Balanced Abstract Web-MVC Style: An Abstract MVC Implementation for Web-based Applications

Nalaka R. Dissanayake
Department of Computing
Informatics Institute of Technology
Colombo 6, Sri Lanka
nalakadmnr@gmail.com

G. K. A. Dias
University of Colombo School of Computing
Colombo 7
Sri Lanka
gkad@ucsc.cmb.ac.lk

***Abstract*—The features and the capabilities of web applications are growing rapidly, and the complexities and difficulties of web applications engineering are also growing in parallel. If the architectural formalism of these advanced web applications is well realized, the complexities could be understood, thus the difficulties could be reduced. Model-View-Controller (MVC) has been recognized as a well-formed architectural style, and has been widely used in web applications engineering in various forms of implementations. These MVC implementations are heavily dependent on specific set of technologies and/or some other facts; hence, they do not provide an abstract realization to be used in a wider range of web application engineering. We propose an implementation of MVC in more abstract form, which – we think – will increase the realization of the advanced web applications, thus lower the engineering complexities and difficulties of web applications. We believe that this implementation is more applicable in a wider range of environments and technologies, and will upturn the architectural properties like performance and modifiability. Based on this implementation we introduce an MVC based architectural style for web applications. In future, we expect to improve this further towards supporting Rich Internet Applications.**

Key words- MVC; Software architecture; Web applications

I. INTRODUCTION

The World Wide Web and the Internet have become popular, and web applications have marked their domain within the world of information systems, containing dedicated methodologies, Techniques and Technologies (TTs), which is rapidly growing [1]. The demand towards the web applications is increasing [2], since the web applications have been evolved, up to large-scale enterprise-level systems [1]. A large amount of related TTs in a wider range have been introduced throughout the last decade, to support the web application engineering.

This paper proposes an implementation of popular architectural style named Model-View-Controller (MVC) for the web application development, indicating the pros of its' use, against the cons of available MVC based TTs. This implementation utilizes a novel approach to adopt MVC into the web application development, introducing a new MVC based architectural style, which is not depending on TTs, thus abstract.

This section discusses the target problem, which the paper is focusing on, and the motivation, then the methodology we utilized in this research. Section II provides the background of the domain, giving an overview of the web applications, software architectures, and MVC, then gives a detailed discussion about the classic MVC and its' adoption in desktop applications development. Section III reviews the available work related to the MVC adoption in the web, highlighting the cons of available MVC implementations for the web, which are supposed to be addressed by the MVC implementation introduced by this paper. Section IV discusses the abstraction of the concept of the MVC towards the web applications development, then section V delivers the derivation of the proposed architectural style, based on the discussed abstract MVC version for the web. Section VI evaluates the introduced style, and finally section VII concludes the paper, stating limitations and future work.

A. Problem and Motivation

The web applications have evolved into more advanced systems and their complexity has grown significantly [1], where diverse types of components are integrated into various ways in modern web applications, therefore causing difficulties in understanding the architectural formalism of them. This setting affects the engineering processes of the web-based systems in a negative manner. To address the related issues and support web engineering, numerous concepts, and TTs have been introduced. These supporting artifacts come with additional learning curves along with their pros and cons.

However, the foundation of these advanced web applications is still laid on the Client-Server (C-S) model, which is the basic architectural form of any web application [3]. Therefore, we assume that there are common characteristics – mainly related to the architectural formalism – among the web applications, regardless the scale and the TTs used in the development. Furthermore, if this common architectural formalism is well identified and specified, it may assist in increasing the realization of the web applications, and thus in reducing the complexities and difficulties, in advanced web applications development.

There are architectural styles already available, which help in reducing complexity by increasing the realization of the formalism of the web applications; however, we note that they are heavily depending on some specific set of TTs, thus not abstract. The term abstract is used in this paper to denote the independency of the concepts from TTs. The notion of the term "abstract" can be well explained through an example. The concepts like Object Oriented Programming (OOP) and C-S style can be given as abstract concepts, which are independent from any TTs. Once the concept is understood, they can be easily adopted into development regardless the TTs used. TTs independency provides advantages like: conceptual abstraction of common characteristics, increased realization, knowledge and experience sharing, lower learning curve, and assistance in better TTs selection and adoption [4]. Deep discussions

DOI: 10.5176/2251-3043_5.3.375

about the TTs independency and its pros and cons are intentionally kept out of the scope of this paper.

The architectural style introduced in this paper – based on the proposed implementation of the MVC for the web applications – is abstract thus independent from TTs. In this style, we attempt to increase the visibility (separation of the components) in the web applications in an abstract manner, increasing the realization of the formalism of the web applications, based on the proposed implementation of the MVC. We worked on identifying the ground basic TTs when designing the proposed architectural style, to make it more abstract, hence TTs independent. We think that the abstraction provided by this style will make it more flexible, thus will offer easy adoption in web development, in wider environments, while highly satisfying architectural properties such as performance, modifiability, and scalability.

B. Methodology

A literature survey was conducted to gain the domain knowledge of the areas of the web applications, software architecture and architectural styles, conceptual abstraction and TTs independency, MVC, and the TTs used to develop MVC based web applications.

It was noted that there is lack of literature for TTs independency and the concept abstraction. Therefore to gain that related knowledge through experiencing the utilization of available TTs into MVC based web development – towards gaining empirical evidence – a series of experiments was conducted. The experiments were prototype based and conducted in an incremental manner. Facts learned in literature were tested in early iterations, in the direction of identifying the bottlenecks and issues, then solutions for the identified problems were tested in later iterations. Identified solutions were continuously refined to verify that they do not conflict with the artifacts found in later iterations.

To develop the web pages of the prototypes, HTML5 and CSS3 were used; the client-side component development was done using JS and jQuery; for the development of the server-side components PHP was used. Apache was used as the web server and MySQL was used for database development, which were hosted locally utilizing the XAMPP tool.

The empirical evidence gained through the experiments can be considered as the backbone of this research, and they were utilized to formulate the requirements for the introducing style and derive it. Additionally, some parts of the research were presented in research conferences, and the feedback received was taken into the considerations.

II. BACKGROUND

In this section, an overview of the web applications, software architecture, architectural styles, and their evaluation, and overview of MVC is given. Then the background of classic MVC and its adoption in desktop application development is discussed, to lay the foundation for the rest of the paper.

A. Overview

This section provides an overview of the web applications, software architectures, architectural styles, and how they are evaluated, and overview of MVC, in order to deliver the basics of the domain covered in this paper.

1) Web Applications

The web provides a platform for the multi-user systems with centralized control and management. Discussions on the web, the web applications, and other web-based systems have their own larger space. In this section, we focus on the basics of the web applications, which run on the browser, providing an overview of their components and development TTs.

The fundamental architectural model of a web application is the C-S architecture, which involves two separate nodes, the client and the server. The C-S model – in other words, the two-tier architecture – is the key for centralized control and management of the web applications [5], yet it makes the components of the web applications location/partition-dependent (components are depending on either client or server nodes), therefore not run in a single address space [6]. The web applications use the request-response model for communication between the client and the server nodes – and between other nodes if available – over the HTTP protocol [3].

Since there are multiple components in the web applications, and they run in multiple locations and environments (mainly – and at least – in client and server nodes) – which are different in platforms from each other – these components are heavily technology-dependent. For example in client-side, for the web browser platforms, for Graphical User Interface (GUI) components, languages like HTML and CSS are used, for client-side application component development JavaScript (JS), and various JS-based libraries are used; and for server-side components, languages like PHP or JAVA, and related TTs like servers, platforms, frameworks, etc. are used.

