the authentication and authorization is meant.

Authentication

Authentication is the process where a user or another system authenticates itself to be the one it claims to be. Authentication can be done over several parameters. The most common and well-known authentication is with a username and a password. But you can imagine also other authentication systems: like only a password or even more parameters: a username, a pin, a password, a time window and the weather outside. All these parameters could decide if a request is authenticated to be a valid request from a proven source. However, it does not say anything about if a request is authorized for the given action.[2]

Authorization

Within a (multiuser) application authorization is done before an action is executed and a (sensitive) operation takes place. For this (sensitive) operation a decision is taken wether it is allowed to do the given action (*e.g.*, edit a given content) or not. The part which is responsible for the authorization can interrupt the current action and display an error message, it can simply say no to a given authorization request or it can even invoke additional actions, like a login process.[3]

1.2 A pluggable security system

In many applications the security system is part of the application. It is directly included into the application and its design represents often the architecture of the application. So the security system is included as a fix part in the application and can not be exchanged nor can its behavior be changed in a bigger manner without changing the whole setup. The common way a security system is part of the application is illustrated in Figure 1.

With such a high coupling of the security system inside the application it is hard to reuse it in another application and often in a new application a similar

[1] Some more thoughts can be found in: Thuraisingham (1989)
[2] See as well Tanenbaum (2003) P. 847 or Vogel (2004) P. 87
[3] See as well Tanenbaum (2003) P. 847 or Vogel (2004) P. 87

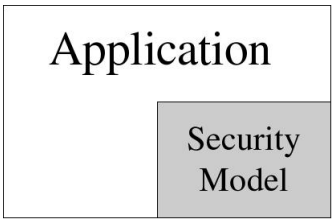

Figure 1: *security model* part of the application

system has to be rewritten from scratch.
Therefore, we propose a pluggable security system, a framework which is situated *outside* the application and can be reused. The separation of the security system from the application offers a better solution to the *common* way of endorsing a security system with the evolution of the application. As already mentioned a progress with a very high coupling of different and in fact independent parts of the application leads to the known problem that they are more and more dependent on each other. This results in a system architecture, in which a new way how something could work is harder to integrate. Therefore it is possible that in the whole application code pieces have to be exchanged, as there are everywhere points which are connected to the security system.
For a developer it can be as well very inflexible and counterproductive to integrate a new feature under huge circumstances or to test it and being forced to do every time a complicated authentication process. With a clear separation and a well defined interface it is even possible to *out source* the developments of the *security model* to someone else or even receive possible models from other developers.
The framework should not only be reusable in other applications, it should as well give the possibility to change the way access is granted. So it has to be factored out of the application and has to provide the capability to be fully configurable, modular in its design and reusable in different situation. A framework or pluggable security system like that would change the integration of the system as illustrated in Figure 2.
There will be only low coupling between the application itself and the model(s) which decide if permission is granted or not. As you can see in Figure 2, there is as well the possibility for several *security models*.
The main goal for such a framework is that nearly everything is abstracted from the application and its singularities. This results in one main method with which you simply ask for permission for a given request and the result is simply granted or denied. This request is then delegated to the selected *security model* and within this model the decision is taken whether the request is granted or not, the requesting side has to be first authenticated and so on. This leads to a simple and small, but well-defined API to get permission for a request.
All the information about the current status of the application, as well of any information of the current action has to be encapsulated in a very abstracted and general way into the request. However, a model can still have not enough information to finish its decision process, therefore the *security model* must have the possibility to ask the application for the specific information. For example

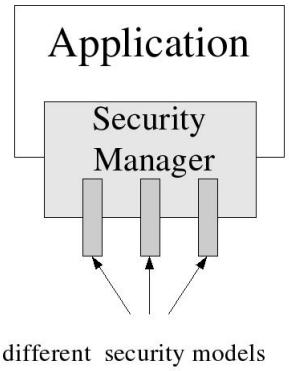

Figure 2: *security models* outside the application

if the framework is asked to get a decision for a request to edit an article and the security model needs additional information about the article (which isn't yet stored in the security session) to decide about the request, then it can simply ask the application for the specific information, which is used. As well the *security model* must have the possibility to invoke additional methods on the application to do something (*i.e.*, a login process).

