GholamAli Nejad HajAli Irani / International Journal of Engineering Research and Applications (IJERA)
ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 2,Mar-Apr 2012, pp.697-703

A new modular Authentication and Authorization architecture for web portals and content management systems

GholamAli Nejad HajAli Irani
Faculty of Engineering, University of Bonab, Bonab, Iran

ABSTRACT

All web based Information Systems such as web portals and content managements systems (CMS) need an Authentication and Authorization (AA) architecture. More than 1200 web portals and content management systems have been developed as yet. Existing portals and CMS use a similar centralized approach that gather all similar parts of system in a common part which is named Core. So with increasing the scale of system, scale of the Core will increase as well.

In centralized approach, all functionalities of AA perform by Core. With increasing scale of system, this architecture will be faced with many problems.

In this paper, a new decentralized architecture has provided for AA in web portals and content management systems. To obtain this aim, firstly, all existing approaches and their disadvantages for AA, has been investigated and categorized. Secondly, to obtain a new architecture and solving disadvantages of existing approaches, new AA principles has been developed using robust object oriented principles and heuristics. New architecture has been developed based on these obtained AA principles.

Finally, for evaluation of new architecture, it has been shown that all requirements of existing approaches covers by new architecture.

Keywords **- Modular Software Architecture, Authentication and Authorization, Object Oriented Analysis and Design, CMS**

I. INTRODUCTION

Authentication is the process of confirming someone or something's identity [4]. It can be considered as giving answer to this question: Are they the ones who they say? Authorization is the process of allowing someone or something to actually do something [4]. Can they do this?
CMS and web portals as web based information system are composed of several modules [16]. More than 1200 web portal and CMS is presented in web applications [17]. For example, Drupal is composed of more than 8700 modules [18].

Every information system must have authentication and authorization part to perform and apply security features of each part of system. All existing web portals and CMS have used a centralized approach. In centralized approach all similar functionalities of system have been collected and gathered in a common part of system which is named Core. AA of any system, at the first glance, is look at similar functionalities. Therefore in existing architectures all functionalities of AA perform by the Core.

As increasing the scale and complexity of system, scale and complexity of Core is increasing as well. Then management of Core is turned to a big problem. On the other hand, while extending and modifying the Core, all modules might change. Therefore as increasing scale of Core, the Core can turn a GOD module [1].

There are many patterns of Authentication and Authorization of systems which are described in part 1 and each day is added to the number and variety of them. So with putting all these patterns into Core, the dependency of modules to Core is increased and modularity of whole system is decreased. Centralized approach has some other disadvantages which are described in part 2.

Our suggested architecture for modular CMS and web portals is presented in Fig. 1. The main difference of our suggested architecture in comparison with other existing architectures is usage of robust object oriented principles and heuristics in designing “Core Functionalities” and its communications with other modules. Based on suggested architecture all modules have to control their contents by themselves and Core just prepares an infrastructure for that. In this architecture, all authorization functionality of each module, granted to itself. Meanwhile, authentication functionality performed by Core.

To support maximum extensibility and modifiability for communication between Core and modules, we used event-driven architecture [13]. To support various implementation platforms and maximum compatibility we used XML as communication protocol [14].

The aim of this paper is to provide a new architecture for AA in web portals and CMS which can be apply as a standard for AA in any Modular IS. New method is based on decentralized approach and tries to distribute the AA functionalities between modules.

To obtain this aim, Firstly, previous architectures and patterns of AA have been investigated. Secondly, disadvantages of centralized approaches have been investigated. Thirdly, to decentralize and distribute AA functionalities between modules, robust Object Oriented heuristics have been used and some modular decentralized principles has obtained. Then new AA decentralized principles extracted from these modular decentralized principles. Then, a new modular AA architecture for web

portals and CMS has been proposed. Finally, for evaluation of new architecture, it has been shown that all requirements of existing approaches covers by new architecture.