The web applications nowadays are evolved and very advanced, and they are usually compounded with customized navigation topologies, and they support for triggering transactions in their underlying corporate applications [7]. The size and the capabilities of these web applications have grown within a short period of time. All these facts – location and TTs dependencies, complex structures, variable size, and higher capabilities – together have made the web application today a very advanced and complex entity. Therefore realizing the abstract formalism of them is essential towards hazel-less development.

2) Software Architectures and Architectural Styles

Software architecture provides a high-level abstract view of the components within the system and their relationships, at the run time [8]. Architecture can be seen as the foundation of the software system, and it offers a strong structure with many advantages, for carrying out smooth engineering operations [9]. Fielding [8] classifies the architectural elements within a system into three main types: the processing components, communication connector components, and data elements; and the integration relationships of these elements is referred as Configuration.

Architectural patterns – in other words, architectural styles – provide means for capturing the knowledge about successful solutions in software development. According to Fielding [8], "an architectural style encapsulates important decisions about the architectural elements and emphasizes important constraints on the elements and their relationships." Architectural styles can be designed by adding constraints to the null style – which represents the style with no constraints – or to an existing style [8]. When designing styles, identification of abstract architectural

elements and their configuration, separating the components well enough towards lowering coupling is essential. Furthermore, constraints addition may align to a desired set of architectural properties, to be satisfied by the end result.

This paper is focusing on the architectural properties below [8].

1: **Simplicity** – Employment of the 'separation of concerns' principle [10], to effectively allocate functionalities within the component(s) through proper modularization.

2: **Modifiability** – Ability of tolerating the changes made to the application architecture in both initial engineering and post-deployment stages. This is described under four sub sections as below. Again, the modifiability also can be increased with proper modularization.

2.1: Evolvability – The degree to which component(s) can be changed without negatively affecting other components.

2.2: Extensibility – Ability to add functionality to the system, without negatively affecting other functions.

2.3: Customizability – Ability to temporarily specialize the behavior of an architectural element, so that it can then perform the service in a different way. For example: a component is customizable if it can be changed for one client, without adversely impacting other clients who get the service of the same component.

2.4: Configurability – This is related to both extensibility and reusability in a way that it refers to post-deployment modification of components, or configurations of components.

3: Reusability – The property of an application architecture, if its components, connectors, or data elements can be reused – without modification(s) – in other applications. The primary mechanisms for inducing reusability within architectural styles is by reducing the coupling (knowledge of identity) between components and constraining the generality of component interfaces. This also can be gained through efficient modularization of the system.

4: **Performance** – Performance can be increased by gaining the maximum utilization of client-side processing power via proper modularization.

5: **Scalability** – Ability to scale, serving a high number of clients, through effective modularization. Additionally consider the ability to utilize the WS, in order to address the C10k scalability issue [11].

Dedicated methodologies such as SAAM [12] and ATAM [13] are utilized to objectively evaluate the architectural styles, without implementing them. This research uses the method specified by Fielding [8], which uses a derivation tree to indicate the architectural properties induced by the application of the constraints.

3) MVC

MVC can be seen as one of the available architectural patterns, which is a popular and widely used pattern [14]. The MVC concept was introduced in 1970s by the company named Xerox Parc, with their development environment called Smalltalk-80. MVC can be considered as a use of protocols to define components, instead of using concrete implementations [15]. The MVC paradigm is elegant, simple, and different from the traditional programming approaches [16]. Currently, the MVC is widely accepted as both a design pattern and an architectural style, in both desktop and web-based systems, engineered using Object-Oriented paradigm, in corporate software development [14] [17].

Selfa *et al*. have discussed some advantages the MVC is engaged with, such as: less coupling, higher cohesion, bigger flexibility and agility provided by the Views, more design clarity, and facilitations for the maintenance [14]. Prakash *et al*. explain the benefits of MVC, such that "migration of legacy programs has become easy since the model and controller are totally separated in MVC and it makes tailoring the user category or platform much simpler" [18].

B. Classic MVC

This section discusses the features of the original version of the MVC as introduced by Smalltalk, in the direction of understanding the abstract concept behind its formulation.

MVC provides a responsibility-based modularization for software components, explicitly for the software systems with GUIs [15]. Modularity of the components in a software system provides some good benefits. For example, it helps in isolating functional units from each other as much as possible with low coupling, which makes it easier for the application designer and the developer to understand and modify each particular component, without having to know everything about the other components [19]. Also, modularity helps the conceptual development of the system, and also allows reusability of the components [19].

Modularization in MVC separates the parts, which represent the logic of the underlying domain, from the ways the information are presented to the user, and from the ways the user interacts with the system, using three modules, namely the Model, the View, and the Controller [19]. This module separation enables different teams of developers to independently work on each module either in series or in parallel [20]. And, this separation helps to lower the coupling between the modules, and hence reduce the complexities in architectural design, which leads to increase the flexibility and code reuse [21]. Fig. 1 shows the classic MVC as presented by Krasner *et al.* [19].

1) Modules in MVC

This section discusses the modules of the MVC paradigm as it was introduced by the Xerox Parc in Smalltalk-80, indicating the rationale for them.

Model: The module, which represents the knowledge of the domain, is called the Model. The Model is developed of classes, which are responsible for the application domain specific information [19]. It should be noted that the Model contains and manages both the behavior and the data of the domain [16].

View: A View displays the application's state, which is developed as a GUI window; and the Views can be considered as the user's version of the Model [19].

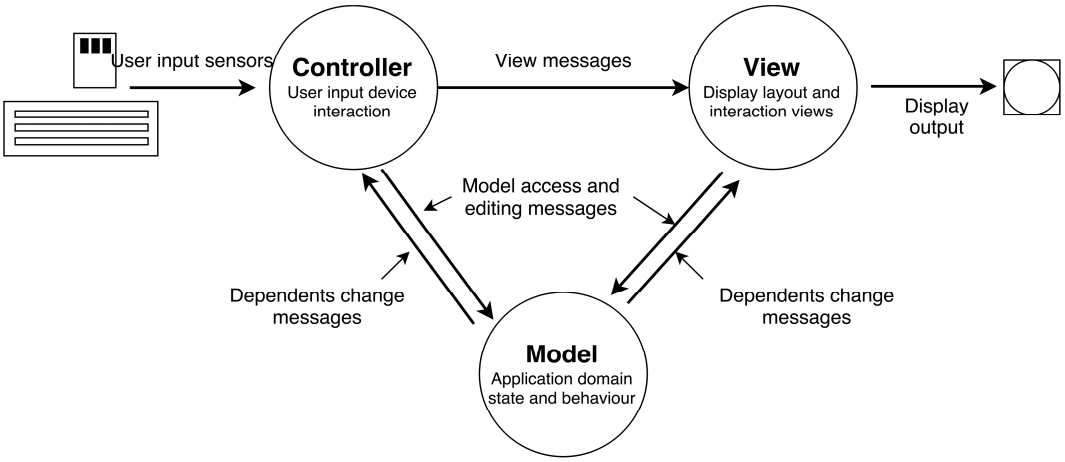

Figure 1. MVC as in Smalltalk [19]

Controller: The Controller is the middleware between its associated View, Model, and the input devices [22], and it also provides an interface between the associated View and Model [19]. The Controller interprets the user inputs via keyboard or mouse into commands to the Model or/and the View to change as appropriate [16]. We can imagine the Controller as an event-handling mechanism like in modern event-driven programming frameworks such as JAVA, which has a built-in event loop, so that the developers have to code only for the event-handlers.

