J. Basic. Appl. Sci. Res., 1(12)2835-2842, 2011

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

DAMA: A New Data Access Architecture for Modular Information Systems, a Robust Object Oriented Approach

[1]GholamAli Nejad HajAli Irani* and [1]Jamal Dabbagh Yarishah

[1] Bonab University, Bonab, Iran

ABSTRACT

The All Information Systems (IS) need a data access (DA) layer for communicating and sending data between modules and Database. Existing Enterprise Systems, Frameworks and Portals use a similar centralized approach. In this paper, a new decentralized architecture has been provided for DA in Modular Information Systems. To obtain this aim, firstly, all existing approaches and their disadvantages for DA, has been investigated and categorized. Secondly, to obtain a new architecture and solving disadvantages, new modular DA principles has been developed using robust object oriented principles and heuristics. New architecture has been developed based on these obtained modular DA principles. In addition, the new architecture decreases the Core complexity and distributes that into modules. Also due to use of an event-driven architecture and extensible markup language (XML) as communication protocols, new architecture has maximum quality of reusability and extensibility and modifiability that can be applied to all Modular Information Systems.

KEY WORDS: Modular Software Architecture, Object Oriented Analysis and Design, Data Modelling.

INTRODUCTION

As the scale and complexity of software systems are increasing, new approaches are developed to overcome these complexities. System Modularizing is one of mentioned approaches. Several architectures have been provided for modular development in [1-2]. Our suggested architecture for modular development 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" 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. To support maximum extensibility and modifiability for communication between Core and modules, we used an event-driven architecture [3]. To support various implementation platforms and maximum compatibility we used XML as communication protocol [4].

Indicated architecture is consisted of three main parts: Content Access Layer (CAL), Module Installer Layer (MIL) and Runtime Layer (RTL). MIL installs modules, holds their profile and uninstalls them. RTL controls modules at runtime. Exception Manager manages all system exceptions and AA Manager (AAM) provides Authentication and Authorization services for whole system. CAL controls modules to access their contents. Content consisted of Database, Files and Web Services. Data Access Modular Architecture (DAMA) is part of CAL. In addition to be a data access layer, DAMA must support modular systems that can be used in all modular architectures. Moreover DAMA must have maximum extendibility as adding new modules can performs easily even at runtime. Finally, DAMA must use all-known protocols that any platforms can use it.

The aim of this paper is to provide a new architecture for DAMA which can be applied as a standard in Modular IS. The process we suggest for developing DAMA should cover this followings:

- To collect previous methods in modular systems and examine incompetence of them.
- To gather and categorize a requirement list.
- Determining new principles of Modular Architecture Design based on Object Oriented principles.
- Developing a new architecture and standardize communications between architectural elements.
- Designing new architecture from static and dynamic point of view.

