BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **17**, No 3
Sofia • 2017 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2017-0033

Modules for Rapid Application Development of Web-Based Information Systems (RADWIS)

S. Cheresharov[1], *H. Krushkov*[1], *S. Stoyanov*[1], *I. Popchev*[2]

[1]*Faculty of Mathematics and Informatics, Plovdiv University "Paisii Hilendarski", 4000 Plovdiv, Bulgaria*
[2]*Faculty of Economics and Social Sciences, University of Plovdiv "Paisii Hilendarski", 4000 Plovdiv, Bulgaria*
E-mails: stoyancheresharov@gmail.com hdk@uni-plovdiv.bg stani@uni-plovdiv.bg ipopchev@iit.bas.bg

***Abstract*:** *This paper describes a model of modular system for Rapid Application Development of Web-based Information Systems* (*RADWIS*). *The existing modular systems on technology, framework and platform level does not fully solve the problems of functionality reuse, rapid application development and balance between the complexity, size and functionality. The proposed modular system addresses these problems in a new way. The current work fills the gap between the modular systems on the framework and platform level. The model uses flexible, reusable modules, which can be built with different technologies. They are installable and shareable with the standard dependency manager of the technology and can communicate using web services. The modules use NoSQL approaches in SQL databases. A workflow engine module, based on the Petri Nets theory, allows a graphical and formal mathematical solution for a wide variety of problems.*

***Keywords*:** *Modules, Web-based information systems, Petri Nets, NoSQL in SQL.*

1. Introduction

The Distributed e-Learning Centre (DeLC) is a project implemented at the Faculty of Mathematics and Informatics at the University of Plovdiv, the aim of which is to build infrastructure and software environment for the provision of e-Learning services and learning content [1, 2]. The environment is continuously expanded with components such as, for example, an intelligent software learning support agent [3], automatic generation of test questions [4, 5], a middleware for mobile services [6], Selbo2 electronic content editor [7].

One of the main shortcomings of DeLC (as well as widely used e-Learning systems) is that they ignore the physical world in which they operate. Reporting on what is happening in the physical environment reveals new opportunities to improve the usability and efficiency of the environment by different user groups. In this sense, trying to improve the environment, in recent years we have been working to transform

DeLC into a new infrastructure called Virtual Learning Space (VLS) that integrates the virtual environment with the physical world. The space is built as an Internet of Things (IoT) ecosystem. At the same time, new challenges are emerging to build this kind of space. New approaches and technological solutions are needed, mainly involving the acquisition, collection and transformation of sensor data from the physical world. For this purpose, appropriate technologies are those that use software components known as modules. Modules are self-sufficient software structures that deliver a certain complete functionality.

This publication presents a generic modular approach for Rapid Application Development of Web-based Information Systems (RADWIS). The rest of the paper is organized as follows. The second section provides a brief overview of the module-oriented approach; the third section presents the model and the applied methods, the forth section describes the results, the fifth section is the discussion, and the last section is the conclusion.

2. Module-oriented approach

The problem we try to solve is how to build as many types of web-based information systems as possible, as fast as possible, in a standard way, without writing code and without compromising the quality, with reusable, shareable, interchangeable modules. The development of Internet of Things (IoT), Artificial Intelligence (AI) Web Services, serverless architectures etc., requires rapid application development without compromising the quality of the system. As the authors of [8-16] have investigated, recreating the very same functionality with different technologies and for different projects is a common problem. Our goal is to write the code and create the functionality once, and use it in as many systems as possible. Sometimes the prebuild systems/platforms, such as Joomla, WordPress, etc. are too complex and big for the small problems we try to solve or does not match the specific requirements. We have to combine and mix, their functionality, which is extremely difficult or not always possible. There also should be a balance between the complexity and functionality of the system.

There are existing solutions for these problems on different levels. The modular, object-oriented and aspect-oriented programming are some of them. Solutions from a different level are the frameworks, libraries and platforms. They encourage the reuse of prebuild elements. These elements can be seen as modules.

Since there are no formal definitions for the terms we are going to use, we have to establish our terminology first. From the documentation of the technologies as Java, JavaScript, C, C^{++}, LISP we can draw a conclusion that the modular programming is a software design technique that encourages the separation of the functionality of a programme into independent reusable, interchangeable modules. Each module contains everything necessary to execute one aspect of the functionality.

The term module is related to the paradigm of the modular programming. Module is a concept which meaning is determined by the context. In the context of a programming language, the modules can be the classes with their structure and functionalities. If we go through all the layers of a software stack, we can find

modularity on each level. The modularity on the lower level – programming language is presented by using classes, objects and methods. The frameworks and platforms are modular systems from other levels.

From the documentation of Java, C++, PHP it is possible to conclude that a software technology is a programming language and the tools, servers, libraries etc. which allow us to create software applications.

A framework is a reusable, "semi-complete" application that can be specialized to produce custom applications [17, 18].

The term platform we are going to use for prebuilt web-based systems like Joomla, WordPress, Umbraco, Alfresco, Magento, phpBB, etc. built to fulfill certain end user requirements. With them can be built different types of web-based information systems such as blogs, portals or Content Management System (CMS), forums, wikis, social networks, Enterprise Resource Planning (ERP) systems, etc. These systems are built on top of their own frameworks, have their own paradigms, installers, plugin systems and philosophy. Some of them (Joomla, WordPress, etc.) gradually became platforms, because the end user without coding skills can build different types of web-based applications. They are not anymore only CMS or blog systems, but multipurpose platforms.

Each technology usually has its own package/dependency manager. It helps to create, share, install packages and takes care of the dependencies between them.

The technologies, frameworks and platforms can be treated as modular systems. The meaning of the term module is different for each one of them and it is determined by the context.