2) The Communication between the Modules

The View holds the responsibility for establishing the intercommunication within an MVC triad [16]. Each View is closely associated with a unique Controller and vice versa, and they maintain a tight coupling [16]. View and Controller classes have a certain set of messages that they have to respond to [15]. Since the View and the Controller are explicitly meant to work together, the communication between the View and its associate Controller is straightforward [16].

The modules' knowledge of the existence of the other modules is as follows: the Views and the Controllers need to know about their Model explicitly, but the Models need not know about their Views or Controllers [19]. Views register themselves as dependent on their Model and respond to Model's change updates [15]. As per Krasner *et al.*, each View has exactly one Model, but a Model may have many View-Controller pairs [19]; nevertheless, there cannot be any Model-View pair without a Controller [16].

The Model responds to the requests of the state changes from the Views, and to the instructions to change the state from the Controllers [16]. The Model's communication can be divided into two modes, the passive mode and the active mode [16]. In passive communication mode, the user initiates the process and the Model acts on the commands of the Controller [16]. In active mode, the Model broadcasts the updates to all the dependent Views, when its state is changed [15] [16].

A complete cycle of a process within the modules – and the user – is called an interaction cycle. Krasner *et al.* describe the standard interaction cycle of MVC as follows: *"t*he user takes some input action and the active Controller notifies the Model to change itself accordingly. The Model carries out the prescribed operations, possibly changing its state, and broadcasts to its dependents (Views and Controllers) that it has changed, possibly telling them the nature of the change. Views can then inquire of the Model about its new state, and update their display if necessary. Controllers may change their method of interaction depending on the new state of the model." *[19]*

C. MVC adoption in desktop application engineering

This section discusses how the MVC has been adopted in desktop application development. The knowledge provided by this section can lead towards identification of the limitations of the use of MVC in the web (refer section A under the section IV for details of the limitations of using MVC in the web).

Smalltalk framework had its own implementation for MVC development; however, when we adopt MVC in other development environments, we have to follow or design suitable implementation(s). Some languages have incorporated MVC into their frameworks, and additionally there are libraries to enable MVC adoption. Conversely, different environments utilize different techniques, and they have tailored the MVC concept to align to their environment towards better adoption. In this section, we briefly discuss how the MVC is interpreted by various researches, to be adopted in desktop applications engineering.

1) MVC Modules Definitions, as per Related Researches for Desktop Development Environments

View: There can be one or more Views for the Model, and they know the existence of the Model [17]. The Views present the data in the Model based on the current state – as the output to the user – and they receive inputs from the user [20].

Controller: Controller allows manipulation of the View, and the Controller handles the inputs [17]. Controllers know their Views and have the knowledge of the platform/operating system to manipulate the Views, and the events do come from the Controller [17].

As per Morse *et al.* [20], the Controller coordinates the activities between the View and the Model, and is responsible for processing inputs from the user. Based on the user input, the Controller determines: which methods of the Model should be invoked and which View should be updated with the results. Furthermore, the Controller provides the event-handling, and responds to the events triggered by the user on the GUI of the View. Morse *et al.* specify, that for

the moment, there is no way for the Controller to respond to the events triggered by the Model [20].

Model: According to John Decon [17], the Model is supposed to be a singular term, which represents the domain, and consists of a set of classes. John explains that the Model supports the underlying problem, and tends to be long-lived stably, as the problem itself. Moreover, he notes that the classes of the Model do not need to know anything about the connection to the outside world. John Decon suggests that "a better acronym for the architecture would be: M_dM_aVC," where M_d is the Domain-Model and M_a is the Application-Model. As per John, what Smalltalk programmers sometimes mean by "Model" is the Application-Model, and what the analysts and designers would think of "Model" is the Domain-Model. The Domain-Model supports and models the problem, where the Application-Model – which knows the existence of the Views – communicates with the Views, utilizing the Domain-Model. In other words, the Application-Model is the interface between the Views and the Domain-Model, and the Application-Model acts as a coordinator [17].

As per Morse *et al*. [20], the Model contains both data and methods of the application, and is often referred to as the business logic of the system. The same set of business logic – or parts of the Model – may be used by numerous different Views.

2) Communication between MVC Modules in Desktop Development

According to John Decon [17], there can be multiple Views for a given situation, and they need to know the existence of their Model. The events come from the Controller, and the Views register the handlers for the events they wish to control. When events occur, messages will be sent from View to Application-Model via Controller; then, the Application-Model utilizes the Domain-Model to process the message.

III. Review of Related Work

Since the web applications are partition-dependent (refer overview of the web applications in section II Background, for partition-dependency) they have their own set of characteristics over the desktop/standalone applications. The MVC had been originally introduced for partition-independent systems, which means that all the Model, View, and Controller modules reside and execute in a single address space; hence, partitioning-related issues do not arise [6]. Adoption of MVC in location-dependent web-based systems is a difficult task [6]; therefore, the adoption of MVC in the web applications may need some specific techniques or features.

In this section, we review how the MVC had been adopted in web development, by experts, in various works. We have identified some related work done by different researchers; each of them has its specific enhancements/modification/implementation of MVC, and we present these features under the work they have done.

The review is done in the context of the architectural properties presented in the overview section of the software architecture in section II background, along with facts: partition independency, TTs independency, and lower learning curve.

A. Oracle's Model2 architecture [23]

An architecture based on the MVC had been used in early JAVA-based web application development, which is called *Model1*, which uses JAVA-based TTs, such as JSP and JavaBeans. The advanced and enhanced version of the *Model1* is called *Model2*, which utilizes the servlets. This *Model2* architecture has been introduced for JAVA-based web applications, thus the development is based on Oracle JAVA and related TTs. The *Model2* is shown in Figure 2.

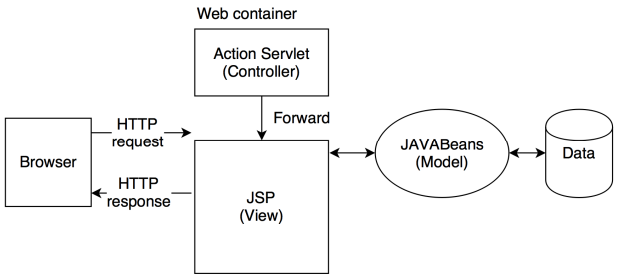

Figure 2. Oracle model 2 architecture [23]

The Model of *Model2* comprises JavaBeans classes, which define the internal state, and the actions that can change the state. The View is a web page constructed using JSP or Oracle ADF UIX technology. The Oracle ADF framework facilitates binding data of the Model layer to the Views in *Model2*. The primary component of the Controller is the servlet. The Controller focuses on receiving requests from the View(s) in the client-side browser, decides what business logics should be performed, and then delegates the responsibility to the appropriate View for producing the next phase of the UI.

Model2 adopts MVC entirely in server-side, therefore adequate support for performance and scalability requirements cannot be expected due to the higher load on the server. Since the client-side components are completely ignored, *Model2* does not sufficiently satisfy the simplicity. *Model2* is based on JAVA and related TTs, therefore the modifiability can be high, however within JAVA development environment; and also adoption into JAVA based systems could be adequately supported, with a lower learning curve, due to its abstract JAVA implementation. Since *Model2* architecture is highly coupled with JAVA and related TTs, it does not provide an abstract MVC adoption for the web-based applications, hence adoption of *Model2* in other development environments is not feasible.

B. Dual-MVC: Developing Highly Responsive User Interfaces with DHTML and Servlets [24]

The authors of this work, Betz *et al*. introduce an MVC-based alternative architecture, named *Dual-MVC*. As per their analysis – in terms of the classical MVC – the Model is a set of business objects, which entirely resides in the server. The View resides in the client-side browser, where the code and the logic for View generation reside in the server. A part of the Controller code – like button-click event-handlers, to submit forms to the server – resides in the client, and most of Controller code – to receive the client's HTTP request and invoke the appropriate methods – reside in the server.