1.3 Security models

Security models are the part of the framework where concretely the decision about a given request is taken.[4] They are plugged into the *security manager* (JPAAMSecManager) and requests are being delegated to the selected *security model*, which decides about the request. They are independent from the framework in the sense that it is not important for the framework, which or how many *security models* are plugged in. They are simply selected by the framework and the request of the application is passed on to the model for decision.
A *security model* is as much as possible abstracted from the application domain as it should not consider how the application works, it should simply decide about a given request. Therefore it should be possible to reuse *security models* in other applications. *More about that in the technical part in section 2.3*
Because *security models* are exchangeable a developer has the possibility to switch during the development process to a very restrictive (*i.e.*, the AllDeniedModel) or a very open (*i.e.*, the AllPermittedModel) model, which allows him to have a better development process or he simply can check very fast how things are handled if they or even everything is denied.

[4]Some thoughts in general: Jonscher & Dittrich (1993) or `http://poetshome.com/development/?p=2`

1.4 The JPAAM framework

The name JPAAM is derived from PAM (Pluggable Authentication Module) which is a well known Unix Module for applications to do authentication. As the proposed framework includes authorization as well and is written in Java the framework is called JPAAM.

2 JPAAM - technical discussion

The framework is built with several modular parts with different responsibilities. The most used and also most important parts are presented in the next sections.
The goal of the framework is to have one simple interface, which defines a small API, to be accessed by the application. The application itself should just hold an instance of the framework, which it can access from whereever it need to have a decision about a request. Therefore the only interface it has to call in the framework is the JPAAMSecManager, which is instantiated in the application and has to be hold in a session to not loose the state. The JPAAMSecManager will delegate all the required work to several interfaces, which represents the several modular parts of the framework and which provide different interfaces for the different actions, which have to be executed to get a decision for a request. These modular parts are provided by a localizer, which represents an implementation of the strategy pattern[5] and provides the concrete classes set by configuration flags. The localizer can be changed as well by a configuration flag to have a different behavior in providing the requested part of the framework.
As we have described a request is handed over to the framework and makes its way through the framework. A sequence diagram of the process through the framework while asking for a request is illustrated (simplified) in Figure 3

Most parts of the framework are normally not used to decide whether a request is granted or not, or (additional) authentication is used. This is the responsibility of one part: the *security model*. However, all the different pieces of the framework provide the possibility to change the behavior of the framework in different kind ways. For example you can decide which *security model* is used or even change the request itself. In the basic implementation simply one *security model* is provided which is selected based on a configuration flag

2.1 JPAAMSecManager

The JPAAMSecManager it the main interface to the application, which holds an instance of it and can ask with the provided method boolean askPermission(IJPAAMRequest aRequest) to get a decision for a request. The method returns a boolean value with which you can immediately decide if the permission is granted or not. The JPAAMSecManager provides access to the last or i-th decision which are stored in the security session. The application which initializes the JPAAMSecurityManager needs to implement the IJPAAMClient-Interface. This

[5] Frank Buschmann & Stal (1996)

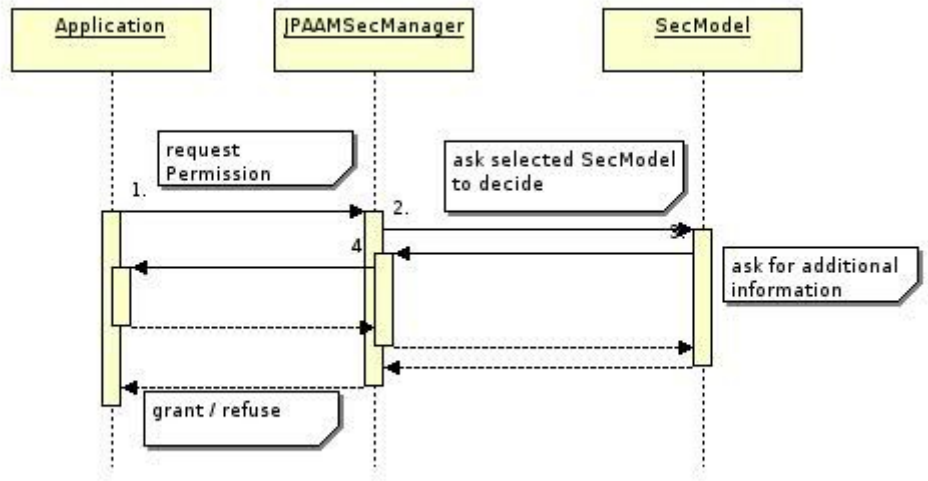

Figure 3: Sequence diagram of a request.