There is no best solution, when it comes to build web-based information system with one of the described modular systems. It all depends on the requirements of the project. In the attempt to solve the problem for rapid application development without compromising the quality and keep the functionality and complexity in balance, libraries, frameworks and complete solutions/platforms have been created. Right now, the best solution is to find the right technology, framework or platform for a given project. With another words we have to find the right for the problem modular system.

The main limitation of the modular systems on technology, framework and platform levels is the fact that we cannot reuse functionality across them and building the system is time consuming or inadequate complex for solving simple problems. Which means that each time we have to write again code to achieve the very same functionality, we already done it, for another technology, framework or platform.

Each of the approaches with using technologies, frameworks and platforms comes with its own pros and cons.

The modular system on technology (programming language) level allows fine control on the low-level elements, speed optimisation, knowing only one programming language, full control over the processes in the application, independence of the work of other developers since the application depends only on the programing language. On another hand, this approach slows down the development, because it requires creating many low-level modules, very good knowledge of the design patterns and best practices, creating documentation because

the structure of the application is not standard, maintenance of the modules, highly skilled and experienced developers. The cost of maintenance and change is very high.

The modular system on framework level has many advantages such as faster building of an application, increasing the quality of the system, because it is built with high quality tested modules, using naming and coding standards enforced by the framework, joining new developers to the project is fast and easy. The cost of change is relatively low.

However, the approach of building an application using a framework is still not fast enough, because for achieving a certain functionality the developers still have to write a lot of code, requires knowledge of the framework, the project becomes dependant of the framework.

The modular system on a platform level is the fastest way for building an application of a certain type. No development skills are required. However, even the smallest change of the functionality can be slow and difficult. The platforms use their own philosophy, naming and coding standards and installation procedures. They are usually open source projects and are usual targets for hacker attacks. To be backword compatible, they use old techniques and bad practices. The modules from one platform cannot be used in another one. The complexity and size of the systems very often are not in balance with the functionality. For very simple functionality are used very complex platforms.

The frameworks and the platforms are built with different technologies and there is no compatibility between their elements. Even in only one technology, there are many not compatible frameworks. With the modular systems on the platform level, the situation is the same. Another cannot use the plugins from one platform. The next time we have to build another system, we go through again the same slow, painful process of finding the best technology, framework or platform that suites best the problem. Therefore, we again have to find solutions to problems already solved in our previous projects using different technologies, frameworks or platforms. However, we are not able to reuse the same functionality already built for another project.

What we try to achieve is to create a modular system from another level (RADWIS Modules, Fig. 1), that combines the advantages of the modular systems on technology, framework, platform levels, and allows the problems solved once to be reused everywhere. Our goal is to create a model of a modular system that allows Rapid Application Development of Web-based Information Systems (RADWIS) with high quality and reliability. The functionality should be always in balance with the size and complexity of the system. The modules should be able to use web services to communicate between each other and external systems. The modules have to be built on such way allowing maximum flexibility and independence from the database schema. By adding a module that offers mathematical formalism for modelling parallel processes, we can further extend the abilities of the modular system for creating applications, by using formal mathematical and graphical approach. Such module adds another abstraction layer, allowing building a web system to be done by creating its formal mathematical model. We are going to use

Petri Nets theory, because of its wide spread and popularity, to build a Workflow Management System (WMS) Module (Workflow Engine).

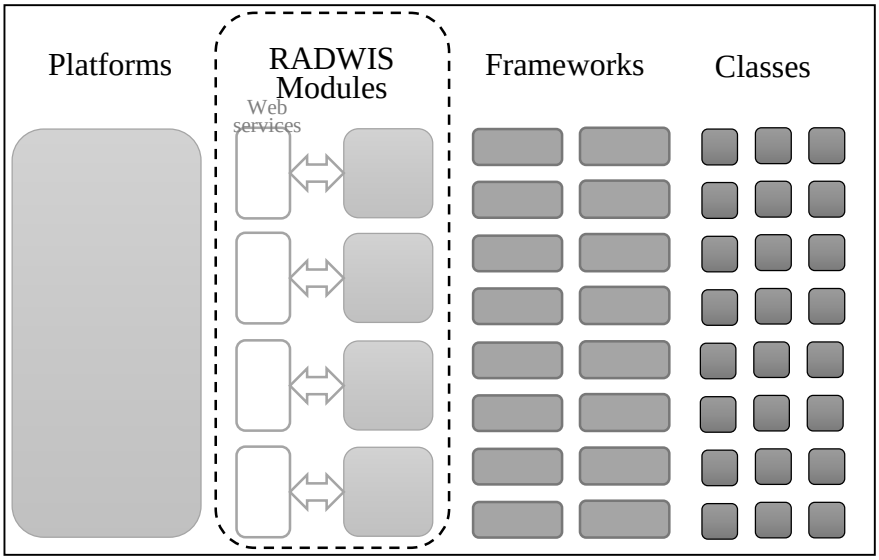

Fig. 1. RADWIS Modules: Fill the gap between platforms and frameworks

3. Related work

Related work has been done in the same direction in another domain. There are similar solutions for different types of applications. One of them is "Reactive Blocks". It uses modules called blocks for building IoT applications. They try to solve the very same problems for reuse, rapid application development, quality, easy installation, deployment and balance between the complexity and functionality. Reactive Blocks is a plugin for Eclipse, which allows applications to be created systematically from building blocks. The building blocks can encapsulate Java code and have special interface descriptions, which goes beyond traditional interfaces. Therefore, blocks can be easily shared with other developers and be reused easily. In principle, everything that runs in Java can be encapsulated in a building block. What is surprisingly difficult in program code is simple to express by graphics [19].

Reactive simply means to react appropriately to events, keeping an eye on time, and handling events as parallel as possible, but in an ordered way. It is difficult to develop reactive systems right with pure programming only [20].