As per Betz *et al*., the classical MVC-based server-centric approach has the simplicity, but the server must always be involved with the client's screen-update requirements.

Resulting overhead and latency of the communication may degrade the response time, and hence lower the user experience.

According to the analysis of Betz *et al*., the mixed approach addresses the performance problems associated with the server-centric approach, combining a limited form of client-side processing using JS. User input validations in mixed approach can reduce the round-trips and increase the performances. However, the refreshes need to engage with the server, since the Model resides in the server.

The *Dual-MVC* approach, proposed by the authors of this paper, addresses many issues in both classical server-centric and mixed approaches. This solution adds the classical MVC implementation to the client-side, and the client also maintains an MVC of its own, which does not rely on the server-Model. The client-Model corresponds to a relevant subset of the server-Model, and the state of the client-Model is typically more current than the server-Model. This technique enables the screen refreshes, entirely in client-side in certain cases, and therefore improves performance.

The implementation of the Dual-MVC contains two frames. An invisible frame called *anchor fram*e, which is consisted of 1) the client-Model, developed using JS, which maintains a set of JS variables; 2) a View, which generates DHTML source that represents the client-Model, then writes the DHTML to the *interaction frame*; and 3) the Controller, which is invoked by the *interaction frame*, when the client input changes and does not require server processing. The other frame, which is visible and called the *interaction frame* is similar to the screen in mixed approach. When the user requests for a screen refresh, interaction frame's JS processes the user inputs and updates the client-Model in the *anchor frame*. If the refresh does not need to be involved with the server, then the anchor frame updates its View to reflect the changes of the updated client-Model, and then writes the anchor frame View to the *interaction frame*. When the server must be involved with the refresh, the *anchor frame* sends an HTTP request to the server, the server runs its Controller, updates the Model, generates the View, and returns to the *anchor frame*. Then, this View will be mapped into an updated client-Model by the *anchor frame*. However, the synchronization between the client-Model and the server-Model lacks in this architecture; to maintain a better synchronization, some better techniques need to be used. Figure 3 illustrates the Dual-MVC architecture.

Conceptually, the application of separate MVC tirades in both server and client will increase the Simplicity; and as authors specify, the Dual-MVC also increases the performance. Since the Dual-MVC decreases the communication with the server, it will also increase the scalability. The simplicity provided by the Dual-MVC will address the modifiability, if the initial learning curve is ignored.

When the development aspects are considered, Betz *et al*., had used JAVA for the server-side development, however the server-side development is not tightly coupled with JAVA; which indicates that this concept can also be adopted in other environments. Anyhow, this approach is consisted with new types of components like *anchor frame* and *interactive frame*, therefore it is associated with a high initial learning curve; thus the adoption is not straight forward. And also, maintaining two separate MVC tirades in both server and client will introduce additional development workload, lowering the maintainability and the modifiability.

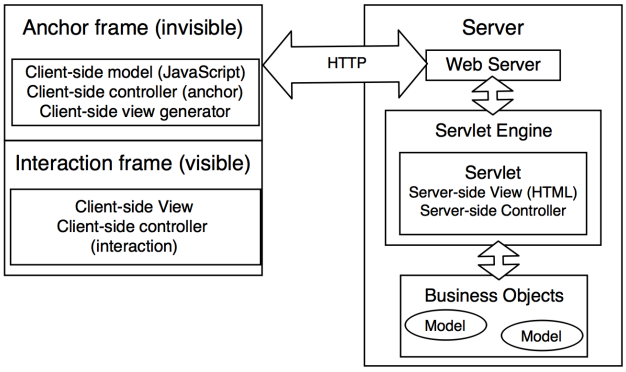

Figure 3. Dual-MVC architecture [24]

C. *Web Application Development Using the Model/View/Controller Design Pattern [6]*

According to the research by Leff *et al*., Views display the information to the user, together with the Controller, which processes and responds to the user's interactions. The Model contains both: the data presented by the Views, and the logic used to process the data, while responding the user's interactions. Authors approve that like in any other interactive software systems, the web applications can also benefit from the MVC. However, as they highlight, the problem with using MVC in the web is that the web applications are intrinsically partitioned between the client and the server. The View is displayed on the client, however – as they say – "the Model and the Controller can (theoretically) be partitioned in any number of ways between the client and server," which makes the situation more complicated and difficult.

Leff *et al*. note that neither the thin-client or thick-client is ideal for many environments. Furthermore, they specify that the web application could use MVC when the correct partitioning is known and if the available technological infrastructure is compatible with that partitioning. They introduce a concept named *Flexible Web Application Partitioning* (fwap) to enable more natural usage of MVC in the web applications.

Leff *et al*. analyze the MVC-based web application into three separate groups and discuss the adoption of MVC in these three groups by applying the fwap concept. The three categories are: 1) Single-MVC (smvc), which is the classical MVC application and more suitable for desktop applications; 2) Thin-client applications where Model and Controller reside and execute in single address space in the server, and generate View, which is rendered on the client; and 3) Dual-MVC (dmvc) where Model and Controller reside in both the client and the server [24].

The fwap is an approach of decision-making for designing of partitioning features between the client, the server, and the MVC modules. It does not provide any specific designing or implementation details; it is purely conceptual and the quality of the results mainly depends on the design and the experience of the designer. Therefore the satisfaction of the architectural properties in based on how the fwap is applied and the system is designed. However, as a conceptual approach fwap is TTs independent, thus may assist in wider environments.

D. The Research of PHP Development Framework Based on MVC Pattern [21]

This research introduces an MVC-based framework for PHP development environment. In this framework, the components in the server are partitioned into three layers: Data persistence layer, Business logic layer, and Web layer. The Data persistence layer and the Business logic layer are fulfilled by the Model, which contains the domain logic to add meaning to the raw data. According to the authors Cui *et al*., "MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the model."

The Web layer corresponds to the Views and Controller. The key of the View is to separate the code of HTML and PHP using templates, and this architecture uses a template engine to detach the program code from HTML entirely. The View renders the Model into a form, which is suitable for interactions with the user, by means of GUI. Multiple Views can exist for a single Model for different purposes. This implementation uses a dual-caching technique to prevent regeneration of same Views after the first visit.

The Controller helps in the separation of the Model and View layers, allowing the same Model to be accessed by a variety of Views. Controller receives the user requests, processes, and responds to the events; selects the right View to return to the user; and also may indirectly invoke changes/updates on the Model. The Controller is split into two, the Front-end-Controller and the Action-Controller, where the Front-end-Controller is used to achieve the centralized control of the framework, authentication, data validation, and other functions, and the Action-Controller is an adapter between the customer requests and the business-logic handling, which separates the business logic from the request(s).

The architecture of this research is illustrated in figure 4. This paper also includes some development-related details and sample code snippets in PHP.

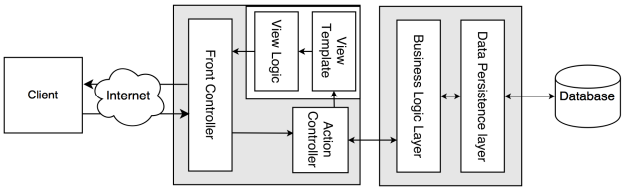

Figure 4. The architecture of the research PHP development framework [21]

This architecture uses MVC only in server-side, thus lack in simplicity, performance, and scalability. The thin client nature of this solution, however, may increase the assistance for modification and testing, by limiting the relevant activities to the server-side. Even though the development is targeted at PHP, the adoption of its abstract concept in other environments is also viable, since the development is not based on any PHP only techniques. However, the approach used in this architecture is novel, thus it may incorporate high learning curve.

