Decentralized Principles: New Modular Software Development Principles, a Robust Object Oriented Approach

GholamAli Nejad HajAli Irani
University of Bonab
Velayat Avenue, East Azerbaijan,
Bonab 5551761167, Iran

ABSTRACT

Modularity as an object oriented principle helps to develop appropriate large-scale and complex software. But modularity has some deficits [14] such as modular decomposition etc., which is not allowed widely using modularity in software development in current years. In this paper some principles have been provided for increase modularity of software systems and help for turn an existing system to a modular system. These principles distribute functionalities of each module to them and decrease dependency of modules.

To obtain this aim, object oriented principles and heuristics has been analyzed then by considering a module as an object, new modular principles have been provided. In the reminder to evaluate new principles, a new modular architecture has been provided. The strength of new principles has been shown with two complete case studies.

New principles can be used in any large-scale software architectures, modular architectures and any service oriented platforms.

Keywords

Modular Software Architecture, Quality Attributes, Object Oriented Analysis and Design.

1. INTRODUCTION

As increasing the scale and complexity of software systems, several types of approaches and architectures are proposed to overcome the complexity of there. Centralized approach is one of the proposed approaches. In this approach, the most common tasks of system have been extracted and assigned to the common part of system as named Core. Architectures consist of several components [3]. The main purpose of centralized approaches is to reduce the complexity of developing components. So components are developed quickly and many quality attributes of system is improved. In centralized approaches other components are highly depend on Core and if we change the Core, other components has changed as well. So extensibility, modifiability and flexibility of software are increased.

Modularity as an object oriented principle helps to have extensible, modifiable and flexible software. To reach perfect modular software we must have maximum amount of cohesion and minimum amount of coupling [14]. Therefore independent parts of software are excellent.

Based on [14], the biggest problem in modular development is decomposability problem, means that we can't always decompose each system to parts that are independent from each other [14]. So the big trade-off has been made between being modular and modular decomposition.

Most software has used centralized approaches. 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], so we can't reach to modularity and written modules for a system can't be used in other systems.

For example, more than 1200 web portal and CMS is presented in web applications [2]. The CMS is composed of several modules [4]. For example Drupal is composed of more than 8700 modules [5]. So, each module has been written thousands times in general. These problems are due to lack of modularity.

In this paper, to reach a highest modularity between modules, a new modular principles based on decentralized approach have been provided and use object-oriented principles and heuristics, and distributes the Core complexity between other modules.

Steps of developing and evaluating new principles have been defined as following:

1. To examine incompetence of previous approaches and categorized requirement list.
2. To determine new principles of Modular Software Development.
3. To develop a new architecture and describe architecture modules, to prove the benefits of new principles.
4. To provide steps for developing new modules based on provided principles and describe how to use them.
5. To provide complete case studies based on new principles and develop new ideas.
6. To evaluate the new provided principles based on gathered requirement list.

2. PROBLEMS OF CENTRALIZED APPROACH

Being centralized in comparison with decentralized and modularized can cause numerous problems in development software which categorized as fallowing:

Req1: Modules have to use Core Functionalities (CF) – the common functionalities that have been centralized and 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 CF. Consequently; complexity of developing small-scale modules will be increased.

Req3: The implemented CF is not complete in general. Probably, developing large-scale modules is needed to use a new CF 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 CF types and patterns, 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 interface. Therefore system integrity and modules portability decrease.

Req7: In centralized approach, the overall CP (so-called Big Picture) is apparent to all modules. So, encapsulation of CP 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 the reminder, 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		x		P: Portability;
Req7			x		x			T: Testability.

3. NEW MODULAR DEVELOPMENT PRINCIPLES

Based on table 1, theses problems are related to Modularity and in detail related to interoperability, portability, testability, extendibility and modifiability. In this section, to solving these problems, we used Object Oriented principles and heuristics in [1].

Based on [1], we can consider a module as an object then we can apply object oriented principles to obtain new 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 principles and heuristics and their relation with modularity concepts, we can provide new principles for developing modular systems which is shown in table 2.

Table 2. Mapping Object Oriented heuristics to Modular principles.