To enable producing more and more software, the most economical way is to reuse as much of the existing software as possible. However, reusing existing software often has challenges of its own [21].

Reactive Blocks is a visual model-driven development environment supporting formal model analysis, automated code generation, hierarchical modelling, and an extensive library of ready-to-use components for the Java platform. By combining re-usable blocks, a developer can create complex applications graphically. This

supports the quick and intuitive understanding and collaboration [22]. There is a market place where the developers can collaborate and share blocks.

Reactive Blocks project relates to our work, because it is a modular system, that shares the same ideas, methods and approaches like ours, for different type of applications. It allows building application using installable, reusable, shareable modules and uses graphical tools for development.

Another effort for modular programming is made in NetBeans platform as described in [23]. The author stated that, nobody writes software entirely in-house anymore. Outside the world of embedded systems, almost everyone relies upon libraries and frameworks written by someone else. By using them, it is possible to concentrate on the actual logic of the application while reusing the infrastructure, frameworks, and libraries written and provided by others. Doing so shortens the time needed to develop software.

As investigated in [23] the rise of open-source software over the past decade makes library reuse doubly compelling. For many kinds of programs, there are existing solutions for various problems, and those solutions are available at zero monetary cost.

The other thing to realize is that no one fully controls the schedule of the whole product. Not only the source code, but also the developers are spread all over the world, working on their own schedules. Such a situation is not actually as unusual or dangerous as it sounds. Anyone who tried to schedule a project with a team of more than fifty people knows that the idea of ever having "full control" over the process is at best a comforting illusion. The same model works with distributed development.

The ability to use external libraries and compose applications out of them results in an ability to create more complex software faster and with less work. The trade-off is the need to manage those libraries and ensure their compatibility. That is not a simple task. However, there is no other practical, cost-efficient way when one wants to assemble systems of today's complexity.

The technological solution to the challenges of distributed development is modularization. A modular application, in contrast to one monolithic chunk of tightly-coupled code in which every unit may interface directly with any other, is composed of smaller, separated chunks of code that are well isolated. Then, those chunks can be developed by separate teams with their own life [23].

Boudreau, Tulach and Wielenga [23] cover the topics about the distributed development, modular programming, versioning, dependency management, modular manifesto and how to use NetBeans to do modular programming.

The proposed solution solves the problem in one particular IDE (NetBeans) in Java technology. What we need is a solution that extends far beyond the borders of a technology or IDE.

Busbee [24] has investigated the modular structured approach using C^{++}. The author describes the possible approach for creating modular system in C^{++} using the elements of the programming language. This is a modular system on a low level and not for web-based systems.

There is a solution for creating modular system in assembly languages. J u [25] describes the modular programming conventions in assembly languages. His paper proposes a calling sequence convention and a calling sequence handler for intermodal communication. The scheme is simple to use and enhances good programming practices, such as simplicity, flexibility, comprehensibility and integrity.

The solutions described in [24] and [25] are on technology (programming language) level and they do not offer solution for web-bases systems. Our goal is to propose a system on much more abstract level.

I z a d k h a h, E l g e d a w, and I s a z a d e h in [26] investigate the E-CDGM: An Evolutionary Call-Dependency Graph Modularization Approach for Software Systems. Since we offer a model of a modular system, the proposed approach is applicable in our research.

Z h e n g et al. in [27] gives a formalized definition of workflows constrained by an input and output. A Petri Net-based model is proposed. One of our modules is based on Petri Nets theory and the mentioned research in [27] gives a possible approach.

A t a n a s s o v, D i m i t r o v and A t a n a s s o v a in [28] present formal definitions of the concepts Intuitionistic Fuzzy Generalized Nets from first, second, third and fourth type and describes the algorithms for tokens transfer within the separate transitions and the nets in general. Generalized Nets are an extension of Petri Nets. It is another tool for modelling and optimising discrete parallel processes. In our modules, we are searching for formal mathematical approach to model discrete parallel processes. This research gives another mathematical formalism for implementing workflow management system.

4. Methods and model

We propose a method for building web-based systems – RADWIS Modules. The method is using a model of a modular system. In addition, the methods of the object-oriented, aspect-oriented, modular programming and mathematical formalism (Petri Nets theory) have been used.

The problems solved by the web-based information systems are very different. The requirements are sometimes controversial. It is impossible to build a system that meets all requirements and solves all problems. Despite the fact that the requirements for the web-based information systems are very different, there are some of them, which are common. All systems have to be high quality and built as fast as possible. They have to be flexible, robust, and easy to install change and manage.

4.1. Requirements

One of the most important steps is to determine the requirements for the model. The model should allow building a modular system having certain characteristics. The modules should be built in a way, which allows extending their functionality, without changing their internal structure. They have to be open for extensions and closed for changes. Their flexibility should allow easy adapting to different applications. The

model should allow the modules to be distributed across different hosts in Internet, and communicate remotely with each other and with other systems. This will allow the individual modules to be built with different technologies. If the modules are situated in the same host system, they should be able to communicate using the environment provided by the host system. Each module should have a single responsibility and centralised communication between the layers.

Very important aspect of the model is to provide a standard way for installation and sharing the modules. The model needs to offer a standard infrastructure for creating, building, sharing and installing the modules. The modules built using the model should be able to use the standard package/dependency manager of the technology.

The communication with the database and other data sources should be done in the most abstract possible way and the modules should be independent from the database schema. The database is situated in the core of the application. The changes in it creates waves of changes in all other logical layers. The model should offer a solution to this problem.

Very often, the mechanism for authorization is distributed across the elements of the system. This way the elements have more than one responsibility and they violate the single responsibility principle. To solve this issue, the model should allow a centralised, abstract, reusable mechanism for authorization.