***Corresponding Author:** GholamAli Nejad HajAli Irani, Bonab University, Bonab, Iran.
Email: Irani@Bonabetu.ac.ir

- Evaluating the new provided architecture in comparison with previous methods

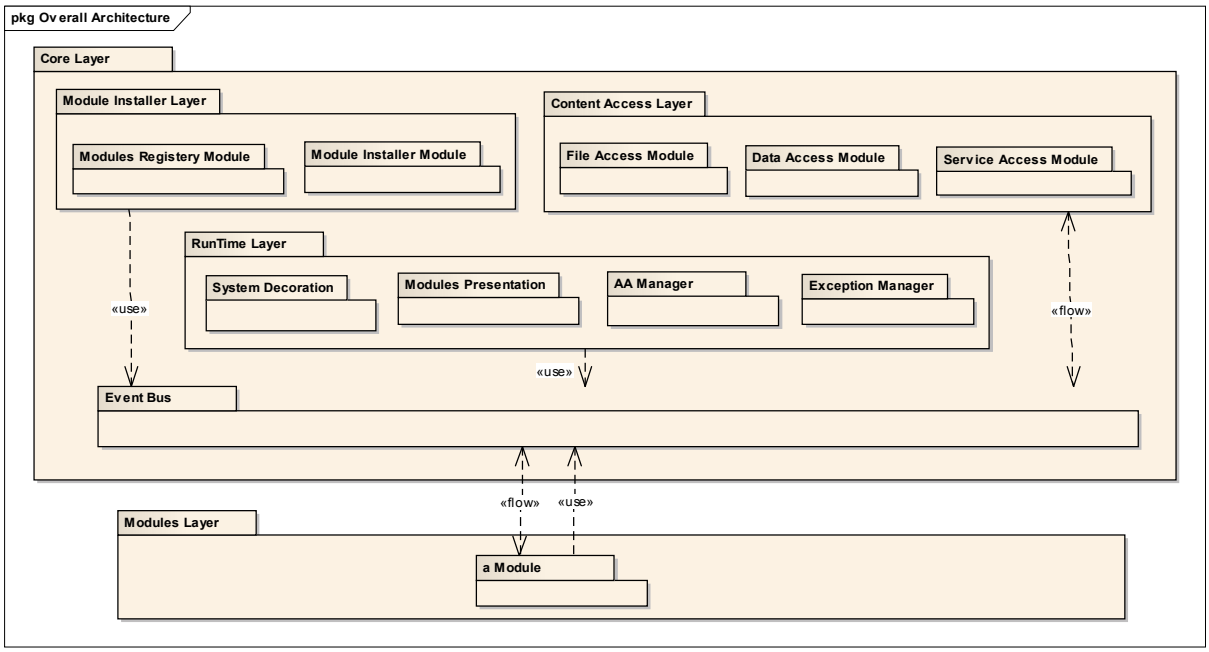

Fig. 1: Suggested architecture for Modular IS.

- **Investigating previous approaches**

As the scale and complexity of systems is increasing, several methods have been provided to conquest their complexity. All provided methods have used a centralize approach to implement DA. This centralized thinking has some inadequacy which will be discussed in the reminder.

Data Access Patterns (DAP) are architectures or patterns that manage the data access layer in IS. Nowadays more than 50 DAPs have been developed [5]. Based on developed patterns from [5-6] and etc, we categorized all DAPs in six group which are shown in table I. There are not any modular data access patterns in provided patterns and they can't be used in our architecture.

Persistence Frameworks (PF) are similar to DAPs and they can be used as a communication layer between applications and database. In [7] and [8] some of PFs are listed. In [9] 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 [10-14], use a provided PFs or DAPs and none of them are modular. In [15-17] new DAPs have been developed, but none of them are modular as well.

Another approach is Modular Database. In [18] modular databases have been suggested as a challenge and many aspects of it have been investigated. Afterwards, in [19] and [20] new architectures for modular database have been provided, but this approached 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.

Table I: 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.

- **Problems of existing approaches**

As the scale of the systems is increasing and their architectures used a centralized approach, the scale and complexity of Core is growing as well. One of the important advantages of being centralized is that DA 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 DAP 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 DAP of the Core. Consequently, complexity of developing small-scale modules will be increased.

Req3: Surely, the implemented DAP of the Core is not complete in general. Probably, developing large-scale modules is needed to use a new DAP 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: Because of centralized approach, integration of implemented modules into different systems with different Cores takes some efforts due to lack of standard DA interface. Therefore system integrity and modules portability decrease.

Req6: In centralized approach, the overall DAP of the Core (so-called Big Picture) is apparent to all modules. So, encapsulation of Core's DAP is violated.

Table II 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. It's due to lack of modular data access pattern or architecture.

Table II: Quality attributes affected by Requirement List.

	E	M	Mo	I	Io	P	T
Req1	x		x			x	
Req2	x		x				
Req3	x	x					
Req4							x
Req5				x		x	
Req6			x		x		

LEGEND

E: Extendibility;
M: Modifiability;
Mo: Modularity;
I: Integrity;
Io: Interoperability;
P: Portability;
T: Testability.

- **DAMA Development Principles**

Problems of existing approaches can be categorized in two groups. Group one is problems with centralized approach and group two is problems with lack of extendibility. Based on these problems, we have provided new modular principles.

To eliminate centralize thinking and provide a new architecture for DA in modular IS, we used Object Oriented principles and heuristics provided in [21]. Based on [21], 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 [21] 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, 4.5, 5.3 and 7.1. By analyzing concepts of above-mentioned heuristics and their relation with modularity concepts, we can provide some advice to develop modular systems which is shown in table III.