Heuristics	Code	Provided Principle
H2.1, H5.3	M1	Each module should hold and manage its own data by itself and Modules can't access each other data.
H2.9, H2.10	M2	Each module should perform all its functionalities by itself and Modules can't access each other functionalities.
H2.2, H2.3, H2.4, H2.5, H2.6, H4.1, H4.2, H4.3	M3	Minimize module relationships and interface. Independent modules are excellent.
H3.1, H3.2, H3.7, H3.8	M4	Beware of creation of God module [1]. (Minimize the Core).

4. EVALUATING PROVIDED PRINCIPLES

For evaluating and testing the provided principles which are shown in table 2, we proposed an architecture which was named MEDA. MEDA which is shown in figure 1 has been provided based on modular principles in table 2. In this architecture to reach maximum quantity of modularity and based on M2, each module has to perform all its functionality by itself and Core just run final instructions that have been sent by each module.

To reach maximum quantity of extendibility and modifiability, we need a dynamic structure and dynamic channel to communicate between modules. So, eXtensible Markup Language (XML) technologies [12] have been used to have dynamic structure and communication protocol between modules. To have dynamic channel, a combination of Event Driven Architecture [11] and a simple type of Bus Architecture [13] has been used. Event-Bus module in Core does these functionalities.

To support M1 and M4 from table 1, CAL (Content Access Layer) as a content layer has been placed in MEDA. CAL must execute instructions related to content access. Instruction is sent from modules to Event-Bus and then from Event-Bus to CAL. Contents can be Data, File or Web Services. In CAL, we need modular content access that modules can't access content of each other. For example in modular file access we must use operation system APIs that modules can't access files of each other. In modular data access we need assign a specific database username and password to each module that can't access table of each other.

Module Installer Layer must install and hold each modules and their profile. RTL (Run Time Layer) must do all runtime actions except content access instruction. Exception Manager must control and manage all exceptions that are sent from modules. AA Manager do authentication of whole system and based on M2, M4 authorization of system delivered to each module. Module Presentation must control presentation actions. Theme changing and language changing are done by Module Presentation. For example when a user changes the theme of CMS, theme changing as an event is sent to all modules by Module Presentation. Systems Decoration must manage template of CMS and locate modules presentation part in considered place.