The model should offer ways for managing different type of content/data and a workflow management system (workflow engine). The model should offer a formal, abstract way of managing discrete parallel processes. This will expand the areas where the model can be used. Having an abstract layer offering a standard, formal, mathematical approach for solving problems could change the way of building applications.

Using popular, high quality, robust frameworks will increase the speed and quality of the development process. The model should allow easy deployment on many different host systems and many developers with different background, skills and experience, to work on the project.

4.2. Conceptual model

The conceptual model of the RADWIS Modular system is represented on Fig. 2. Its structure allows the model to address each of the desired characteristics described in 4.1.

Each module has only one dependency from the host system. This is the Event Manager. This way every module can be replaced without affecting the rest of the system. Every logical layer of a given module can be extended or replaced by the modules loaded later in the system configuration file. The module remains closed for internal changes, but allows replacement or extension of each layer.

The flexibility is a consequence of the openness. The flexibility of the system comes from the fact that modules loaded later in the configuration file can replace or extend logical layers of the previously loaded modules. This way every aspect of the functionality can be extended or changed.

The modules can communicate between each other and other systems using web services or sockets. This will allow each individual module to be built with a different technology.

The responsibilities defined by the model are Authentication, Authorization, Navigation, Content Management System (CMS), and Workflow Engine. The responsibilities reflect the base modules that need to be created. Many more modules can be added to fulfil other requirements.

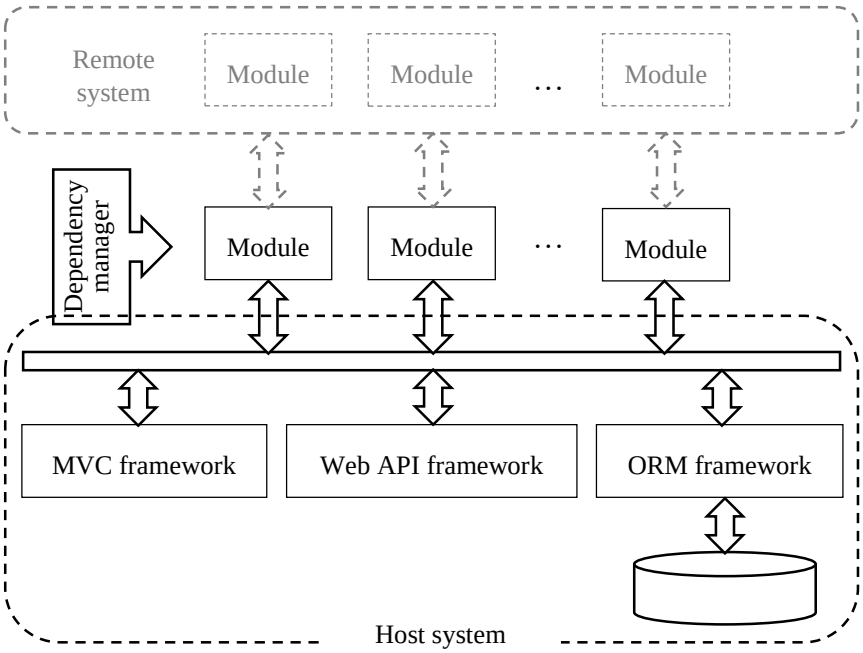

Fig. 2. RADWIS Modular system: Conceptual model

To solve the problem with the centralised communication between the layers, the model is using the methods of the Event Driven Development (EDD). Each module has only one dependency from an object called Event Manager.

The installation of each module is done through the standard for the technology package/dependency manager. Each module should meet the requirements of the standard package/dependency manager.

To abstract the communication with the data sources the model is using suitable for the technology Object Relational Mapping (ORM) framework. The framework creates an abstract layer between the application and the persistence layer. Even though the ORM frameworks offer abstraction and separate the application from the persistence layer there is still a dependency from the database schema. The model can use NoSQL approaches in SQL databases [29] to solve this problem and makes the modules independent from the database schema. We pay a special attention to the NoSQL approaches in SQL databases in 4.4 due to its importance for the model.

To allow dynamic management of resources and users, the model is using a special Authorization Module. The module is using the principles of the EDD to

intercept the requests for accessing the resources. Access Control List is used to determent the privileges of the user for a given resource.

The conceptual model includes a CMS module. Another optional module is the File Manager. It may be included into the set of the modules.

To increase the abstraction of the system and expand the number of problems that can be solved, the model includes a workflow management system (workflow engine) module based on the Petri Nets theory. It deserves a special attention and it is described in more details in 4.3.

The model uses popular, high quality, robust frameworks as it is shown on Fig. 2. The frameworks have documentation, tests, coding and naming conventions. They bring discipline and organisation in the project. If there are no suitable frameworks, we have to create our own for a given technology.

To allow easy deployment, the technology chosen for building a given module should be able to work on many different operating systems with different web and database servers.

The Agile Software Development and Extreme Programming should be used. These methodologies allow many developers to work at the same time on the modules.

4.3. Workflow Management System (Workflow Engine) Module based on Petri Nets theory

In order to implement a Workflow Management System (Workflow Engine) Module, first a suitable formal approach for modelling a workflow process needs to be found. Petri in [30] is the first to formulate a general mathematical theory for discrete parallel processes.

Petri Nets is a formal and graphical language for modelling discrete parallel processes [30, 31]. It is a generalisation of the automata theory.

The main reason for using Petri Nets theory is its popularity and widespread. There are a lot of publications, software tools, extensions, etc., which can be used. For example, there is a Platform Independent PetriNet Editor (PIPE) available. It is a graphical tool for modelling discrete parallel processes. Petri Nets theory even has its own Petri Net Markup Language (PNML).