E. Other Related Researches

This section provides brief discussions about some other minor MVC related work we identified in the literature survey.

1) A Database and Web Application Based on MVC Architecture [14]

The authors of this paper discuss a case study of using MVC in a web application, and they discuss the designing and development of the system. For the designing, UML is used, and for the development, ASP and VB are used. Their interpretation of the MVC in shown in Fig. 5.

As the authors Selfa *et al*. explain, the Model contains the application data and the core functionalities; the View manages the visual display of the Model, giving feedbacks to the user; and the Controller gets the mouse and keyboard inputs from the user, commanding the Model and the View to change appropriately. They use a passive pattern, where the Model does not know the existence of View or Controller, but in most cases the Model must have a link to the View to inform the changes made to the Model's state, caused by internal procedures. Each View is associated with a unique Controller and vice versa, where these particular Views and the Controllers are always connected. The Model can have more than one View-Controller pair at a time.

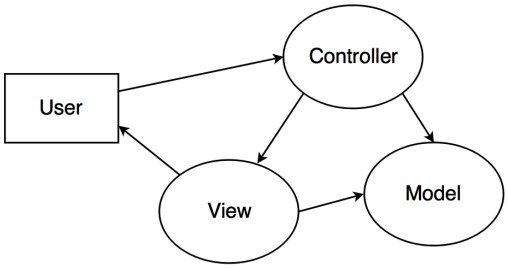

Figure 5. MVC adoption in the research - A Database and Web Application Based on MVC Architecture [14]

The Controller communicates with the View to determine which objects are being manipulated by the user, and call Model methods to make changes to these objects. Then, the Model carries out the changes according to the Controller's commands and notifies the View to update.

2) MVC Architecture Driven Design and Agile Implementation of a Web-Based Software System [18]

This paper reports design and implementation of a web-based system in JAVA environment, using Oracle's Model2 [23] architecture. The authors of this paper explain that "*Generally, in model2 web applications the web server (e.g. Tomcat in the case of TMS) contains view and controller both at application run time while data storage is done in third tier named as Model.*" Their specification about the Controller is such that the Controller is a component of the web server, which processes the incoming requests from the client to a single servlet instance. Since there is a heavy load of processing requests, the authors call this Controller a fat-Controller.

This paper presents a complete case study of implementing a web-based system, using UML for designing and JAVA-based development using Struts. The authors focus on the agile methodology based implementation and conclude that the idea had been implemented successfully.

3) A Software Architecture for Structuring Complex Web Applications [7]

In this paper, Jacyntho *et al*. discuss their views on the web applications in a perspective of object models, and justify the needs for improving current web architectures. They describe the *Object Oriented Hyper Media Design*

Model (OOHDM)-Java2 architecture, and present a case study using the OOHDM-Java2. The authors discuss a good set of literature, state an analysis of the limitations of MVC, and present an overview of OOHDM-Java2 architecture. Jacyntho *et al*. also introduce a mapping between the design model and the OOHDM-Java2 architecture, and discuss the design of the case study using UML. Furthermore, they discuss the development of the sample scenario in JAVA environment, giving the sample codes for various sections.

4) MVC Web Design Patterns and Rich Internet Applications [25]

This paper presents and discusses the concepts of MVC implementation, in the web applications and Rich Internet Applications (RIAs). The authors introduce the terms: 1) *Server-side MVC,* which contains all three modules in the server; 2) the Mixed client-side and server-side MVC, which gives more functionality to the client by taking some features of the Controller and View to the client; and finally 3) the RIA MVC, which takes the Controller and the View modules completely to the client, and keeps the Model in both server and client. Chaparro *et al*. review the three concepts indicating their limitations, and also discuss their iteration cycles.

The paper covers a good conceptual research and proposes an extended version of their architecture called RIA Deeper MVC for Rich Internet Applications. The authors do not discuss the implementation details and how these concepts can be designed and developed, yet the concepts presented are more abstract and – as they conclude – applicable.

F. MVC Adoption in Web Frameworks

In web application frameworks, mostly the MVC is adopted as single-sided-MVC, either in the server-side or client-side, or both sides separately. Some examples for server-side MVC frameworks are: CodeIgniter and Zend for PHP; Struts and Spring for JAVA; ASP.NET MVC for ASP environment. For client-side, frameworks like AngularJS and EmberJS are available, and the development using them is supposed to be based on JS.

Since all these frameworks are heavily technology-dependent and do not provide abstract concepts, we avoid deep discussions about them.

IV. Conceptual Abstraction of the MVC for Web

Despite all the advantages of the MVC, there are limitations when trying to adopt into the web. Adopting all the features of classic MVC into the web-based systems is not practical and feasible, due to these limitations. This situation can be seen as a reason for why still there are researches conducted, and various concepts and solutions are introduced for adopting MVC into the web applications development [7].

Moreover, these limitations might cause the cons of the available work as pointed in the literature review. One of the main and common characteristics of these available solutions is that they are highly dependent on technologies and/or some other facts. Thus, they do not provide abstraction – as expected from an architecture – which is needed for implementation in wider environments. These concepts/solutions add an additional learning curve into the process, and as they depend on a technologically limited environment, they also add constraints to the engineering process.

In this section, based on the literature knowledge and empirical evidence gained through experiments, we discuss the limitations, which affect adopting MVC into the web; and then propose a more abstract version of MVC, which can address the presented limitations, and flexible to be adopted into the web applications development.

A. Limitations of MVC in Web Adoption

Jacyntho *et al*. discuss some limitations of MVC in their paper [7], stating that the MVC does not fulfill the requirements of the web applications. As they say, MVC is based on transactional perspective of the software, and does not consider the navigation aspects in the web applications. Other than that, as we understand, the complexities engaged in MVC web adoption are due to two main characteristics of the web-based systems. And these might be the reason for most solutions out there to be technology-specific, and hence not abstract.

Partition dependency: In desktop/stand-alone applications, the MVC adoption is not much complicated, since all the components in desktop applications execute in a single address space, hence partition-independent, and the MVC is meant to be used in such environments. When it comes to the web-based applications, they are not partition-independent; they are always partitioned at least into two tiers: the client and the server. In greater context, they can grow up to n-tiers and/or much complex level as in Service-Oriented Architecture. This partitioned nature of the web applications can affect the effective adoption of the MVC.

Technology dependency: Different techniques and technologies are used to design and develop components in the partitions of the web applications; hence, these partitions and components in the partitions are technology-dependent. For example, the server-side components are developed using server-side technologies like JAVA, PHP, ASP, etc. The client-side components are mainly developed using JS and/or JS-based frameworks/libraries. The GUIs are developed commonly using HTML and CSS, and additionally either server-side or client-side, or both technologies can be used to generate the GUI components too. When it comes to adopting MVC into the web, which modules of MVC should be in which partition, and which TTs should be utilized to develop them must be decided carefully.

For MVC to be adopted into the web-based applications in a more-practical and less-complex manner with lower learning curve, more abstract implementation is required. Taking the two limitations discussed above into consideration, we propose that this abstraction should be partition and TTs independent; therefore, it can be used in a wider range of environments, despite the TTs.

B. Generalization of MVC in the Perspective of Web Applications

In this section, we propose a version of MVC, which we think is more suitable for the web environment. Since the MVC is used in an architectural perspective, we propose that the adoption of MVC in the web could be more abstract and free from technological dependencies. The conceptual presentation should be: easy to realize, without introducing additional learning curves. Additionally, the adoption of the abstract MVC into the web applications engineering should support the architectural properties presented in the background.