We can have other types of advice like inheritance between modules. But they are so hard to implement because there is not any infrastructure and platform to implement it.

In the centralized approaches, Core will become a God module. Data Management of each module doesn't belong to Core functionality. Therefore Data Management should not be centralized and its functionality should be distributed horizontally among modules. According to provided advice in table III, we can obtain some modular DA principles which are shown in table IV.

Table III: Mapping Object Oriented Heuristics to Modular Advice.

Heuristics	Code	Provided Advice
H2.1, H5.3	M1	Each module should 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, H4.4	M3	Optimize and minimize module interface.
H3.1, H3.2, H3.7, H3.8	M4	Beware of creation of God module [1].
H4.5, H7.1	M5	Each module can contain and inherit another module.

Table IV: Object Modular DA Principles

Code	Advice 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.

- **DAMA: a new Data Access Architecture**

Fig. 2 illustrates DAMA architecture based on principles which are shown in table IV. In this architecture, all data access management functionality of each module, granted to itself. Regarding suggested architecture which is shown in Fig. 1, In DAMA, in order to support maximum extensibility and modifiability for communication between Core and modules, we used event-driven architecture [3]. Also, to support various implementation platforms and maximum compatibility we used XML as communication protocol [4].

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 when a module wants to Select from a table, Module will send a request (in the form of an event) to get Select from Core. This action will perform by the use of an event like: +Select(String strSQLCommand):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.

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.

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. Whole of this process that a module wants Select from another module is shown in Fig. 5. In this process a module (AM) raise an event (DTD1AM) and send it to another module (BM) by EventBus and BM raise another event (DTD1BM) to select from Database with DAMA. Finally result returned to AM.

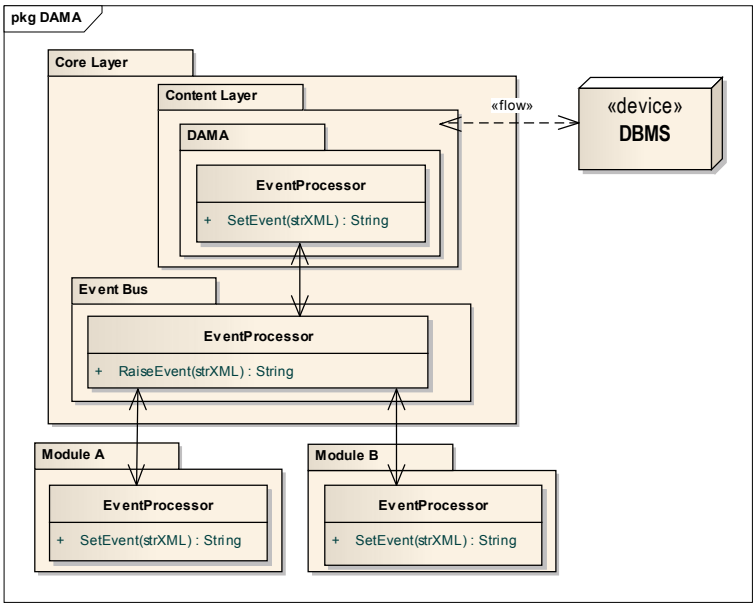

Fig. 2: DAMA architecture for Modular IS.

```
<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>
```

Fig. 3: An instance of DTD1.

```
          <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>
```

Fig. 4: An instance of DTD2.