The Workflow Management System (Workflow Engine) Module is one of many RADWIS Modules. Its purpose is to increase the speed of creating web-based software systems. It enriches the abilities of the modular system to solve different types of problems. The module adds an extra abstraction layer and can manage and automate discrete parallel processes. Every problem, that can be described with the tools of the Petri Nets theory, can be automated and managed. It is a tool for solving a huge amount of problems. The module allows a program to be built with graphical tools and implemented almost without writing programming code.

The objects in the Petri Nets theory are described by P e t r i in [30] and shown on Fig. 3. They follow formal mathematical rules described in the theory.

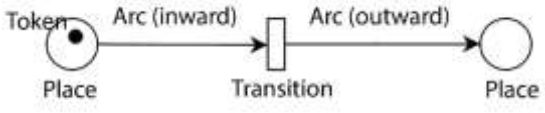

Fig. 3. Objects in Petri Nets

The logic of the module is using the Petri Nets theory objects and rules to create an environment for executing workflows. Each participant has its own role in the workflow. In a workflow management system, each user has a so called “in-box”. This in-box contains task instances (workitems) which can trigger a transition.

The workflow engine module is not an obligatory part of a web-based information system. It is used at will and brings formal mathematical approach for solving huge amount of problems.

4.4. NoSQL approaches in SQL databases

The model of RADWIS Modules can use NoSQL approaches in SQL databases, described in [29], to decouple the modules from the database schema.

NoSQL databases are a perfect candidate to fulfil the requirements of the model. However, they come with some disadvantages. They do not support transactions and do not follow the Atomicity, Consistency, Isolation, Durability (ACID) principles. To avoid the problems and to combine the advantages of NoSQL and SQL databases, the modern applications are using many databases (New Data Base or Polyglot Persistence). The main disadvantage of this approach is the fact that we have to manage many different types of databases. This introduces complexity in development and maintenance.

The approach described in [29] combines the advantages of NoSQL and SQL databases by using single SQL database. Such approach has the following advantages:

- We have to manage only one database server. This decreases the cost of the development and maintenance.
- The developers have to know only one database
- We can use standard software tools for SQL databases.
- We can use standard SQL queries.
- We can use transactions. They guarantee the integrity of the data.
- We will have the flexibility of the schema-less NoSQL databases
- The fact that there is no schema in the database will allow us to postpone making business decisions and working in very dynamic environment.
- The approach of using single SQL database will allow bringing flexibility into existing projects already using single SQL database.

5. Results

As a result, of our research, the described in 3 methods and model were used, to build a prototype of a modular system. The prototype can be used as a proof of a concept. First suitable technology needs to be found, which matches the requirements of the

presented model. PHP was used to build the prototype of the modular system. It is the most used technology for web based informational systems. PHP is created for building web-based information systems. It is open source and has big and vibrant community of developers. PHP has a standard dependency manager and many frameworks for building web-based systems, ORM frameworks and frameworks for web services. The deployment of PHP application is easy and affordable. We can use Virtual Private Servers (VPS), dedicated servers and shared hosts. PHP runs on many different operating systems and integrates very well with different web servers and database servers. The programming language PHP is extremely easy to learn and work with. This would allow developers without experience and skills, even students, to work on the project. All these features of the PHP technology perfectly match the requirements of the model and allow the described methods to be applied. One possible disadvantage is the fact that on PHP the same functionality can be achieved many different ways. To avoid this problem we have to use frameworks, Test Driven Development (TDD), documentation, coding and naming conventions, Agile Software Development, and enforce discipline. PHP is a loosely typed or dynamic language, which means it is not type safe. However, the use of discipline, Integrated Development Environments (IDE) and the fact that there is no wasted time for compilation and build can speed up the development process.

PHP has a very popular dependency manager – Composer, which perfectly matches the requirements of the model.

In PHP, there is a huge amount of frameworks to choose from. After analysing the frameworks in PHP we have chosen, Zend as a base framework for building the modules, Doctrine as an ORM framework and Apigility as a framework for creating the web services.

Zend is a framework that matches perfectly the requirements of the model and methods described in 4. It implements the best practices and design patterns used also in technologies like ASP.NET MVC and Java Spring MVC. Zend combines perfectly the advantages of the PHP technology with the ideas coming from technologies like .NET and Java. Zend is in fact a component library. What makes it a framework is one of its components – MVC component.

Doctrine is an ORM framework for PHP and perfectly matches all the requirements of the model. Doctrine shares the ideas, design, patterns, best practices from ORM frameworks like Entity Framework in .NET and Java Hibernate. Doctrine is perfectly integrated with Zend components. It works with SQL and NoSQL databases.

The requirements of the model for a web services framework, are met by Apigility. It is built with Zend components and it is perfectly integrated with it and Doctrine, which makes it perfect choice. Apigility allows the modules to communicate through web services.

In the modules, we are using the very well-known and established Model View Controller (MVC) architecture. Other logical layers can be added if necessary.

The base modules for the modular system are the modules for Authentication, Authorization, Navigation, CMS, and Workflow Engine.

Part of the modules were created as an Open Source Project, to experiment with the community reaction and willingness of the developers to join the project and work on the implementation of the model.