Without trying to address the issues related to the partitioning and technology constraints, in our approach we focused on identifying the features of these constraints, and utilize them in an advantageous way. Our approach can be seen as applying the features of the web into MVC, instead of applying the MVC into the web. First, we prepare the MVC in a more abstract version – which we suggest as more suitable for the web – and then apply the partition and technology constraints, deriving an architectural style, as discussed in Section V. While generalizing the MVC for the web – in the conceptual level – we ignore the facts that the web applications are partition- and technology-dependent. The preparation of the abstract version of MVC are discussed below.

1) Modules and Their Association

Model: The Model contains both the data structures, which hold the data at runtime, and the business logic to process these data. The data structures could be variables, objects, etc. and the data stored in these structures could be user's inputs or data read from any source like a file, a database, etc. or fetched/received from an external source like a web service, etc. The business logic could contain the code for reading data from external sources or from users as inputs, processing them and saving for persistence as needed in files or databases, etc. The Model is responsible for performing Create, Read, Update, and Delete (CRUD) operations on data sources; and any other business-related processing – based on the requests of the View/Controller – and produce the results back to the View/Controller.

View: The web pages are the GUIs of the web-based systems, and they can be seen as the Views. They may display some static content and/or dynamically generated non-data-related contents such as UI components and/or information (processed data) based on the state of the Model. The users interact with the Views by means of the inputting data using input devices – mainly the keyboard and the mouse – and receiving information, which is visually rendered on the GUI, and/or generated as audio and/or multimedia. Additionally, the outputs also could be files for downloading via the Views.

Controller: Controller is a very controversial module, which is interpreted, designed, and developed in many different ways by different experts. We would like to go alone with the original concept of the classic MVC, considering the Controller as an interface between the View, Model, and the input devices, which provides the event-handling and mediation between the View and the Model [22]. The Controller is an active module, which mediates between the passive View and Model, performing the event handling.

Association between modules: We would like to align to the idea of considering the Model as a single component [5]. There can be multiple classes in the Model according to the scenario; and there can be multiple Views and Controllers as in classic MVC. Each View is closely associated with a unique Controller and vice versa, and they maintain a tight coupling [16]. One View-Controller couple may associate with one or more classes in the Model, and a class in Model could be associated with one or more View-Controller couple. This association is illustrated in figure 6.

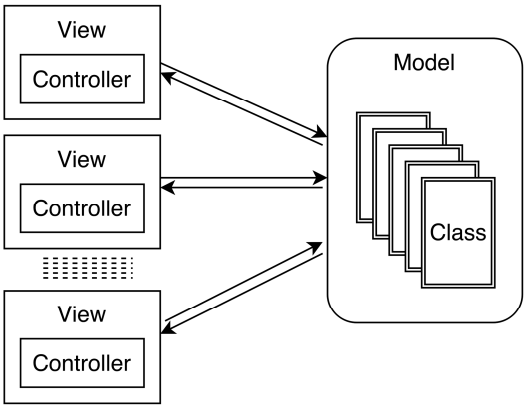

Figure 6. Association between modules in Abstract Web-MVC

2) Communication and Basic Interaction Cycle

We modify the classic MVC slightly as shown in figure 7, to make it less complex and easier to be adopted into the web applications.

The user interacts with the View(s) by inserting inputs and/or triggering events associated with the UI elements using input devices like keyboard or mouse. For example, typing text in a textbox, selecting an item from a list, clicking on a button, etc. The View fires the events, then the respective event-handler in the Controller will be invoked, who knows the action to be taken. The Controller will read data from the View if needed, and prepare/format data, to send to the Model for processing, and send the data along with commands to the Model. Model processes the data as commanded by the Controller and sends the response back to the Controller. Then, the Controller may process/format and prepare data further to be displayed on the View, and render them as information on the View.

Sometimes, View(s) may need to render information on GUI in the initial loading. In such situations, View(s) may communicate directly with the Model, requesting for Model's state. The Model may respond to such requests by rendering the necessary information on the UI directly without interacting with the Controller, and this entire communication will be done in the server.

V. Deriving the Balance Abstract Web-MVC

The core of this paper is to propose and introduce an architectural style based on an abstract implementation of MVC. This section presents the derivation of the proposed style, based on the abstract MVC version discussed in the previous section.

A. Requirements for BAW-MVC style

It is important to identify the requirements, which can be utilized as constraints and architectural properties to be satisfied by the end result, in the direction of deriving an architectural style. Based on the knowledge gain from the literature survey and empirical evidence gained through experiments, we have identified the requirements below, to be satisfied by and abstract MVC based style for the web applications development.

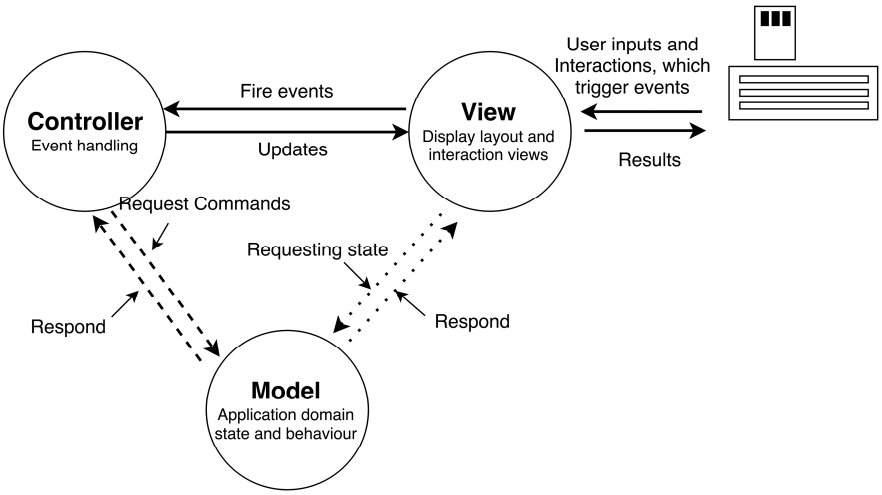

Figure 7. Proposed modifications in MVC architecture for web adoption

1) Functional requirements

Considering the constraints to be utilized for the derivation of the proposed style, the functional requirements are set as below.

FR1. Multi-tier expansion supportive Client-Server partitioning: The proposed architectural style, should be based on basic Client-Server architecture, and should support expansions up to multi-tier architecture or SOA based larger systems, by adding new layers to the basic model. The property, which is expected to be satisfied by the FR1 is the Extensibility of the system, and additionally Evolvability, Portability, and Adoptability also could be provided.

FR2. **MVC based modularization**: Proposed style should provide a good modularization of the features, based on MVC pattern, satisfying the modularization expectations. Simplicity is the main property to be satisfied via this requirement, however, it will ensure the Scalability, Modifiability, Reusability, Portability, and Adoptability as well.

2) Non-functional requirements

The architectural properties presented in the overview section are utilized as non-functional requirements to be satisfied by the proposed style.

NFR1. Simplicity
NFR2. Evolvability
NFR3. Extensibility
NFR4. Customizability
NFR5. Configurability
NFR6. Reusability
NFR7. Performance
NFR8. Scalability

3) Technical requirements

Considering core requirement for the introduction of abstract MVC version for the web, the technical requirements are set as below.

TR1. **Platform independency**: The style should not be based on specific platform or environment including operating system, servers, and browser/non-browser client platform. This requirement is related to the portability.