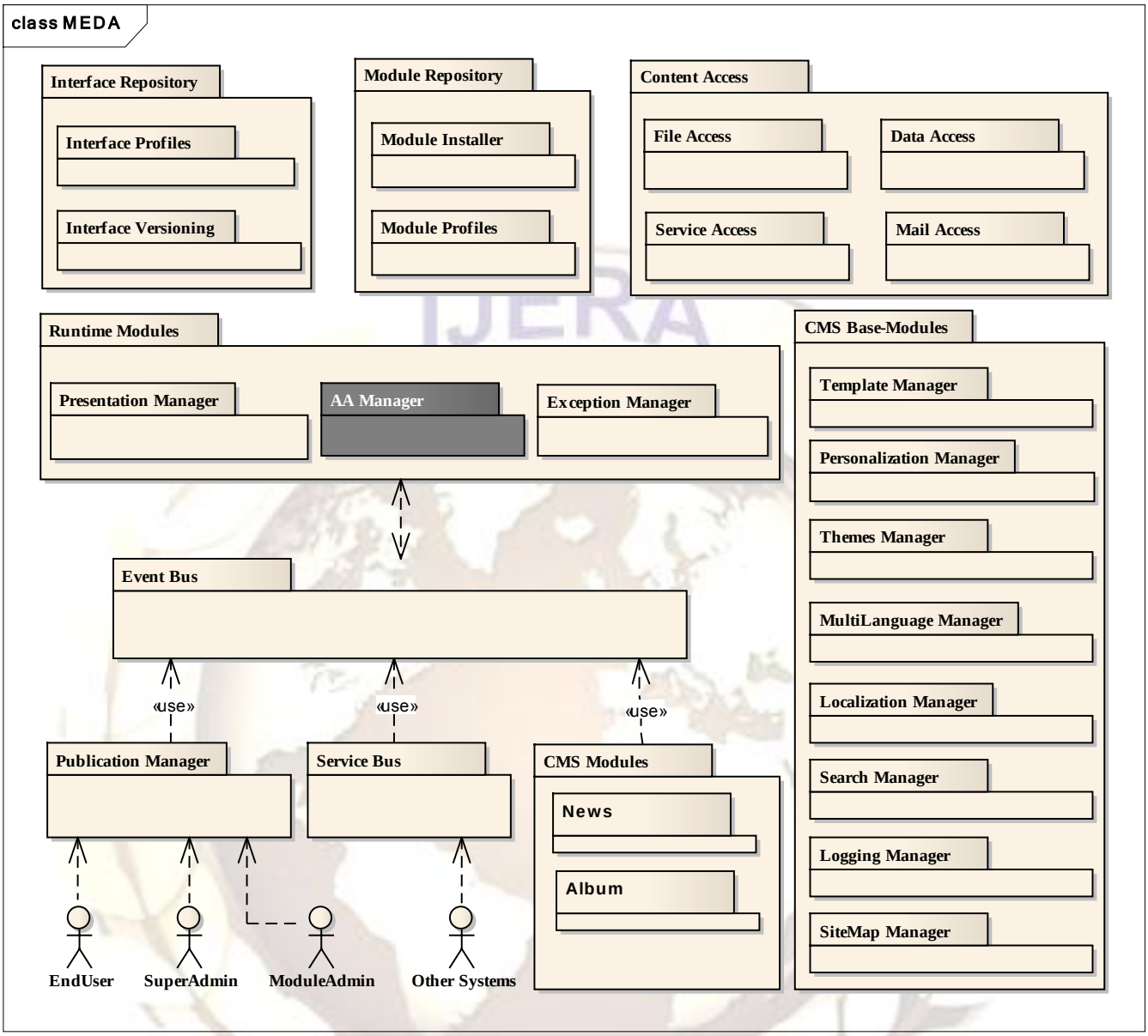

Fig. 1. Proposed modular architecture for web portals.

II. INVESTIGATING PREVIOUS APPROACHES

This section has been provided in three categories. Firstly, AA architectures of web portal and CMS have been provided, and then AA approaches in other areas have been investigated and presented.