5.1. Architecture

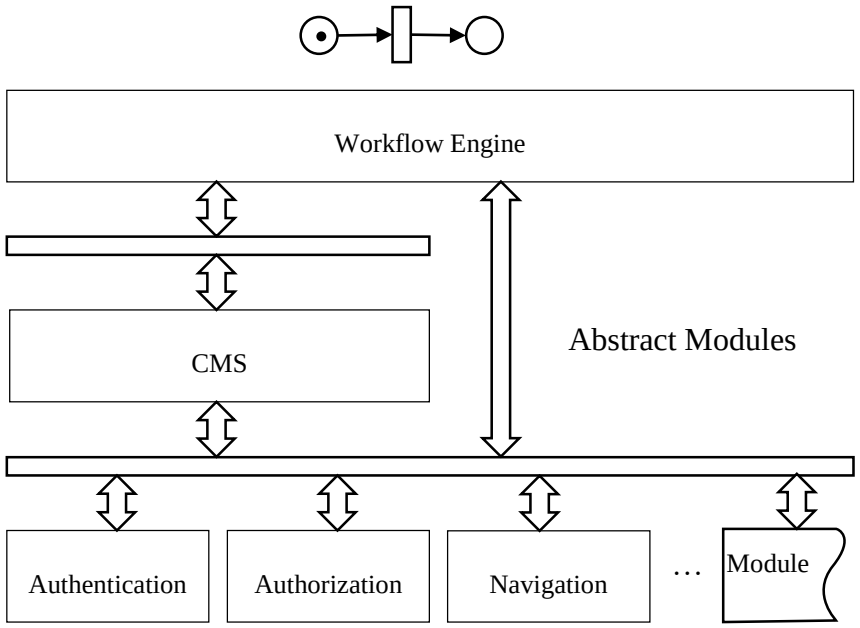

Fig. 4. Architecture

The architecture of the RADWIS Modules is shown on Fig. 4. The system consists of base modules, such as Authentication, Authorization, Navigation, etc. The base modules create an environment on top of which modules that are more abstract can operate. One of them is the CMS module. The module that offers the maximum level of abstraction and mathematical formalism is the Workflow Engine Module. It creates a new quality and allows a different way of building applications. All base modules usually have abstract core, configuration, data access layer, presentation layer, web services, business logic, etc. They will be discussed in the following paragraphs.

The abstract core is isolated in the service layer, in the prototype. The input controller knows only the interface of the service it uses. This way every aspect of the functionality can be changed. One way to make this change is by module that loads after the original module. The later loaded modules can extend or replace the existing implementation. The abstract core consists of interfaces, implemented by the concrete implementation of a module. The interfaces itself can be encapsulated in an abstract module, which does not have concrete implementation and cannot be used directly [9].

The configuration of the system is done with a configuration file. Each module needs to be declared in this file. There is another approach with dynamic loading of

the modules and checking certain folders for modules. The former approach is not used in the prototype because is more difficult to be implemented and the performance hit impact. Much simpler and faster is to explicitly declare the existence of a module in a configuration file. The configuration file is using plain PHP associative array. This approach is simple, because it does not require parsing of XML, YAML or other formats.

The Data Access Layer is in fact the ORM framework Doctrine. We are not using an adapter to separate the ORM framework from the modules. This approach greatly simplifies the implementation, but creates dependency of the modules from ORM framework – Doctrine. This is not of significant importance for the prototype in this case, because the modules are explicitly based on three frameworks. The module has access to the Entity Manager of the ORM framework through its single dependency – Event Manager.

The presentation layer is created with HTML, CSS and JavaScript. The input controllers pass the data to the view scripts, which use PHP as a template language and put together HTML, CSS and JavaScript. To create the final interface is used "Two Steps View" design pattern [9].

The web services are created using the framework Apigility. It is perfectly integrated with the framework Zend and Doctrine. The web services use the service layer. Apigility offers RESTful services. The model allows using of Remote Procedure Call (RPC) or even sockets for communication if necessary. The prototype offers only rESTful services.

Composer is used to create and package each module. It allows packaging of any kind of resources and installs packages for each project individually.

5.2. Authentication module

This module implements the authentication mechanism. The usual way of authentication is by username and password. The module allows other authentication mechanisms by changing the adapter. The functionalities offered by the concrete implementation are:

- Registration of the user by saving the username, name, password and e-mail. A confirmation e-mail can be sent at will. This option is available in the configuration of the module.
- Confirmation e-mail with a unique link for registration confirmation.
- Using the encryption mechanism (md5, bcript, etc.) with static and dynamic salt.
- Extracting part of the user data and saving into the session for faster access.

5.3. Authorization module

The authorization module controls the access of the users to the resources of the system. The control can be done by using Access Control Lists (ACL) or Role-Based Access Controls (RBAC). Changing the mechanism can be done by changing the adapter. ACL was used in the prototype.

The module is listening for events during the routing process in the system (the event “route”). After routing, it is clear what resource the user is trying to access. At this moment, the authorization mechanism needs access to ACL. In the ACL the resources, roles and privileges are described in human readable format. This way the authorization services are independent from the concrete implementation of the system.

The ACL can be saved in a file or database. The configuration file allows to choose between using a file or database.

The existence of Event Manager in Zend makes it very easy to integrate the authorization module. The subscription for the event “route” is implemented in the “Module” class of the module.

5.4. Navigation module

Every web-based information system needs a menu to allow the users to navigate through the functionalities. The menu organises the resources in a logical way. Creating menus is the main purpose of Navigation module. A special configuration file allows describing the resources in the application in a uniform way. The prototype is using PHP associative array to describe the hierarchy structure of the menu. The module has integration with the presentation layer. There are two variants of this module. The first one is without dependency from the Authorization module/mechanism. The second variant has a dependency from the Authorization mechanism. The second module allows building different menus for any different role. There is no sense to show menu elements in the menu to resources which the user cannot access.

5.5. Content Management System (CMS) module

Every web-based information system manipulates data usually in text format. This is common functionality and can be created as a module. The responsibilities of this module is to manage content, usually in HTML format. The module can be augmented with a File Manager module that takes care of all kind of files.

In the CMS module, we need a way to control the access to the content. This is the reason the CMS module has a dependency of the Authentication and Authorization modules. Every document is a resource and every user that try to access the resource has a role. The privileges of the roles to access resources are described in the ACL.

5.6. Workflow Management System (Workflow Engine) module

We have used the model described in 4 to build a Workflow Management System (Workflow Engine). It is able to manage workflows with the combination of other modules.