TR2. **Development Technology independency**: The style should not be based on any language, framework, library, or any other specific technology (like JAVA in model 2 architecture), or specific technique (like iframes in dual-MVC). This ensures the ability to utilize the developers' usual tools and knowledge, minimizing the learning curve of use; therefore also increases the adoptability of the available TTs into the development when the proposed style is used.

TR3. **Low learning curve and easy adoptability**: Proposed style should incorporate a lower learning curve, hence easy adoption. This can be ensured by making the concept of the style more abstract; and also the learning curve can be lowered by incorporating the knowledge of well-known concepts into the proposed style.

B. Deriving the proposed style

This section presents the derivation of the proposed style, by adding constraints to the null style. The functional requirements are utilized to formulate the constraints and the non-functional requirements are considered as the architectural properties induced by the application of the constraints.

1) Start deriving the proposed style with null style

The derivation of the proposed style begins with the Null style, which represents an empty set of constraints. In the context of the research, the Null style is represented as a web-based application, in the domain of the web. Figure 8 illustrates the Null style to begin the derivation of the proposed style.

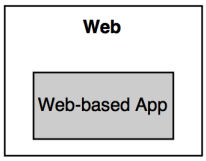

Figure 8. Null style for the RiWAArch style

2) Application of the Constraint 1 – Partitioning

This section applies the first constraint to the Null style, focusing on layering the system for Extensibility and Evolvability. Partitioning is done in two steps based on two criteria, to satisfy the FR1 completely and for the easiness of the evaluation. The first criterion focuses on the degree of the likeliness, the style offers towards supporting the

development of the standalone web-based applications based on C-S style. The second criterion indicates the degree of the support, the style offers for the extension of the standalone web-based application up to a multi-tier application.

For the partitioning constraint, as the candidate styles, C-S architecture, three-tier architecture, multi-tier architecture, and SOA were considered. Analyzing the features of these candidates, it was noted that the basic formalism, which all these architectural styles are based on is the C-S style. And also C-S style is capable of addressing the standalone web-based applications, which are consisted of only the client and the server layers. Furthermore, the C-S style can be extended by adding more layers; in other words, the other candidate styles can be seen as an extension of C-S style. Considering these facts, the C-S style was selected for partitioning.

a) Constraint 1 - Step 1: Client-Server Partitioning

Figure 9 illustrates the first step of partitioning by applying the C-S style to the Null style. This can be considered as the basic style of the standalone web-based applications. This style shows the client-node containing the client-component, and server-node, containing the server-component at runtime.

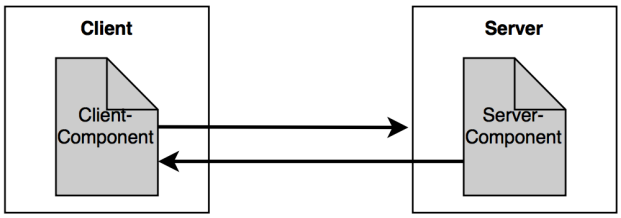

Figure 9. Client-Server partitioning for web-based applications

The Client-component sends the request to the web server, and the server directs the request to the proper server-components. Here the web server's processing components are not considered as a part of the system. The server-component processes the request and responds to the client-component with the result. For the browser-based web applications the response is mostly a web page, and can also be either an image or another type of file.

The client-component's communication is directed to the web server, instead directly to the server-component, since the target of the request might not be pointed to the exact server-component. The identification of the correct server-component and directing the request to it can be controlled by either using server configurations or a dedicated component in the server. However, this part is considered as a duty of the server itself, hence not included into the style. This will be re-discussed later, in the context of modularizing.

b) Constraint 1 - Step 2: Multi-tier Extension

The second step of the partitioning constraint specifies that the style should support expansion of the system to multi-tier or SOA by adding more layers. The layers other than the basic client and server layers are considered as external layers, which extend the web application to a multi-tier application. There can be layers dedicated for file(s) or databases for persistence, Web Services, or even Enterprise Service Bus (ESB) in the case of SOA. The server-side component can communicate with these external layers, and provide the client with necessary data/information. Figure 10 illustrates the Multi-tier Extensive Client-Server style (MECS), which satisfies the FR1 for the proposed style by providing functions: Client-Server based, and extensive up to multi-tier or SOA, by adding external layers with more functionalities.

Additionally, nowadays the client-components are capable of performing Cross-Origin Resource Sharing (CORS) [26], communicating with external servers. In such applications, the setting of the client-component from the original domain and the server-components from the external domains can be considered as a separate system; and this separate system can be applied with the MECS style – or any other style – accordingly.

3) Application of the Constraint 2 – Modularizing

This section presents the application of the second constraint of the style derivation process. Modularizing constraint is expected to induce the properties: Simplicity, and Modifiability, including Evolvability, Extensibility, Customizability, and Configurability. Additionally Performance, Testability, Maintainability, Reusability, and Scalability can also be expected.

The candidate MVC versions for this constraint are: Model2 [23], Dual-MVC [24], and the MVC web version introduced in previous section. Considering the TTs independence and the conceptual abstraction, the MVC web version was selected to be utilized for the proposed style.

This section discusses the adoption of MVC into the MECS style. The resulting style is named as the Abstract Web-MVC (AW-MVC), and it is extended to the style named Balanced Abstract Web-MVC (BAW-MVC) towards better performance of the style.

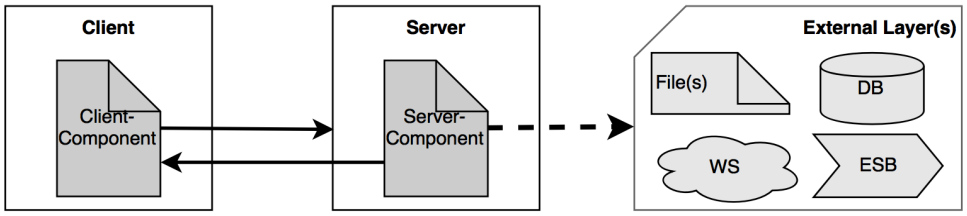

Figure 10. Multi-tier Extensive Client-Server style for the web-based applications

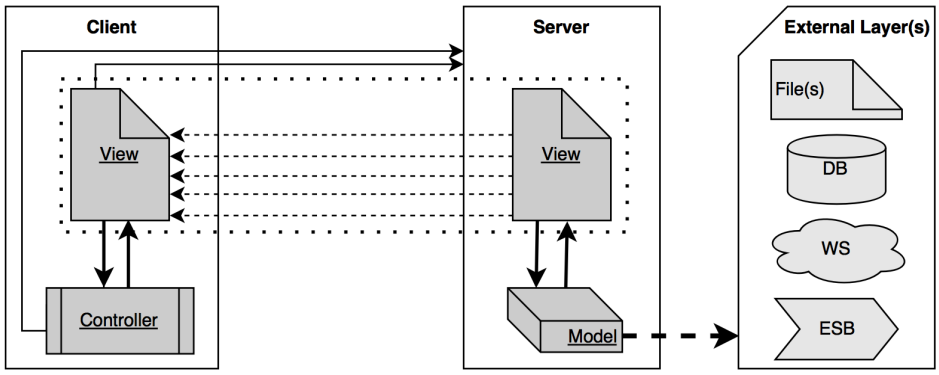

Figure 11. The Abstract Web-MVC style

a) Derive the Abstract Web-MVC Style

This section discusses the unison of the MECS and MVC web version, resulting the AW-MVC style, which is illustrated in figure 11.

This unison of the MECS style and the MVC web version in the AW-MVC style preserves the first constraint for the proposed style. The MVC module separation of the AW-MVC is discussed below.