2.1 AA Architectures of web portals and CMS

To examine AA architecture of web portals and CMD, we used 20 most powerful CMS based on [19] and investigate all their architectures. These are as following: Alfresco WCM [20], CMSMadeSimple [21], Concrete5 [22], DotNetNuke [23], Drupal [24], e107 [25], eZ Publish [26], Joomla! [27], Liferay [4], MODx [28], Movable Type [29], OpenCms [30], Plone [31], SilverStripe [32], Textpattern [33], Tiki Wiki CMS Groupware [34], Typo3 [35], Umbraco [36], WordPress [37], Xoops [38]. None of them used decentralized approach.

Some of the architectures have used other approaches. For example [39], [40] and [41] used component oriented approaches, and [42] have used a pattern based approach. Some another architecture like [4] to reach a high quality of extendibility, used an event driven approach. But none of them used decentralized approach.

2.2 Previous AA approaches

Previous studies of AA used a centralize approach as well, hence all AA data and its implementation is performed by Core [4]. However some studies distributed AA data into modules, Core is controlling and deciding about AA [5]. This centralized thinking has some inadequacy which will be discussed in the reminder.

To authenticating, some systems uses an standard account management such as LDAP, SSO, NTLM, OpenID, OpenSSO and Site Minder etc [2], [3], [4], [5].

To perform authorization, a variety of resources of a system can be accessed by user in different levels. These resource types can be Application, Portlet, Locations and Files (Content-Model-Resources), Communications, Pages (or Forms), Use Cases (or Actions), Tables (or Database Entities), Objects (or Business Entities).

To authorizing, various approaches are provided. Role-Based Access Control (RBAC) is the common method for controlling user access to system resources and actions [6]. Some of the approached, for special uses, separated authentication from authorization [7]. These methods used federated user administration, therefore authentication performs in user's home system and authorization performs in service provider system [8]. To support extensibility and modifiability, AA methods used some new tools like Aspect Oriented Programming Languages [9]. Open Service Gateway initiative (OSGi) framework uses java standard called Java Authentication and Authorization Service which is a centralized approach [15].

Cristian and Gabriela showed that by distributing the security functions, a more flexible architecture can be designed that would lower the costs associated with implementation, administration and maintenance [10].

III. Problems of Existing Architectures

One of the important advantages of being centralized is that AA functions can be developed for once and all the modules can use the same functions and then the complexity of modules decrease. In the other hand, being centralized can cause numerous problems in development of systems which categorized as fallowing:

Req1: Modules have to use the AA pattern (AAP) which is implemented in Core. So, modules cohesion is increasing and dependency on Core is increasing as well, therefore modularity of system will be decreased.

Req2: Developing small-scale modules need to follow the AAP of the Core. Consequently, complexity of developing small-scale modules will be increased.

Req3: The implemented AAP of the Core is not complete in general. Probably, developing large-scale modules is needed to use a new AAP which is not supported by the Core.

Req4: Due to centralized approach and dependency of modules on Core, performing a Unit Test on modules is difficult and quality of testability is decreasing.

Req5: Due to variety of authorizations based on different resource types, considering all of them in the Core cause to complexity of Core.

Req6: Because of centralized approach, integration of implemented modules into different systems with different Cores takes some efforts due to lack of standard AA interface. Therefore system integrity and modules portability decrease. This problem will emerge on "Plugging In" a new module or making a group of existing systems "Single Sign In".

Req7: In centralized approach, the overall AAP of the Core (so-called Big Picture) is apparent to all modules. So, encapsulation of AAP is violated.

Table 1 presents the relationship obtained from the categorizing of above-mentioned requirements with software architecture quality attributes. In previous studies which are described in part 2, none of above-mentioned problems are considered.

Table 1. Quality Attributes Affected By Requirement List.

	E	M	Mo	I	Io	P	T	LEGEND
Req1			x					E: Extendibility;
Req2	x		x					M: Modifiability;
Req3	x	x						Mo: Modularity;
Req4							x	I: Integrity;
Req5	x				x			Io: Interoperability;
Req6				x		x		P: Portability;
Req7			x		x			T: Testability.