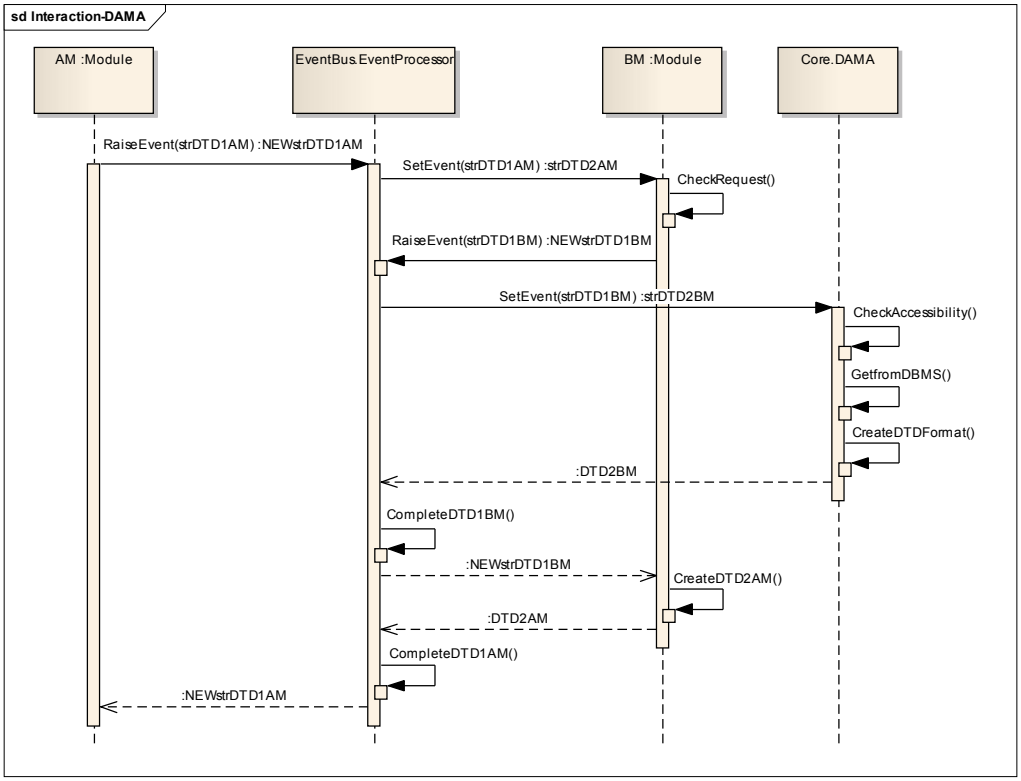

Fig. 5: Process of request and response an event.

1. DAMA Analysis and Design: Data Viewpoint

Each module can perform CRUD methods on its tables. Most parts of CRUD codes perform into modules and for final registration send it to DAMA. Each module has a private username and password for perform CRUD methods on its tables. To access other module tables, a public username and password is exist as well. All modules can access other modules to select from own tables. Therefore each module must set accessibility level to its tables and fields. So each module must response to Get Public Meta Data request that has been sent from other modules. Then modules can retrieve from tables of each other. This method performs by public username and password.
Each module can inform other modules after it performs CRUD methods. For example, when a record of a table is deleted, module can informs other modules from this action by raising an event.

Finally, we can divide all DA use cases and data model into two layers: Core Layer and Module Layer. Fig. 6 is a distilled analysis and design artefact of DAMA.