Interface provides some basic methods which are important for the JPAAMSecManager: Properties getJPAAMConfiguration() is used to get any configuration for the framework from the application. This is the only way how the framework is getting its configuration and how it is provided. So it is up to the application how configuration is done and stored. In the default implementation of the framework with its basic implementations of the different interfaces, only the two configuration directives JPAAM.SecModel and JPAAM.Localizer are necessary, which provide the possibility to set the localizer and the *security model* which is provided by the localizer.
void logJPAAM(String aLogEntry) gives the simple possibility to log executed actions of the framework.

A part of the concept of the framework, is that it is possible to request additional information from the application. The third method provides this possibility: Object invokeClientMethod(String methodName, Map values). This method is used to invoke actions on the client side and to return a result of the executed action. How the client handles these *action requests* is up to the client developers side. This method has 3 different exceptions (JPAAMInterruptedRequestException, JPAAMNoSuchClientMethodException, JPAAMClientActionExcpetion) which can be thrown. The last one mentioned is to throw in a wrapped manner all not JPAAM related exceptions. The second one is to signal the framework that the requested invocation of the method can not be handled and therefore the request to execute the given action can't be fulfilled. JPAAMInterruptedRequestException is the most special exception: this exception provides the possibility to interrupt a request from the client side. Most of the time when a security model asks to execute an action it is to get more information from the application side. But it can also be that the application has to stop the actual request. For

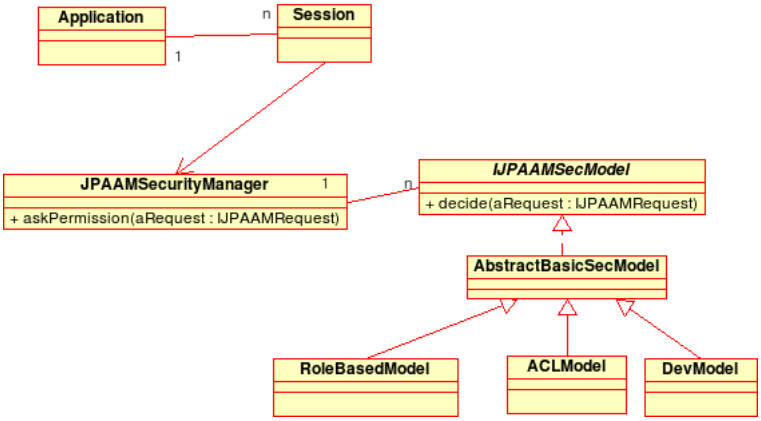

Figure 4: class diagram of the different important classes of the framework

example if the user isn't yet logged in and the application is told to do the login process, so after the application has sent out the login page it has to interrupt the current request, as nothing else should be handled. In most normal Java applications this isn't a big problem. But in web applications which act from request to request you have to interrupt the current request to only serve the new answer the web application has sent out. Therefore you can interrupt the request with this exception, to sign for example the *security model* to stop any further decision taking in the framework, add a denied decision to the decision history with the remark that it was interrupted by the client and then throw the exception to the client.[6]

Localized/Modular Design

Except for the JPAAMSecManager class every other part of the framework is completely dynamically exchangeable due to the architecture of the framework and therefore it possible to add different classes as well as new functions or a completely different behavior. All basic methods which will be addressed within the framework are defined by interfaces, so there is a clear declaration of how the client has to use the framework. Every interface has a so called basic implementation which fits the current requirement and is a general implementation.
The interface IJPAAMSecManagerLocalizer describes the methods used by the JPAAMSecManager to delegate the requests and load the correct security model(s), security session etc. The basic localizer provided by the framework is fully configurable with some configuration directives, so it is possible to select for example the security model which should be used.
The localizer itself is described by an interface and is as well fully exchangeable with the JPAAM.Localizer configuration directive. This gives in example the possibility in an application to write a localizer which returns a security model

[6]For example in our case study in the old style model: when Mir realizes that the user with the actual requests isn't yet logged in it invokes on the client side the action login. This action simply sends out the login page and then throws the JPAAMInterruptedRequestException. This will cause that nothing else is later sent out by Mir to the client while finishing the request.

depending on the request or other circumstances. We have used this in our case study to return the built-in AllPermittedModel for a request of the OpenPosting-Module, as basically in Mir everyone is allowed to do OpenPosting and to return the OldStyleModel for administrative requests.