IV. Modular AA Principles

To provide a new architecture for AA in CMS and web portals, we used Object Oriented principles and heuristics provided in [1]. Based on [1], we can consider a module as an object, and then apply object oriented principles to obtain new approach to modular development principles. List of used Object Oriented Heuristics from [1] are: Heuristic 2.1, 2.2, 2.4, 2.5, 2.6, 2.9, 2.10, 3.1, 3.2, 3.7, 3.8, 4.1, 4.2, 4.3, 4.4 and 5.3.

By analyzing concepts of above-mentioned heuristics and their relation with modularity concepts, we can provide some advice for developing modular systems which is shown in Table 2. In the centralized approaches, Core will become a God module.

Table 2. Object Oriented Heuristics to Modular Advice.

Heuristics	Code	Provided Advice
H2.1, H5.3	M1	Each module should hold and manage its data within itself.
H2.9, H2.10	M2	Each module should perform all its functionalities by itself.
H2.2, H2.3, H2.4, H2.5, H2.6, H4.1, H4.2, H4.3	M3	Optimize and minimize module interface.
H3.1, H3.2, H3.7, H3.8, H2.9, H2.10	M4	Distribute common functionalities of modules to themselves.

Authorization of each module doesn't belong to Core functionality. Therefore authorization should not be centralized and its functionality should be distributed horizontally among modules. According to provided advice in Table 2, we can obtain some AA principles which are shown in Table 3, in order to provide a new architecture.

Table 3. Obtained AA Principles.

Advice Code	Principle
M4, M2	Each module has to perform its authorization by itself.
M4, M1	Each module has to hold and manage its authorization data.
M3	Standardize an AA interface between Core and Modules.

V. NEW AA ARCHITECTURE

Fig. 2 illustrates detailed AA architecture based on principles which are shown in Table 3.

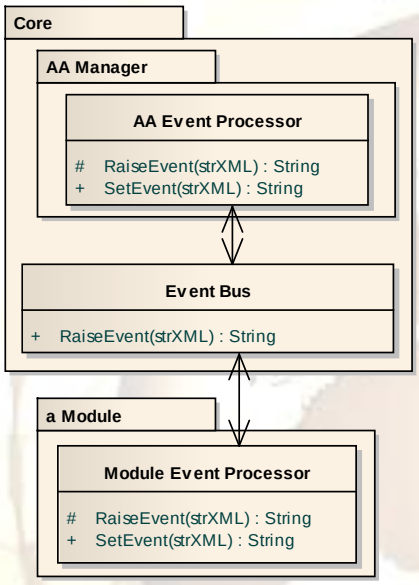

Fig. 2. AA Manager detailed architecture.

In this architecture, each module and Core as a module, have to implement a class by the name of EventProcessor and use EventBus as a channel for interacting messages between Core and modules. In this architecture we put messages in the form of Events. All of modules and even the Core use RaiseEvent method from EventBus for sending events and EventBus uses SetEvent method from EventProcessor (implemented in each module) for sending delivered events to target modules. Security of interaction between Core and modules is performed by EventBus.

Interaction between Core and modules are prepared by two XML files by the name of Document Type Definition 1 (DTD1) and DTD2. For example after a user logged in to system, Core will send a request (in the form of an event) to get User Access List from all the modules to represent User Control Panel. This action will perform by the use of an event like: +getAccessList(String Username):String;

DTD1 is a template for sent events. DTD1 contains event-type, event-name, input parameters names and values, return type and value, event-sender, event-receiver(s), etc. An instance of DTD1 presented in Fig. 3. The standard DTD of DTD1 has been shown as following:

```
<?xml version="1.0"?>
<!DOCTYPE DTD1 [
<!ELEMENT Event
(InputFields,ReturnType)>
<!ATTLIST Event
        EventType       CDATA #REQUIRED
        EventName       CDATA #REQUIRED
        getAccessList   CDATA #REQUIRED
        EventID         CDATA #REQUIRED
        SenderName      CDATA #REQUIRED
        SenderID        CDATA #REQUIRED
        ReceiverNames   CDATA #REQUIRED
        ReceiverIDs     CDATA #REQUIRED
        RaiseDateTime   CDATA #IMPLIED
        Description     CDATA #IMPLIED>
<!ELEMENT InputFields(Field+)>
        <!ELEMENT Field(EMPTY)>
        <!ATTLIST Field
                Name    CDATA #REQUIRED
                Value   CDATA #REQUIRED >
<!ELEMENT ReturnType (DTD2*)>    ]>
```