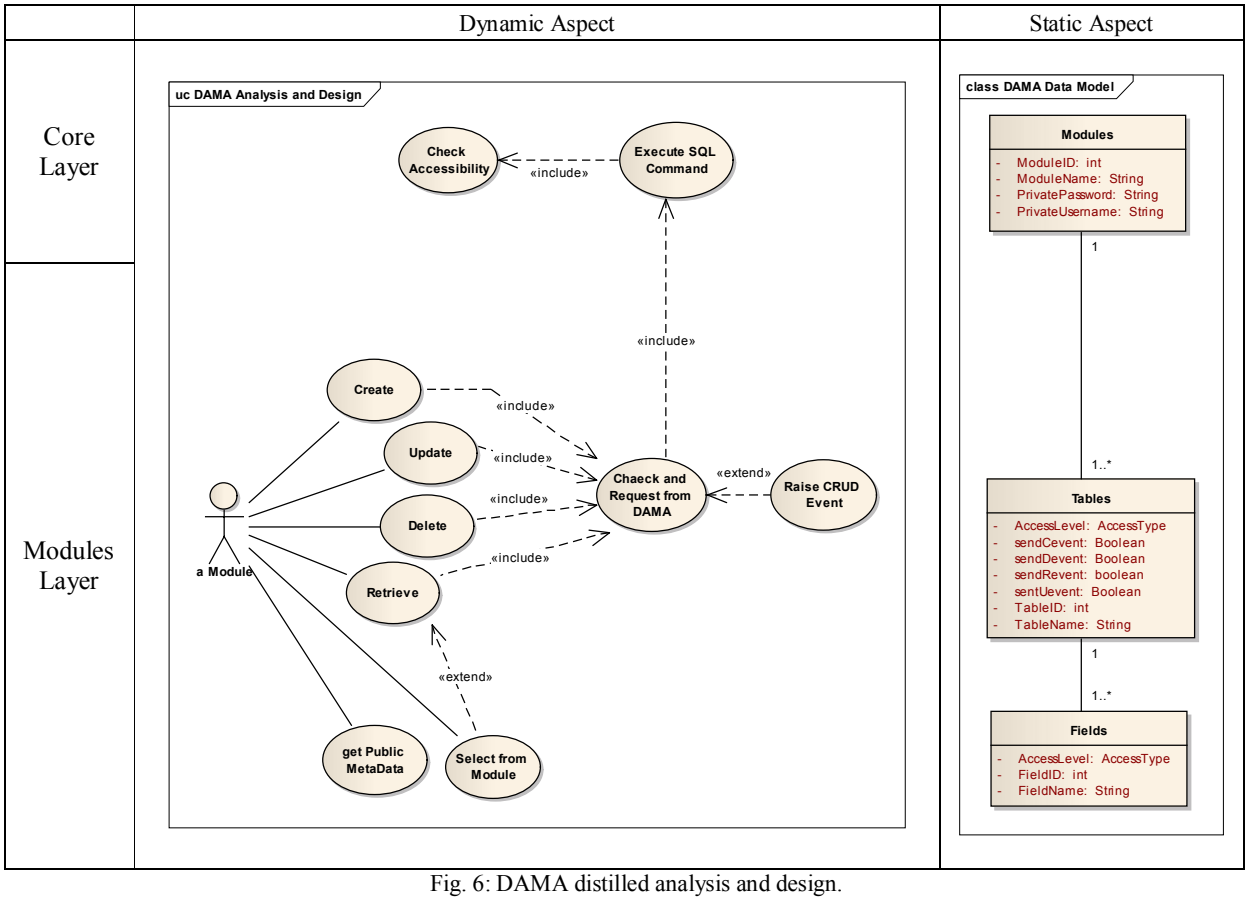

Fig. 6: DAMA distilled analysis and design.

2. Optimized Event List for DAMA

Regarding to Event-Driven Architecture, we should extract an optimized event list for communication between Core and Modules, as it capture all DA use cases. The obtained optimized event list and its descriptions are shown in table V.

Adding a new event will not affect DAMA in any way. For example, in order to gather Content Search, we can add an event such as: +getContentSearch(String ContentInfo):String; (from Core.DAMA 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.

Table V: Optimized Event List For DAMA.

Use cases	Events	from	to	Description
Create, Update, Delete	ExecuteSQL (String strSQL):String;	A Module	DAMA	When a module wants to perform a SQL command from its tables by DAMA, 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	DAMA	When a module wants to Retrieve from its tables by DAMA, 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.

3. Evaluation

In section 2, we categorized a requirement list as problems of existing methods. The provided architecture in this paper (DAMA) captures all of these requirements which are shown in table VI. In fact, DAMA improves all quality attributes which are mentioned in table II.

Table VI: Requirement list is captured by AAMA.

Description	Captured Requirements
Modules are independent in selecting their own DAP. 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
As establishing a new standard interface for Core and modules communication, integrity and portability of modules was increased.	Req5, Req6

4. Conclusion and future works

In this paper, DAMA as new extensible and modifiable data access architecture for modular IS by a decentralized approach has been provided. DAMA can be use in Enterprise IS, Service Oriented Platforms and any large-scale modular software. DAMA used an Event-Driven method, so changes can be applied easier by adding new event in it. DAMA 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 Exception Manager, Modular File Access and Modular Service Access.

REFERENCES

[1] 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.

[2]P. Intapong, S. Settapat, B. Kaewkamnerdpong, T. Achalakul. Modular Web-Based Collaboration Platform, *International Journal of Advanced Science and Technology*, 2010.

[3] O. Etzion, P. Niblett , *Event Processing in Action*, Manning Publications, 2011.

[4] D. Hunter et al. *Beginning XML, 4th Edition*, Wrox Press, 2007.

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

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

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

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

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

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

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

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

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

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

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

[16] 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.

[17] 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.

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

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

[20] 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.

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