2.2 IJPAAMRequest

In general a request describes for what exactly the application needs permission. So it contains all the information which should be necessary to decide the permission for this request. The main identifiers of a request are the action for which permission is asked and the current context of the application. The two describers are held in a very general manner, which should allow the framework, especially the *security model*, to identify the request uniquely. This is very important as a request can hold many other pieces of additional information which are needed but the request itself should still be identifiable for example from where in the code this request is derived.
The interface implements the standard Java interface Map which should provide the possibility to add and get in a very general manner additional information of the request. So additional information can be provided with the methods provided by the Map-Interface.

IJPAAMRequestTranslator

Another goal of the proposed framework is to give the possibility to reuse an already implemented security model in another application. The translator gives the possibility to translate a given request to an understandable request for the reused model. As the framework is working within the request with *keywords* of the context of the application and the action which should be executed, they are still always related to the application. However, if there are similar applications with similar needs for a *security model* it is an advantage to have the possibility to reuse an already implemented model for another application. To avoid problems, which for example can appear only from a different naming convention within the request, the translator can be used to translate a request in an understandable format for the reused *security model*. Therefore you wouldn't need to adapt the current model to the new application and it is possible to reuse it.
The interface offers one simple method translate which simply accepts a request and returns the (translated) request. The translator is provided by the localizer and therefore fully configurable. The basic localizer is looking for a configuration string like: JPAAM.SecModel.$SecModelName.Translator. If none is provided the BasicRequestTranslator is loaded which simply returns the given request with no modification.
Criticism: In practice abstraction of a model is very hard and therefore it seems nearly impossible to abstract a model so that it can be reused in a another application. But by overriding some methods/behavior of the reused model it should be possible to reuse already implemented stuff. However, a translator gives as well the possibility to change an incoming request not only for translation purposes.

2.3 IJPAAMSecModel

This interface describes a *security model* and the most important method is IJPAAMSecModelDecision decidePermission(IJPAAMRequest aRequest) which is the only method used by the JPAAMSecManager to ask for a decision for a request. The returned class IJPAAMSecModelDecision holds the decision with additional information and the request itself and is later added to the security session.
The *security model* needs also to know the calling JPAAMSecManager to have the possibility to ask additional information from the application. Due to restrictions[7] in loading the classes dynamically, we need to have the possibility to set JPAAMSecManager manually, therefore the void init(JPAAMSecManager secManager)-method ensures that the specific *security model* is initialized with a JPAAMSecManager.

Integrated SecModels

There are two *security models* integrated in the framework: AllDeniedModel and AllPermittedModel. They provide two samples of an implementation of the interface and should provide possible models for development purposes. As they are programmed in a very general manner and can easily be reused, they are integrated in the framework.
The abstract class AbstractBasicSecModel provides a basic implementation of the init handling and holds a JPAAMSecManager. Most new models can be extended from this abstract class.

2.4 IJPAAMSecuritySession

When the JPAAMSecManager is initialized it receives from the localizer a *security session*. This session holds all the information which are needed within the framework. Every decision which is taken is added to the *security session*. So whenever a trackdown of the asked requests is necessary, this can be accessed through the decision stack.
The security session can also hold important information about the current running session, for example the username and password of the logged in user. This gives the possibility to store information over the whole lifetime of the JPAAMSecManager, in web applications (or other stateless applications) the JPAAMSecManager should therefore be stored in a session to ensure that the information isn't lost during the next request.
When the JPAAMSecManager gets invalidated the *security session* is invalidated as well. In most applications this is like logging out, as the provided authentication information is cleared and therefore authentication needs to be redone.

3 Case study - Mir

For the case study to integrate the proposed framework we had chosen Mir[8]. Mir is a well designed and highly configurable application, but uses a very basic

[7] Objects in Java can not be instanced dynamically with a parameterized constructor while using the default Classloader.