Each module for sending an event must put it in the form of DTD1 and invoke RaiseEvent method from EventBus. Then EventBus analyze delivered event and in order to sending event to target modules, use the SetEvent method from EventProcessor class of each module.

After that, all recipient modules can response to this event by returning value of SetEvent in the form of DTD2. DTD2 contains returned values of each module. An instance of DTD2 presented in Fig. 4. The standard DTD of DTD2 has been shown as following:

```
<?xml version="1.0"?>
<!DOCTYPE DTD2 [
<!ELEMENT ReturnObjects  (Object*)>
<!ATTLIST ReturnObjects
        ModuleName      CDATA #REQUIRED
        ModuleID        CDATA #REQUIRED
        ReturnDateTime  CDATA #IMPLIED
        Name            CDATA #REQUIRED
        Description     CDATA #IMPLIED>
<!ELEMENT Object(Field+)>
        <!ELEMENT Field(EMPTY)>
        <!ATTLIST Field
                Name  CDATA #REQUIRED
                Value CDATA #REQUIRED> ]>
```

```
<Event  EventType="AAMNG" EventName="getAccessList"
EventID="123" SenderName="Core" SenderID="379"
ReceiverNames="Core | a Module | *" ReceiverIDs="12 , 32 |
*"  RaiseDateTime=" " Description=" ">

        <InputFields>
                <Field Name="UserName"
        Value="Jane"/>
        </InputFields>
        <Return  Type="String">
                <!--  return info must be in here in  DTD2
        format -->
        </Return>
</Event>
```

Fig. 3. An instance of DTD1.

```
<ReturnObjects ModuleName="News" ModuleID="123"
ReturnDateTime="" Name="News Access List"
Description="">
<Object>
        <Field Name="ID" Value="1"/>
        <Field Name="Title" Value="Add New News"/>
        <Field Name="URL"
Value="www.test.com/UI/News?1"/>
</Object>
<Object>
        <Field Name="ID" Value="2"/>
        <Field Name="Title" Value="Change News"/>
        <Field Name="URL"
Value="www.test.com/UI/News?2"/>
</Object>
</ReturnObjects>
```

Fig. 4. An instance of DTD2.

Finally, EventBus put all of received DTD2s from each module in the <return> tag of DTD1 and passes the final DTD1 as return value of RaiseEvent.

VI. AA Distilled Analysis and Design

Regarding to previous sections, we can divide all AA use cases into two layers: Authentication Layer and Authorization Layer. Authentication performs in the Core and authorization performs in the modules. Fig. 5 is a distilled analysis and design artifact of AA.