There are many implementations of Workflow Management Systems, based on different approaches and mathematical formalism. Only some of them are based on Petri Nets theory. There is no clear definition for the design of a Workflow Management System based on Petri Nets theory. Our implementation is using PHP

and MySQL. It is based on Zend, Doctrine and Apagility frameworks. In this particular implementation we have created models of all Petri Net objects. We are using NoSQL approaches in SQL databases.

Our goal is to create a web application only by building a workflow model using formal mathematical approach. This way a web application can be created with the use of graphical tools without or with very little writing of programming code.

Our domain model perfectly reflects the objects from the Petri Nets theory. Each element from the Petri Net is represented by a class. The elements are Place, Transition, Arc and Token. We need to deal with more entities in order to represent the workflow in the domain model. The extra entities represented by classes are Workflow, Case, Work Item and Workflow Role.

In the prototype build with PHP, Zend, Doctrine and Apagility, is used the most complex approach with representing each interactive transition with an action in a MVC web-based system. In the interactive transition, is kept the URL of the action. Another possible approach is to hold the code (HTML, CSS, and JavaScript) necessary for building the user interface and logic to advance the process.

The interactive transitions in the prototype are actions in a MVC system build with Zend. The workflow management system consists of an abstract core and administration.

The abstract core has an interface and an abstract implementation. Each input controller from the MVC system, which wants to define actions representing workflow interactive transitions, should extend the abstract class. Each transition holds the URL of the action to be executed and the workitem holds the information about the arguments to be sent to the action. This way the action has everything necessary to build the user interface, interact with the user, manage data and perform the logic for advancing the workflow case.

The most important methods in the workflow engine interface are:

- consumeInTokens – consumes a token from all in places.
- placeOutTokens – places a token in all out places.
- finishWorkflowWorkitem – performs all steps to mark a workitem as completed.
- completeWorkflowTransition – performs the logic and calls the methods finishWorkflowWorkitem, consumeInTokens, placeOutTokens.
- getAllInTransitions – returns a list with all in transitions for a given place.
- getAllOutTransitions – returns a list with all out transitions for a given place.
- getAllInPlaces – returns a list with all in places for a given transition.
- getAllOutPlaces – returns a list with all out places for a given transition.
- isTransitionEnabled – predicate that checks if the transition is enabled.
- placeWorkflowToken – places a token into a given place.

There is an administration mechanism. It is built with the standard elements of Zend MVC, and offers user interfaces and logic to manage the workflow processes in the system. The tasks for managing the workflow processes include assigning users to tasks, changing roles, importing workflows, etc. Part of the administration is the inbox of each participant in the workflow process. Each task (workitem), which has to be performed is shown in the inbox of the user. Third party graphical tools, like

PIPE2, are used for creating and testing the workflows. In the workflow engine are imported only well designed, tested and valid workflows, since the prototype of the engine does not have a validation mechanism. Once the model of the workflow is created with graphical tools, it can be imported in the workflow engine.

The workflow engine module is not a mandatory part of a web-based information system. It gives abstract, formal mathematical instruments to solve, in a uniform way, variety of problems.

6. Discussion

If the standardisation efforts are widely accepted, the developers will write the code once. Modules with certain functionality can be used across the technologies for every project. The modules can be open source projects and many developers can work on them, constantly improving their quality. This will allow the developers to work on new topics instead of reinventing slightly different solutions to the very same problems. The result is a proof of a concept. It shows that a real system described in 5 can be build. Other modules with different functionalities can be added to the existing base modules. They can be created with different technologies. The interfaces of the modules can be standardized.

The new in the research is a method that uses a model for building a modular system of a new level. The model offers standard modules that allow rapid application development by keeping the balance between the complexity and functionality. It is using standard dependency managers allowing the usage of the modules across an entire technology and by using web services across many technologies. It is a standard uniform solution to common problems.

Another new element is using NoSQL approaches in SQL databases. They allow combining the advantages of the SQL and NoSQL within a single database. This approach decreases the cost of the project and simplifies the development process. One possible disadvantage is increasing the complexity of the code and execution time. The developers have to take care of converting the data from one type to another, sorting and filtering, something we have for granted when we use database management system.

The third new element in the research is the implementation of a Workflow Management System (Workflow Engine) using Petri Nets theory. The module allows solving many problems using a standard, formal, mathematical approach.

7. Conclusion

The modules, created with the described model (RADWIS Modules), have been used for building web-based business applications working in the real world. Some of them are financial applications controlling financial transactions and complex workflows, providing services for thousands of users. Other applications use the functionalities provided by the modules with range from controlling the user access to building complex CMS. Since part of the modules have been released as Open

Source Software, we cannot control the way the people use them. However, the positive response from the community and the great interest is a good indicator. RADWIS Modules augment the instruments for building VLS and help in the process of transforming DELC into VES, by providing module-oriented approach and abstraction with mathematical formalism. A possible way of extensions of the model is creating modules using different types of mathematical formalisms for example Generalized Nets.

***Acknowledgment*:** The authors wish to acknowledge the partly support of the NPD – Plovdiv University, under Grant No MU17-FMI-001 "EXPERT (Experimental Personal Robot That Learn)", 2017-2018.