[8] `http://mir.indymedia.org`

security system, which we wanted to extend to have the possibility to use other security models.

3.1 What is Mir?

Mir is a Content Management System (CMS) written in Java and using Tomcat as a Servlet Container and PostgreSQL as persistence storage. It is mainly used and developed for Independent Media Centers (IMCs)[9], which are independent news sites based on the open posting concept.
Mir has several interesting features and for a CMS a bit a special way how content is contributed and represented. Technical Features are:

- Static publishing
- Very configurable
- Structured Object oriented 4 layer design.
- Meta-data/Database schema and classification based on the Dublic Core standard
- Supports categorization into topics and media folders
- Modell and View are strictly separated.
- Many more

As open posting means that content can be submitted by anyone and that it appears directly on the site. The main contributer of content to sites using Mir are the people coming across the different sites. These users can currently only post content to the site but not do any editorial work. This is often done by a group of media activists. Static publishing means that there is no content viewed directly in connection with the database. Every content is produced by a highly configurable producer framework into static html/shtml pages. Therefore it isn't really the goal of Mir to have a restricted view of the content, as it is often used for news sites.

3.2 Current problems with the actual security system

As mentioned above most editorial is done in a group, the current security context was also developed in this way of working. So therefore someone can be User, Admin or Superadmin. If OpenPosting is disabled, which means the publishing of content is not open, users and above logins can post content with their login. There is no possibility to sign-up as user. Users can as well not edit any of the content, they or others published. Admins can always edit any kind of content in the database, even they can add or remove languages, topics etc. (which is in a way system sensitive content). Superadmins can add, change and remove Users. Superadmins are defined in the main configuration file of Mir, which makes it in a way very static as the application needs to be restarted if something changed. There is no possibility to restrict actions of an Admin to a special kind of an article, type etc. If someone is Admin she can do everything.

[9] `http://www.indymedia.org`

As Mir the security system of Mir had done the development process like we have described above and therefore it got hard-wired within the application, we therefore selected Mir to test our proposed framework. With the integration we can see if our framework gives an advantage and enables the possibility to use other security systems.

```
try {
  MirGlobal.accessControl().user().assertMayAddUsers(ServletHelper.getUser(aRequest));

  showUser(null, false, aRequest, aResponse);
}
```

Figure 5: call with the old security system to check a permission

3.3 Integration of the framework

The integration of the framework in Mir and the rebuilt of the current security system of Mir took several steps:

- Analyzing Mir and finding the relevant places in the code
- Removing the old code pieces related to authentication and authorization
- Integrating the framework into all relevant places in mir
- Writing the *old security model* as a IJPAAMSecModel
- Writing additional models

As already mentioned model and view are strictly separated in Mir. This was a plus in finding the relevant parts where to put the new hooks of the framework. While analyzing the architecture of how a request is handled and where everywhere are the old parts of the security system we found 2 main parts, where the calls for permission have to be integrated in Mir.[10]
The first part are the 2 main classes Mir (Mir.java) and OpenMir (OpenMir.java). The separation in these two classes is due to historical reasons of the development of Mir. OpenMir handles all the OpenPosting part of Mir and Mir the administrative parts. So by every request one of them is called and they choose the right servlet according to the get parameter to delegate the request to the right module. These two classes are as well passed as a client to the JPAAM-Framework and are handling therefore the additional requests from the framework. As both are extended from an abstract class most of this handling could be integrated in this class.
The servlets are the second part of the application where the calls to the framework have to be placed in. The servlets are found in mir.servlet and mircoders.servlet and provide the collecting of information and filling the response. The calling of the method is dynamically according to the GET-Parameters *module* and *do*, so a first check can be placed in the dynamic loading of the methods to check if on the whole method permission is granted or not, but for

[10] A more detailed overview in German: `http://docs.indymedia.org/view/Local/ImcDeMirDocu`

more granular permissions requests i.e. depending on the data to edit additional permission requests have to be placed inside the methods filled with the data of the request or already collected data.

To map the current known data for a request into the permission request most of the time simply all values of the request are being added to the permission request. As both application request and permission request are maps this can easily done by with one method call. In some methods data is collected, aggregated or reorganized, therefore there the permission request has to be filled manually with the data.