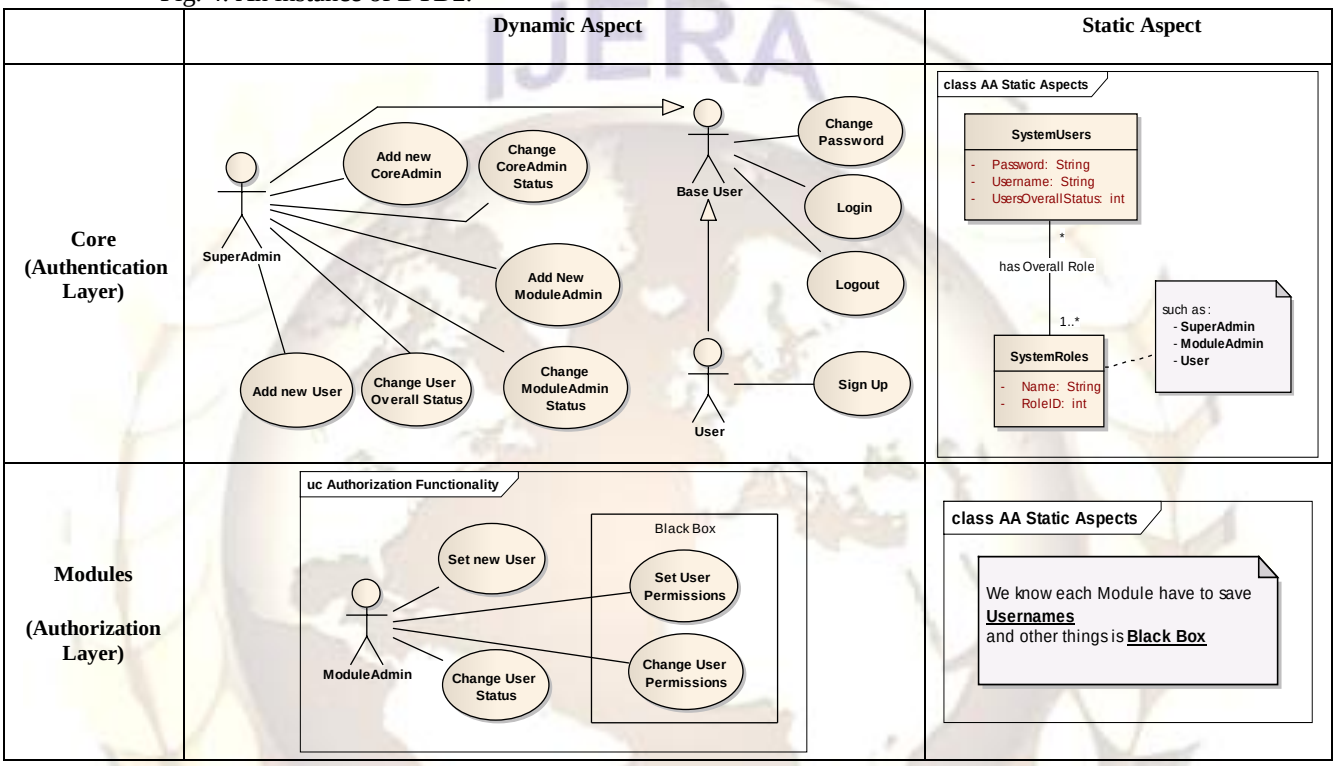

Fig. 5. AA distilled analysis and design.

VII. Optimized Event List For AA

Regarding to Event-Driven Architecture, we should extract an optimized event list for communication between Core and Modules, as it capture all AA use cases. The obtained optimized event list and its descriptions are shown in Table 4. Adding a new event will not affect AA in any way. For example, in order to gather User Overall Log, we can add an event such as: +getLog(String Username):String; into event list. For another example, if modules want to have a different session timeout for each user, at any time it can ask from Core to check the Login Status of that user. It can be done by just adding three events such as: Login and logout from Core to modules and isLogedIn from modules to Core. Since we have not a pure modular content management system that let a module to control its contents just by itself, we can compensate this shortage with adding some simple events to event list. Contents of a module can be Files and Database Tables.

For controlling these contents, modules cannot act independently and Core has to authorize modules to access contents of each other. Therefore, as any user want to access any content, Core asks for user permission from corresponding module.

So, additional events are as following:

+ActionToTable(String Username, String TableName, TableAction action):Boolean;

+ActionToFile(String Username, String FileName, FileAction action):Boolean;

TableAction can be CRUD (Create, Read, Update or Delete) and FileAction can be RW (Read or write).

Core can be considered as a module. It means that Core has to authorize its users as well as other modules. Every event which has risen by Core should be responded by Core as well as other modules.

VIII. EVALUATION

In section 2, we categorized a requirement list as problems of existing method. The provided architecture in this paper for AA Manager captures all of these requirements which are shown in Table 5. In fact, AA improves all quality attributes which are mentioned in Table 1.