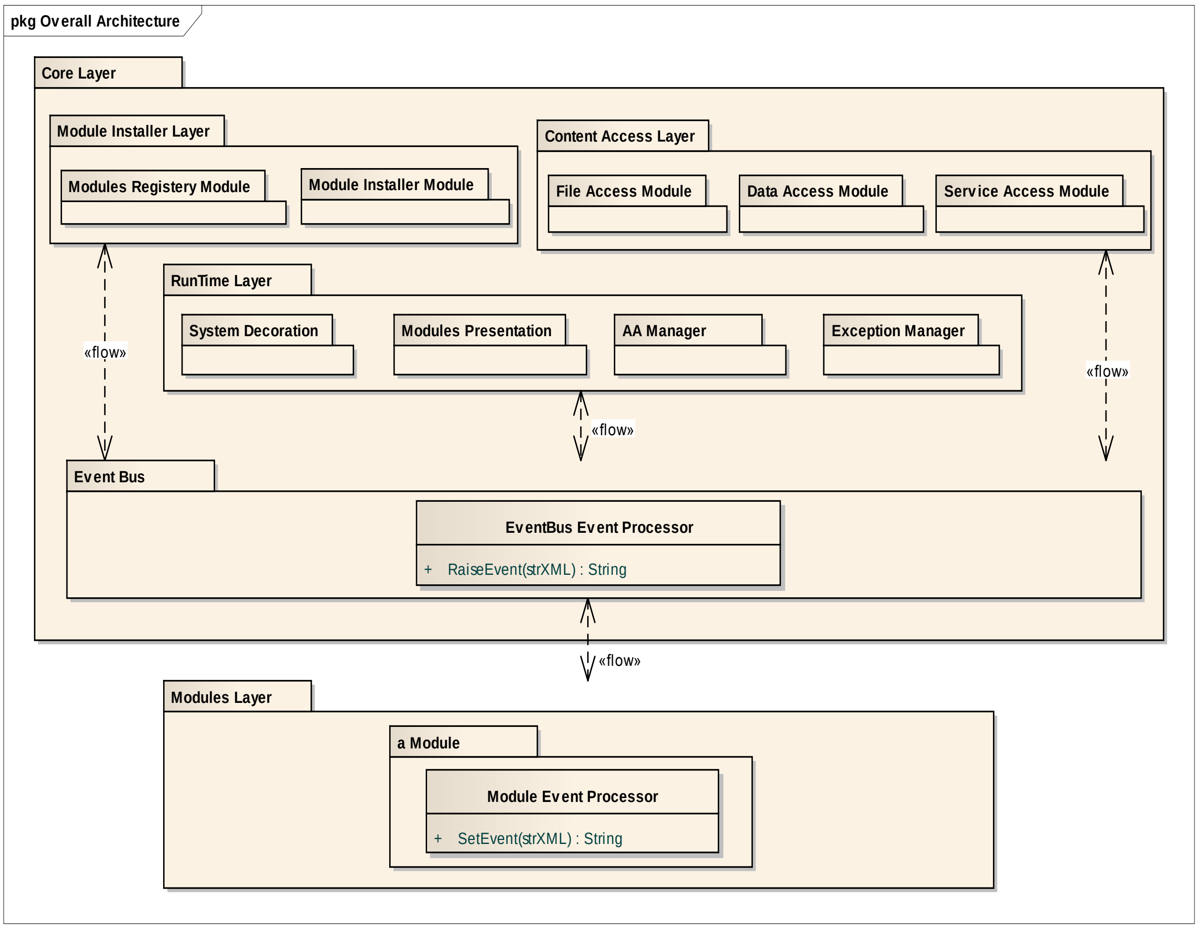

Figure 1: MEDA, provided architecture for evaluating new principles.

5. OVERALL SEQUENCE OF EVENT RAISING IN MEDA

In MEDA, each module and Core as a module, have to implement a class by the name of Event-Processor and use Event-Bus 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 Event-Bus for sending events and Event-Bus uses SetEvent method from Event-Processor (implemented in each module) for sending delivered events to target modules.

Interaction between Core and modules are prepared by two types of 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 usually contains event-type, event-name, input parameters names and values, return type and value, event-sender, event-receiver(s), etc. But while developing each module of MEDA, we must define standard DTD for module event list.

Each module for sending an event must put it in the form of DTD1 and invoke RaiseEvent method from Event-Bus.

Then Event-Bus analyze delivered event and in order to sending event to target modules, use the SetEvent method from Event-Processor 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. Finally, Event-Bus put all of received DTD2s from each module in the <return> tag of DTD1 and passes the final DTD1 as return value of RaiseEvent.

6. STEPS FOR DEVELOPING MODULES

In this section, steps of developing modules based on provided principles are presented as following:

Step 1: Considering that the modular principles presented in table 2 is very general, we must derive detailed modular principles for desired module based on modular principles in table 2.

Step 2: To examine previous methods and approaches in modular systems for the module.

Step 3: To analyze and design of the module based on modular principles and decentralized approach. Use Case Model is necessary, because event list in following steps obtained from Use Case list.

Step 4: Considering M3, we must provide optimized event list of the module based on Use Case list. Event list must cover all use cases of module.

Step 5: To standardize DTD and XML Schema for events of the module.

7. CASE STUDY 1: MODULAR AUTHENTICATION AND AUTHORIZATION

In this section steps of developing modular authentication and authorization (AA) has been provided. In step 1, detailed module principles for AA has provided in table 3.

Table 3. Derived principles for developing AA.

Base Principle	Obtained 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.

In step 2, we must collect and examine previous methods for AA. Previous studies of AA used a centralize approach, hence all AA data and its implementation is performed by Core [15]. However some studies distributed AA data into modules, Core is controlling and deciding about AA [16]. This centralized thinking has some inadequacy which will be discussed in the reminder.

For authenticating, some systems use a standard account management such as LDAP, SSO, NTLM, OpenID, OpenSSO and Site Minder etc [17], [18], [15], [16].

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).

For authorizing, various approaches are provided. Role-Based Access Control (RBAC) is the common method for controlling user access to system resources and actions [19]. Some of the approached, for special uses, separated authentication from authorization [20]. These methods used federated user administration, therefore authentication performs in user's home system and authorization performs in service provider system [21]. To support extensibility and modifiability, AA methods used some new tools like Aspect Oriented Programming Languages [22]. Open Service Gateway initiative (OSGi) framework uses java standard called Java Authentication and Authorization Service which is a centralized approach [23].