Listing 1: simple permission request (mircoders.servlet.ServletModuleAbuse / editfilter)

```
1        this.checkJPAAMPermAndThrow(
2          JPAAMHelper.getABasicRequest(responseData));
```

Listing 2: permission request collecting data (mircoders.servlet.ServletModuleContent / editObject)

```
3        IJPAAMRequest request = JPAAMHelper.
4          getABasicRequest(responseData);
5        request.put("id",anId);
6        this.checkJPAAMPermAndThrow(request);
```

Old security model

As described in section 3.2 the OpenPosting part of the application is open for all, while the administrative part of the application is restricted to logged in users and some parts within only to super users. As mentioned in 2.1 the framework implemented a completely modular design, therefore the implementation of the *Old security model* is using this feature to use two different IJPAAMSecModel s depending on the request. This is done by overriding the method IJPAAMSecModel getSecModel(IJPAAMRequest aRequest) of the BasicSecManagerLocalizer with mir.jpaam.MirOldStyleJPAAMLocalizer which returns depending on the request the model for all non-admin or admin-related requests. To rebuild the old behavior within the admin related context of the application the class mir.jpaam.model.OldStyleModel is responsible.

Additional models

As the framework is now integrated in Mir it should be easy to switch between different models and write new models for it. However, there is still some coupling between model and the application which needs anyway additional configuration, as well reorganization of the code in Mir. Especially the view part of the application needs to have customized templates for the different models. Due to the time limit of this bachelor work it wasn't possible to work more on this issue. However, the functionality of the framework can easily be shown by switching between the *OldStyleModel* and the *AllPermittedModel*, by changing only some configuration methods in the properties file of Mir.

4 Related work

The problem of having *security systems* too much hard-wired into the application is a known problem and there have been several attempts to get a proper and general solution for it.
One possible solution is provided by Sun itself with their JAAS framework which will be discussed later in this section. Another related work is the master thesis of Daniel Vogel which worked on problems with the basic security system of SmallWiki

4.1 JAAS

JAAS[11] is a set of APIs provided by Sun, which provides a pluggable authentication by username and authorization based on roles. The framework extends the standard security API of Java[12] and provides extended features for it.
The authentication is simply based on a username and a password, but the backend is like in the (Unix-)PAM pluggable to any technology (LDAP, etc.) which is required by a user. This leads to the usage that you can develop an authentication part in the application with JAAS which is independent from the backend and therefore the development process doesn't depend on a specific backend and the authentication part is clearly separated from the rest of the application.
The authorization part is based on a simple but finely configurable role based system. The username is simply mapped to several roles which are then needed if the authorized user has access to the different code parts. So the authorization is based on if the user is authenticated or not and is mapped to a specific role. This is then checked against some code characteristics and permission is granted or not.
The aspects of JAAS are very similar to JPAAM but however there are some differences which are disadvantages or provide also a better solution. The authentication is puggable but the fact that it can only be based on a username and a password makes it less flexible[13]. This is in most cases sufficient but leads to the fact that however it is rarely possible to adapt for example a third parameter like a pin to be essential for authentication.
The finely configurable role based part makes it very widely usable but again it depends on one solution, which may fit for the most required authorizations. But it makes the application depend on this kind of authorization.
As JAAS extends the standard security API of Java it has to be simple, and loaded once in the beginning of the application. Afterwards it is supervising the calling of the code from *outside* the application code and is authorizing access to the code based on the given rules. This leads to the fact that you can write code without having any security system relevant code parts within, which is a very simple and nice way to write code. However, a developer has to respect the restrictions which come from the way how authorization is done with the role model, so development process is still not completely free from the underlying *security model*.

[11] `http://java.sun.com/products/jaas/`
[12] See: LiGong (1999)
[13] There are ideas to change that: `http://www.adtmag.com/article.aspx?id=10158`