Table 4. Optimized Event List For AA.

Use cases	Events	Description
New User, Sign Up	NewUser(String Username):void;	As soon as a new user registers in system, Core should inform all the modules, so the modules can grant default permissions to he/she.
New ModuleAdmin	NewModuleAdmin(String Username):void;	As soon as a new ModuleAdmin registers in system, Core should inform the target module.
Change User Overall Status	ChangeUserStatus(String Username, UserStatus status):void;	When Core changes overall status of a user, should inform all the modules.
Login	getAccessList(String Username):String;	For creating Control Panel for a user.
Other Use Cases	---	For other cases modules act independently.

Table 5. Requirement List Covered by new architecture.

Description	Captured Requirements
Modules are independent in selecting their own AAP. They just have to consider Core's standard interface.	Req1, Req2, Req3
Since modules are not dependent to Core for authorization, unit test of each module can perform easily far from the Core.	Req4
Modules are free to choose their authorization types. If a modular content management system is not available, Core has to ask for permission from corresponding module.	Req5
As establishing a new standard interface for Core and modules communication, integrity and portability of modules was increased.	Req6, Req7

9. Conclusion and Future works

In this paper, a new AA architecture as a new extensible and modifiable architecture for web portals and content management systems by a decentralized approach has been provided.

Provided architecture can be use in Enterprise IS, Service Oriented Platforms and any large-scale modular software. New architecture used an Event-Driven method, so changes can be applied easier by adding new event in it and used XML templates as its communication protocol which makes it so understandable for different platforms. Considering that provided architecture is based on robust object oriented principles and developed in a decentralized approach and distributed complexity of the Core among modules, so module development will take extra effort than before. Although it could be a disadvantage in comparison with centralized systems, this extra effort is worth benefiting of being decentralized.

The Process which followed in this paper can be apply to development of other part of suggested architecture in section 1 such as developing a Modular Data Access, Modular File Access and Modular Service Access (Modular Content Access).

REFERENCES

[1] A. J. Riel, *Object-Oriented Design Heuristics*, Addison Wesley, 1996.

[2] R. Jay, *SAP NetWeaver Portal Technology* – The Complete Reference, McGraw Hill, 2008.

[3] M. Shariff, V. Choudhary, A. Bhandari, P. Majmudar, *Alfresco 3 Enterprise Content Management Implementation*, PACKT Publishing, 2009.

[4] J. X. Yuan, *Liferay Portal 6 Enterprise Intranets*, PACKT Publishing, 2010.

[5] K. Pope, *Zend Framework 1.8 Web Application Development*, PACKT Publishing, 2009.

[6] R. S. Sanhu, Role hierarchies and constraints for lattice-based access controls, *In Proceedings of the Fourth European Symposium on Research in Computer Security* (ESORICS96,Rome, Italy, Sept. 25-27), E. Bertino, Ed. Springer-Verlag, New York, NY, 1996.

[7] R. Castro-Rojo, D.R. López, *The PAPI System: Point of Access to Providers of Information*, Terena, 2001.

[8] M. Steinemann, T. Spreng, A. Bachmayer, T. Braun, C. Graf, M. Guggisberg, *Authentication and Authorization Infrastructure: Portal Architecture and Prototype Implementation*, IAM-03-012, 2003.

[9] G. Ahn, H. Hu, J. Jin. Security-Enhanced OSGi Service Environments, *IEEE Transactions on Systems, Man and Cybernetics—Part C: Applications and Reviews*, Vol. 39, No. 5, September 2009.

[10] C. Opincaru, G. Gheorghe, *Service Oriented Security Architecture*, 2008.

[11] M.Sojka, P. Piša, D. Faggioli, T. Cucinotta, F. Checconi , Z. Hanzalek, G. Lipari, Modular software architecture for flexible reservation mechanisms on heterogeneous resources, *Journal of Systems Architecture*, 2011.