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 [24].

In step 3, we must perform analysis and design step for AA based on derived principles in table 3. Distilled use case model is shown in figure 2.

In step 4, we must provide optimized event list of AA based on its Use Case list. Optimized event list for AA is shown in table 4.

In step 5, we must standardize DTD and XML Schema for events of the module. Regard as this step outputs is large, an instance of DTD1 and DTD2 is shown in figure 3.

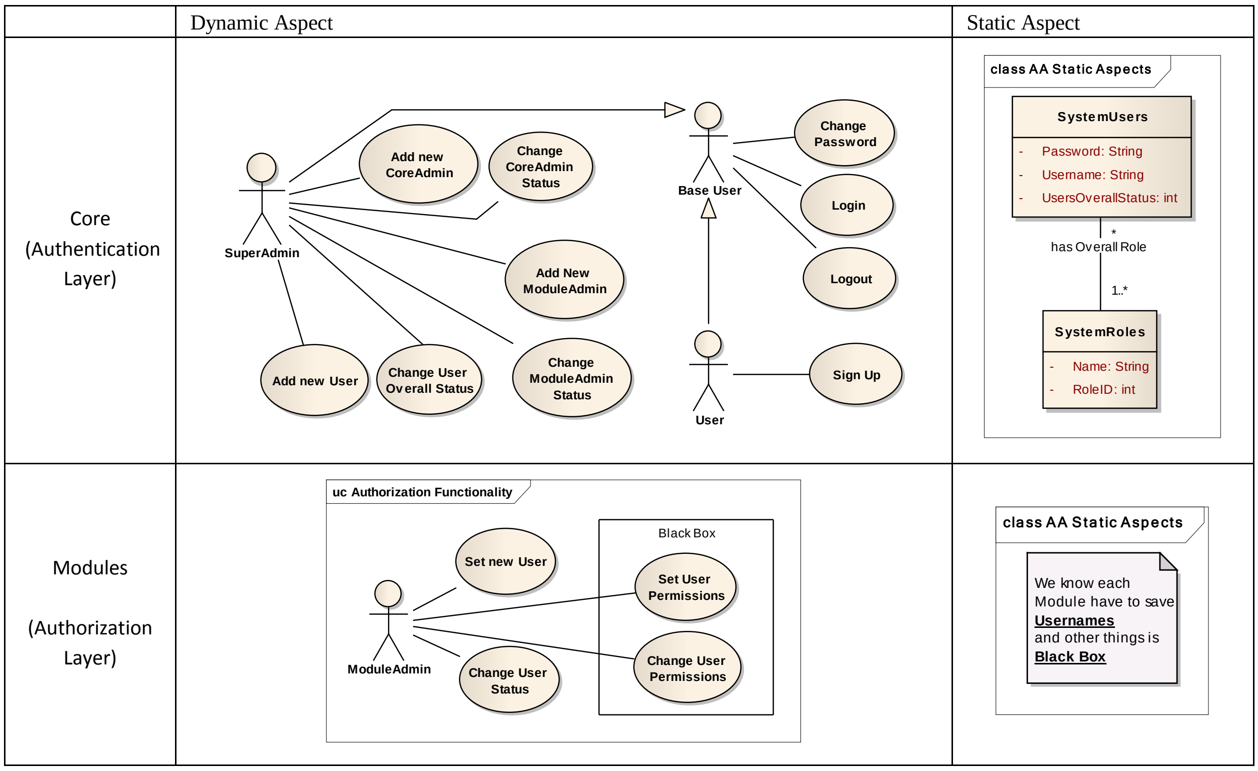

Figure 2: Distilled AA analysis and design.

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.

```
<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>
```

```
<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>
```

Figure 3: (a) An instance of DTD1; (b) An instance of DTD2.

8. CASE STUDY 2: MODULAR DATA ACCESS ARCHITECTURE

In this section distilled steps of developing modular Data Access (DA) has been provided. In step 1, detailed module Software development principles for Data Access (DA) has provided in table 5.

Table 5. Derived principles for developing DA.

Code	Principle Code	Principle
P1	M4, M1	Each module has to manage its Data accessibility within itself.
P2	M4, M2	Each module has to perform its Data Management by itself.
P3	M3	Standardize a DA interface between Core and Modules and between modules.