References

1. Stoyanov, S., I. Popchev. Evolutionary Development of an Infrastructure Supporting the Transition from CBT to e-Learning. – Cybernetics and Information Technologies, Vol. **6**, 2006, No 2, pp. 101-114.
2. Stoyanov, S., I. Popchev, E. Doychev, D. Mitev, V. Valkanov, A. Stoyanova-Doycheva, V. Valkanova, I. Minov. DeLC Educational Portal. – Cybernetics and Information Technologies, Vol. **10**, 2010, No 3, pp. 49-69.
3. Sandalski, M., A. Stoyanova-Doycheva, I. Popchev, S.Stoyanov. Development of a Refactoring Learning Environment. – Cybernetics and Information Technologies, Vol. **11**, 2011, No 2, pp. 46-64.
4. Stancheva, N., A. Stoyanova-Doycheva, S. Stoyanov, I. Popchev, V. Ivanova. A Model for Generation of Test Questions. – Compt. Rend. Acad. bulg. Sci., Vol. **70**, 2017, No 5, pp. 619-630.
5. Stancheva, N., A. Stoyanova-Doycheva, S. Stoyanov, I. Popchev, V. Ivanova. An Environment for Automatic Test Generation. – Cyberbetics and Information Technologies, Vol. **17**, 2017, No 2, pp. 183-196.
6. Stoyanov, S., I. Ganchev, I. Popchev, M. O'Droma. An Approach for the Development of InfoStation-Based eLearning Architectures. – Compt. Rend. Acad. bulg. Sci., Vol. **61**, 2008, No 9, pp. 1189-1198.
7. Mitev, D., S. Stoyanov, I. Popchev. Selbo2 – An Environment for Creating Electronic Content in Software Engineering. – Cybernetics and Information Technologies, Vol. **9**, 2009, No 3, pp. 96-105.
8. Gamma, E., R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994, ISBN 0201633612.
9. Fowler, M., D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002, ISBN 0321127420.
10. Evans, E. Domain-Driven Design Tackling Complexity in the Heart of Software. Addison-Wesley, 2003, ISBN 0321125215.
11. Beck, K. Test-Driven Development by Example. Addison-Wesley, 2002, ISBN 0321146530.
12. Fowler, M., K. Beck, J. Brant, W. Opdyke, D. Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999, ISBN 0201485672.
13. Martin, R. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008, ISBN 0132350882.
14. Fowler, M. UML Distilled Third Edition, A Brief Guide to the Standard Object Modeling Language. Addison Wesley, 2003, ISBN 0321193687.
15. Freeman, E., E. Freeman. Head First Design Patterns. O'Reilly, 2004, ISBN 0596007124.
16. Martin, R., M. Martin. Agile Principles, Patterns, and Practices in C++. Prentice Hall, 2006, ISBN 0131857258.
17. Johnson, R., B. Foot. Designing Reusable Classes. – Journal of Object-Oriented Programming, Vol. **1**, 1988, No 6/7, pp. 22-35.

18. S c h m i d t, D. Applying Patterns and Frameworks to Develop Object-Oriented Communication Software. – In: P. Salus, Ed. Handbook of Programming Languages, MacMillan Computer Publishing, 1997.
19. K r a e m e r a, A. Block by Block Towards IoT Applications. White Paper Document from Internet. Last visited May 2017.
http://reference.bitreactive.com/papers/bitreactive-towards-iot-applications.pdf
20. B i t r e a c t i v e. The Secret Twists to Efficiently Develop Reactive Systems. White Paper Document from Internet. Last visited May 2017.
http://reference.bitreactive.com/papers/bitreactive-secret-twists.pdf
21. B i t r e a c t i v e. Overcoming the Challenges of Reusing Software. White Paper Document from Internet. Last visited May 2017.
http://reference.bitreactive.com/papers/bitreactive-challenges-of-reuse.pdf
22. B i t r e a c t i v e. Integrating Reactive Blocks with JamaicaVM. White Paper Document from Internet. Last visited May 2017.
http://www.bitreactive.com/wp-content/uploads/2016/09/RealtimeBlocks-Integrating-Reactive-Blocks-with-JamaicaVM-2016-09.pdf
23. B o u d r e a u, T., J. T u l a c h, G. W i e l e n g a. Rich Client Programming. Chapter 2-4. Santa Clara, Prentice Hall, 2007, ISBN 0-13-235480-2.
24. B u s b e e, K. Programming Fundamentals – A Modular Structured Approach using C^{++}. CONNE XIONS, Rice University, Houston, Texas, 2008.
25. J u, S. Modular Programming Conventions in Assembly Languages. National Computer Conference, 1977.
26. I z a d k h a h, H., I. E l g e d a w, A. I s a z a d e h. E-CDGM: An Evolutionary Call-Dependency Graph Modularization Approach for Software Systems. – Cybernetics and Information Technologies, Vol. 1**6**, 2016, No 3, pp. 70-90.
27. Z h e n g, C., Y. Y a o, S. H u a n g, Z. R e n. Modelling Workflow Systems Constrained by Inputs and Outputs − An Approach Based on Petri Nets. – Cybernetics and Information Technologies, Vol. 1**5**, 2015, No 4, pp. 27-41.
28. A t a n a s s o v, K., D. D i m i t r o v, V. A t a n a s s o v a. Algorithms for Tokens Transfer in Different Types of Intuitionistic Fuzzy Generalized Nets – Cybernetics and Information Technologies, Vol. 1**0**, 2010, No 4, pp. 22-35.
29. C h e r e s h a r o v, S., H. K r u s h k o v. NoSQL Approaches in SQL Databases. – In: Scientific Conference "Innovative ICT in Business and Education: Future Trends, Applications and Implementation", Pamporovo, 24-25 November 2016, pp. 79-88, ISBN 978-954-8852-72-2.
30. P e t r i, C. Kommunikation mit Automaten. – Instrumentelle Mathematik, Schriffen des IIM, Nr. 3, 1962.
31. P e t r i, C. A. On the Physical Basics of Information Flow. – In: K. M. Van Hee, R. Valk, Eds. Applications and Theory of Petri Nets. PETRI NETS 2008. – In: Lecture Notes in Computer Science, Vol. **5062**. Springer, Berlin, Heidelberg, 2008.