4.2 Management and Security of Collaborative Web Environments by David Vogel

David Vogel reworked in the context of his master thesis the existing security system of SmallWiki[14], a Wiki written in Small Talk.
The Wiki had implemented a *security model* which was called the SmallWiki Default Security Model, which contained users and groups as well as access limitation based on a hierarchical level tree, which represented the structure of the wiki. This *security model* had been implemented to prevent vandalism of the wiki installation and other problems. It allowed as well to separate different parts of the wiki to different groups which could decide independently about permissions for their part of the wiki. However, from these Multi-Admin-Setups resulted other problems which leads one to the fact that it was possible for administrators of one part to set access level rights of other. [15]
David Vogel analyzed the Default Security Model of the SmallWiki and introduced an Extended Security Model, which is based on the Default Security Model, but which prevents the problems that resulted from the Default Security Model. However, the solution he provided is not that generic like JPAAM's solution and is in fact a concrete extension of the Default Security Model, with the goal to prevent the known bug. Therefore he followed another approach (to fix a known problem of the concept) than we did with our proposed framework (to find a generic solution).
He analyzed as well the Default Security Model in detail and worked out a considerable documentation of it and security in collaborative web environments at all. The solution introduced by this work is a specific solution for one application and not a general purposed way to deal with management and security of collaborative web environments. However, the discussion of *security models* in (web) applications is very detailed and covers several aspects and therefore it provides some good aspects.

4.3 Aspect Oriented Programming (AOP)

There are other approaches which try to find a solution. Another reasonable approach can be found in aspect oriented programming[16] (AOP), as security exemplifies cross-cutting concerns. Therefore a solution with AOP can be seen as a similar solution to our proposed framework, as AOP attempts to solve the problems that operations appear scattered across numerous methods. Viren Shah and Frank Hill have proposed such a security framework in 2003[17] using AOP as a solution so solve these problems. However, JPAAM is not using AOP as a concept and therefore it is still following another approach.

5 Conclusion

With the framework we proposed we provide a more general solution which avoids being fixed on one specific solution and one kind of authorization and

[14] `http://smallwiki.unibe.ch/smallwiki`
[15] More in section 4.3.3 Vogel2004
[16] `http://en.wikipedia.org/wiki/Aspect-oriented_programming`
[17] Viren Shah (2003)

authentication. This gives, as we have shown, the possibility of being independent from the users wishes regarding the kind of a specific security system during the development process. With a very loose or strict model it is as well easy for a developer to check very fast the current implementation progress of the application. Therefore this framework gives an advantage for the developer in the development process as he can later think about the exact implementation of a possible framework or delegate this work simply to another developer.

The technical implementation of the framework is very generic, but highly configurable and modular using well known design patterns. Therefore it is possible to adapt the basic structure with small or bigger changes to the current need of a developer. The possible interaction of the framework with the application gives the possibility to interact with the application and to invoke as well functions on the application side.
We have shown this in our case study where we changed Mir to use the proposed framework and adapted its old model to the framework. We have changed a part of the framework with a few lines to use two different *security models* depending on the context of the application. With this solution we could very easy implement a model which interacts on a public part open with no restrictions, but on the closed side with restricted authorization. For future changes in the application it would be easily to adapt only some parts of the models new and gain a different kind of authorization or authentication.
However, what had been realized as well with the case study is that it can be hard for applications, which depend in their deeper structure on functionalities of a *security model* part, to be really separated from the used *security model*. For example if Mir would later want to use the current authentication model, which is based on a username and password, to restrict specific actions to a given article (*i.e.*, only the owner of an article can edit it), the additional information of the connection between owner and article would have been to be stored and managed completely separated from the application data, in the way that it is still possible to switch without changes in the application for example to the AllPermittedModel. Unfortunately there could not have been done more research on this aspect.
With the proposed framework we give a possible and feature-rich solution to the problem we analyzed in the beginning of this work.

References

Frank Buschmann, Regine Meunier, H. R. P. S. & Stal, M. (1996), *Pattern-oriented software architecture: a system of patterns*, John Wiley and Sons Inc.

Jonscher, D. & Dittrich, K. R. (1993), *A Formal Security Model based on an Object-Oriented Data Model*, University of Zurich.

LiGong (1999), *Inside Java 2 Platform Security*, Addison Wesley.

Tanenbaum, A. S. (2003), *Computernetzwerke*, 4 Aufl., Pearson Education, Munich.

Thuraisingham, M. (1989), *Mandatory Security in Object-Oriented Database Systems*, ACM SIGPLAN Notices.

Viren Shah, F. H. (2003), 'An Aspect-Oriented Security Framework', *DARPA Information Survivability Conference and Exposition* **2**, 143.

Vogel, D. (2004), *Management and Security of Collaborative Web Environments*, University of Bern.

List of Figures