In step 2, we must collect and examine previous methods for DA with decentralized approach.

DA methods in software architectures called Data Access Patterns (DAP). DAPs are architectures or patterns that manage the data access layer functionalities in Information Systems. Nowadays more than 50 DAPs have been developed [25]. Based on developed patterns from [25], [26] and etc, we categorized all existing DAPs in six groups which are shown in table 6.

There are not any modular data access patterns in provided patterns and they can't be used in our architecture.

Table 6. Provided category for exist DAPs.

Code	Description
DAP0	These methods don't use any data access layer and any parts of code directly connect to database. Due to high performance, these methods mostly are used in real time systems.
DAP1	These methods use one or more classes in data access layer to encapsulate database access details.
DAP2	These methods use DAP0. Also one entity class is created for each table in database and all CRUD (Create, Retrieve, Update and Delete) methods on this table are handled by its entity class.
DAP3	These methods are similar to DAP2 and entity classes are created for each table, but based on object oriented heuristics, all CRUD methods implemented in one class and other entity classes inherit from it.
DAP3M	These methods like DAP3, but metadata of all tables are stored in database as well. All entity classes inherit from Base Class. It create SQL command dynamically and manage them using stored metadata.
DAP4	In these methods unlike DAP3 and DAP2, entity classes for each table are not created. To gain excellent extendibility and modifiability (like DAP3M), metadata of all tables is stored in database and one or more classes perform all CRUD methods for all tables.

Persistence Frameworks (PF) are similar to DAPs and they can be used as a communication layer between applications and database. In [27] and [28] some of PFs are listed. In [29] other types of PFs and evaluation of them was provided as well. For example Hibernate is a PF for Java. PFs and DAPs use similar approaches to communicate with database. All provided PFs are not modular as well.

Other Web Frameworks and Portals like SAP [17], Liferay [15], Zend [16], Alfresco [18] and Drupal [8] use a provided PFs or DAPs and none of them are modular. In [6], [7] and [9] new DAPs have been developed, but none of them are modular as well.

Another approach is Modular Database. In [10] modular databases have been suggested as a challenge and many aspects of it have been investigated. Afterwards, in [31] and [30] new architectures for modular database have been provided, but don't support all aspect of modularity and extendibility like event-driven. In next sections we will show that modular database can't be a modular data access layer and it is not a panacea for our problem.

In step 3, we must perform analysis and design step for modular DA based on derived principles in table 5. Distilled use case model is shown in figure 4.

In step 4, we must provide optimized event list of modular DA based on its Use Case list. Optimized event list for DA is shown in table 7.

Adding a new event will not affect DA in any way. For example, in order to gather Content Search, we can add an event such as: +getContentSearch(String ContentInfo):String; (from Core.DA to All Modules) into event list and then all modules can response to this event and Core after get all results from modules, complete final result and return it.

In step 5, we must standardize DTD and XML Schema for events of the module. Regard as this step outputs is large, an instance of DTD1 and DTD2 is shown in figure 5.