[12] P. Intapong, S. Settapat, B. Kaewkamnerdpong, T. Achalakul. Modular Web-Based Collaboration

Platform, *International Journal of Advanced Science and Technology*, 2010.
[13] O. Etzion, P. Niblett , *Event Processing in Action, Manning Publications*, 2011.
[14] D. Hunter et al. *Beginning XML, 4th Edition*, Wrox Press, 2007.
[15] R. S. Hall, K. Pauls, S. McCulloch, D. Savage, *OSGi in Action*, Manning Publications, 2011.
[16] B. Boiko, *Content Management Bible*, 2nd Edition, Wiley Publishing, Inc., Indianapolis, Indiana, 2005
[17] The *Content Management Comparison Tool*, available at http://www.cmsmatrix.org
[18] *Drupal, Open Source CMS*, available at http://Drupal.org/Project/Modules
[19] *The 2010 Open Source CMS Market Share Report*, water & stone, available at www.waterandstone.com
[20] D. Caruana, J. Newton, M. Farman, M. G. Uzquiano, K. Roast, *Professional Alfresco*, Wiley Publishing, Inc, 2010
[21] S. Goldstein, *CMS Made Simple Development Cookbook*, Packt Publishing, 2011
[22] *Concrete5 free CMS*, Open Source Content Management System, available at http://www.concrete5.org/documentation/developers/
[23] S. Walker, B. Scarbeau, D. Hardy, S. Schultes, R. Morgan, *Professional DotNetNuke 5, Open Source Web Application Framework for ASP.NET*, Wiley Publishing, Inc, 2009
[24] M. Butcher, G. Dunlap, M. Farina, L. Garfield, K. Rickard, J. Albin Wilkins, *Drupal 7 Module Development*, Packt Publishing, 2010
[25] *e107 website system*, available at http://wiki.e107.org/index.php?title=Category:Development
[26] F. Fullone, F. Trucchia, eZ Publish 4: *Enterprise Web Sites Step-by-Step*, Packt Publishing, 2009
[27] C. Lanham, J. Kennard, *Mastering Joomla! 1.5 Extension and Framework Development*, Packt Publishing, 2010
[28] A. S. John, *MODx 2.0 Web Development*, Packt Publishing, 2011
[29] R. Cadenhead, *Movable Type 3.0 Bible Desktop Edition*, John Wiley & Sons, Inc, 2004
[30] D. Liliedahl, *OpenCms 7 Development*, Packt Publishing, 2008
[31] J. Meloni, *Plone Fast Track The basics of building a content-management system with Plone*, Sams Publishing, 2004
[32] P. Krenn, *SilverStripe 2.4 Module Extension, Themes, and Widgets: Beginner's Guide*, Packt Publishing, 2011
[33] K. Potts, R. Sable, N. Smith, C. Lindley, M. Fredborg, *Textpattern Solutions: PHP-Based Content Management Made Easy*, Friends of ED, 2007
[34] *Documentation for Tiki Wiki CMS Groupware*, available at http://doc.tiki.org/Documentation
[35] W. Altmann, R. Fritz, D. Hinderink, *TYPO3 Enterprise Content Management*, Packt Publishing, 2005
[36] N. Wahlberg, P. Sterling, N. Hartvig, *Umbraco User's Guide*, wrox, 2011
[37] A. Brazell, *WordPress Bible*, Wiley Publishing, Inc, 2010
[38] S. Ruoyu, *Designing for XOOPS: A Designer's Quickstart Guide to Content Management*, O'Reilly, 2011
[39] M. Amor, L. Fuentes, Malaca: A component and aspect-oriented agent architecture, *Information and Software Technology* 51 (2009)
[40] G. Jung, J. Hatcliff, A type-centric framework for specifying heterogeneous, large-scale, component-oriented, architectures, *Science of Computer Programming* 75 (2010)
[41] J.S. Lee, D.H. Bae, An aspect-oriented framework for developing component-based software with the collaboration-based architectural style, *Information and Software Technology* 46 (2004)
[42] C.H. Chang, C.W. Lu, P. A. Hsiung, Pattern-based framework for modularized software development and evolution robustness, *Information and Software Technology* 53 (2011)