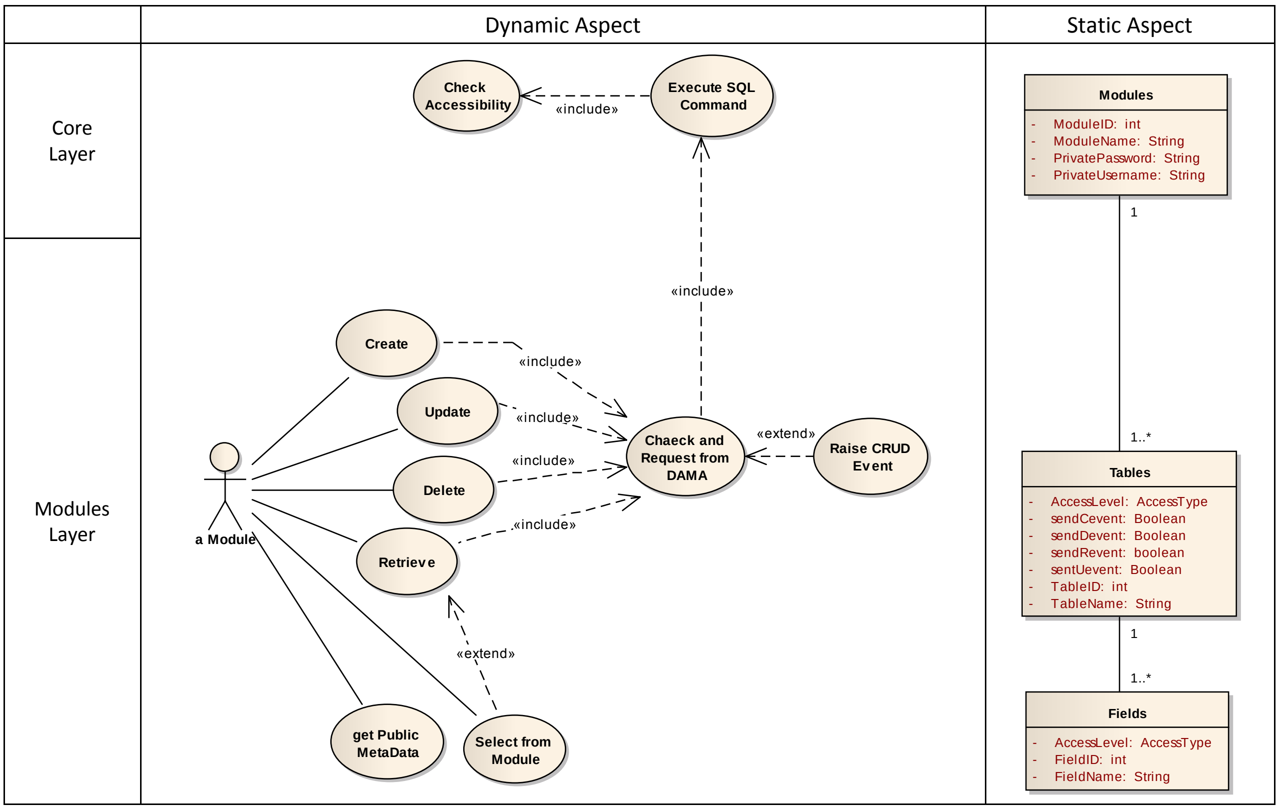

Figure 4. DA distilled analysis and design.

Table 7. Optimized Event List For DA.

Use cases	Events	from	to	Description
Create, Update, Delete	ExecuteSQL (String strSQL):String;	A Module	DA	When a module wants to perform a SQL command from its tables by DA, should use this event. As the type of Create, Update and Delete are different from Retrieve; they were categorized in different event.
Retrieve	ExecuteRetrieveSQL(String strSQL):String;	A Module	DA	When a module wants to Retrieve from its tables by DA, should use this event.
Select from a Module	RetrieveRequest(String strSQL):String;	A Module	Another Module	When a module wants to select from another modules tables, should use this event. This event performs after Get Public MetaDate event.
Get Public MetaData	getPublicMetaData(): String;	A Module	Another Module	Any modules must to response to this event that is sent from other modules and return its public access tables.
Raise CRUD Event	CRUDEvent(String CRUDInfo):void;	A Module	All Modules	Each module can raise a CRUD notice event after CRUD performs. With this event, modules can inform other modules.
Other Use Cases	---			For other cases, modules act independently or don't need to any event.

```
<Event  EventType="DAMA-Select"
EventName="SelectNews" EventID="123" SenderName="A
Module" SenderID="379" ReceiverNames=" B Module"
ReceiverIDs="12"  RaiseDateTime=" " Description=" ">
          <InputFields>
                    <Field Name="SQLCommand"
          Value="Select * from News where
          NewsNO<14"/>
          </InputFields>
          <Return  Type="String">
                    <!--  return info must be in here  in
          DTD2  format -->
          </Return>
</Event>
```

```
<ReturnObjects  ModuleName="B Module" ModuleID="123"
ReturnDateTime=""  Name="NewsRow" Description="">
<Object>
          <Field Name="NewsNO" Value="10"/>
          <Field Name="NewsTitle" Value="news title1"/>
          <Field Name="NewsBody" Value="news body1"/>
</Object>
<Object>
          <Field Name="NewsID" Value="11"/>
          <Field Name="NewsTitle" Value="news title2"/>
          <Field Name="NewsBody" Value="news body2"/>
</Object>
</ReturnObjects>
```

Figure 5. (a) An instance of DTD1.   (b) An instance of DTD2.

9. EVALUATION
In section 2, we categorized a requirement list as problems of centralized approaches. Decentralized principles capture all of these requirements which are shown in table 1. In fact, decentralized principles improve all quality attributes which are mentioned in table 8.

Table 8. Requirement list is captured by MEDA.

Description	**Captured Requirements**
Modules are independent in selecting their own patterns. They just have to consider Core's standard interface.	Req1, Req2, Req3
Since modules are not dependent to Core, unit test of each module can perform easily far from the Core.	Req4
Modules are free to choose their patterns and we don't need to collect all patterns in Core.	Req5
As establishing a new standard interface for Core and modules communication, integrity and portability of modules was increased and modularity of each module was increased to higher degree.	Req6, Req7

10. CONCLUSION
In this paper, new decentralized principles for modular systems have been provided. These principles distribute Core functionalities between modules based on robust object oriented thinking. So, dependencies between modules decrease saliently and existing systems turn to more modular systems. Therefore 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.

These principles can be use in Service Oriented Platforms and any large-scale modular software. In addition, we can use these principles to any aspect of software. For example modular authentication and authorization and modular data access have provided as case studied. Modular exception handling, modular service access, modular file access and etc, can be investigate as future works of this paper.

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

[2] The Content Management Comparision Tool, available at http://www.cmsmatrix.org

[3] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, 2000.

[4] B. Boiko, Content Management Bible, 2nd Edition, Wiley Publishing, Inc., Indianapolis, Indiana, 2005.

[5] Drupal, Open Source CMS, available at http://Drupal.org/Project/Modules

[6] A. Neto, H. Fernandes, D. Alves, D.F. Valc´arcel, B.B. Carvalho, J. Ferreira, et al, A standard data access layer for fusion devices R&D programs, Fusion Engineering and Design 82 (2007) 1315–1320.

[7] S. Fiore, A. Negro, G. Aloisio, The data access layer in the GRelC system architecture, Future Generation Computer Systems 27 (2011) 334–340.

[8] M. Butcher, G. Dunlap, M. Farina, L. Garfield, K. Rickard, J. Albin Wilkins, Drupal 7 Module Development, Packt Publishing, 2010.

[9] G. Manduchi, A. Luchetta, C. Taliercio, T. Fredian, J. Stillerman, Real-time data access layer for MDSplus, Fusion Engineering and Design 83 (2008) 312–316.

[10] W. Mahnke, H. Steiert, Modularity in ORDBMSs – A new Challenge, Tagungsband 13. Workshop, Grundlagen von Datenbanken, GI-FG 2.5.1, Magdeburg, Juni 2001.

[11] O. Etzion, P. Niblett, Event Processing in Action, Manning Publications, USA, 2011.

[12] R. Bourret, A. B. Coates, B. Harvey, G. K. Holman, M. Kay and et al, Advanced XML Applications from the Experts at The XML Guild, Thomson Learning Inc, 2007.

[13] D. Chappell, Enterprise Service Bus, O'Reilly Media, Inc, 2004.

[14] B. Meyer, Object Oriented Software Construction, Second Edition, Prentice Hall, 1997.

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

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

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

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

[19] 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.

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

[21] 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.

[22] 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.

[23] R. S. Hall, K. Pauls, S. McCulloch, D. Savage, OSGi in Action, Manning Publications, 2011.

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

[25] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. Patterns of Enterprise Application Architecture, Addison Wesley, 2002.

[26] C. Nock. Data Access Patterns: Database Interactions in Object-Oriented Applications, Addison Wesley, 2003.

[27] Java Persistence Layer Source Codes, available online at: http://Java-source.net/persistence.

[28] C# Persistence Layer Source Codes, available online at: http://Csharp-source.net/persistence.

[29] R.Barcia, G.Hambrick, K.Brown, R.Peterson, K.S.Bhogal, Persistence in the Enterprise: A Guide to Persistence Technologies, IBM Press, 2008.

[30] F. Irmert, M. Daum, K. Meyer-Wegener, A New Approach to Modular Database Systems, SETMDM '08 Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management ACM New York, NY, USA ©2008.

[31] M. Mammarella, S. Hovsepian, E. Kohler, Modular Data Storage with Anvil, SOSP'09, October 11–14, 2009, Big Sky, Montana, USA.