Model: Primarily, the Model is placed in the server. When extensibility is needed, the extra layers are connected to the Model, and the Model contains the logic and the code for handling communication with these extra layers. An external layer could be a dedicated layer or even an ESB, which can handle more advanced communication with other external layers/components/nodes in favor of the Model. By including all the business logic into the Model, the NFRs Reusability, Testability, and Maintainability can be augmented.

View: The lifetime of the View is mainly reserved to the client-side; however, a View can contain some server-side code, which needs to be executed in the server, in the initial processing, before loading to the client. Therefore a View has an expanded lifetime between both the server and the client. This scenario is denoted by illustrating the View in both client and server layers. However, note that the Views in both partitions are grouped into a single module using a dotted line, to indicate the complete View component. The set of arrows from server-View to the client-View indicates that the same View is executed in the server – in the initial load, once the View is requested – and then loaded to the client and continues the rest of the lifetime in the client, till the next View is requested, loaded and the current View is replaced.

Controller: The idea of seeing the Controller as a request-handler in the server, like in model2 [23] or similar solutions [27] is highly discouraged in this paper. Since the GUI of the View runs and interacts with the user in the client-side, the events are also triggered in the client-side. Thus, the best place for the Controller is to be in the client-side. Unlike in the Dual-MVC [24] or the mixed client-side and server-side MVC [25], this work proposes to take the Controller entirely to the client-side, preserving the concept of the MVC web version introduced in the previous section.

The web server is natively included with the event-handling for the HTTP requests, hence it handles the requests implicitly. Therefore, the request handling is not a part of the web application; it is a part of the server. Considering this fact, the request-handling is not needed to be included into an architecture for the web applications. However, if needed, explicit request-handling techniques could be used within the web application, and in-depth discussions of these techniques are kept out of the scope of this section. In such scenarios, a dedicated component could be developed for explicit request-handling as shown in figure 12.

Since this request-handling component is optional, and also not related to either business logic or GUI-related events-handling, it is not considered as a part of the MVC. Furthermore, this request-handling component is based on some common knowledge and practice; hence, it is not explicitly included into the proposed style.

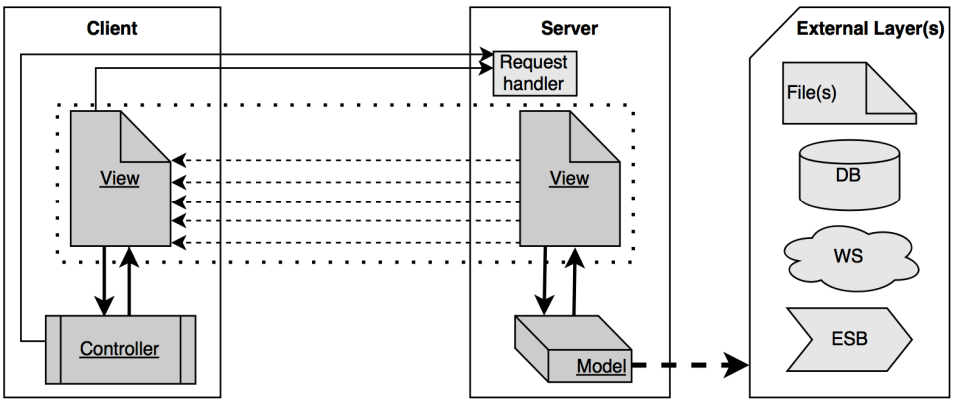

Figure 12. The Abstract Web-MVC with explicit request handling

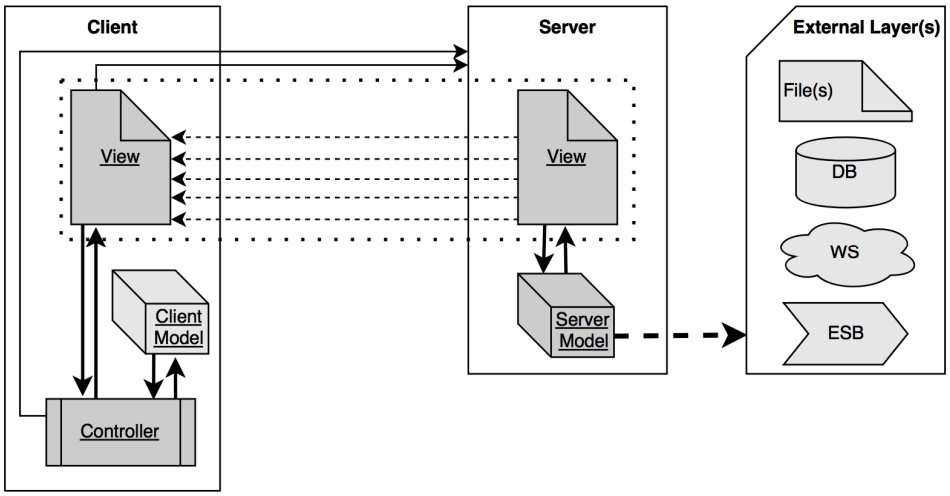

Figure 13. Balanced Abstract Web-MVC (BAW-MVC)

b) Derive the Balanced Abstract Web-MVC (BAW-MVC)

In Dual-MVC [24] and RIA MVC [25], the Model is partitioned between the server and the client, for better performance. This concept of partitioned Model can be integrated into the Abstract Web-MVC concept, with the help of client-side-Controller, as denoted in the MVC web version. It is not a must developing a client-Model, and it depends on the requirements of the scenario. However, the web-based applications engineering may find this concept useful, as the modern advanced web applications like Rich Internet Applications (RIAs) are expected to be highly interactive and perform better.

In the AW-MVC, even though the Controller is completely taken to the client-side, the business logic is still contained and executed in the Model, which is in the server. Hence, an AW-MVC based system as in figure 11, can be seen as a thin-client system. While experimenting, some features of the Model had been identified, which can be taken to the client-side, to form a client-Model. Figure 13 illustrates the architecture with a client-Model component. All the basic input validations – such as identification of blank text boxes, non-selected option lists, etc. – can be developed in this client-Model. Additionally, some basic logic and processing – such as calculation of the age based on the input date-of-birth – which are not critical or sensitive, also can be developed in this client-Model. The client-Model may additionally contain some data structures to temporarily hold data, till they are synchronized with the server-Model.

It is sensible to consider the client-Model as an essential component, and this paper suggests to consider the web applications with a client-Model – with the mentioned features – as a balanced system, instead of a thick-client. Engineers can decide to keep the Model completely in the server and make the system a thin-client application; or take more logic and/or data structures to the client, and make the system a thick-client application – referred to the balanced criteria. This version of the AW-MVC style, based on the aforementioned balanced web application criteria, we name it as the "Balanced Abstract Web-MVC" (BAW-MVC).

Note that the client-Model does not contain any direct communication links with other components than to the Controller. Both the View and the client-Model play a passive role in the client. The Controller behaves as the active component in the system, by handling the events triggered by the user on the View, and commanding the client-Model to act. The Controller may also receive the results from the client-Model, and at the end, the Controller can update the View with the results, by partial rendering the necessary sections of the GUI, without reloading the page. The client-Model decreases the need for communicating with the server, thus increases the Performance and Scalability.

VI. Evaluation of the BAW-MVC Style

This section evaluates the introduced BAW-MVC style in two steps. First, it evaluates the BAW-MVC itself, by indicating the induced properties – by the application of the constraints – using a derivation tree [8]. This helps to understand a list of properties the BAW-MVC style supports. The second step evaluates the BAW-MVC against the available similar styles, discussing how the induced properties are satisfied.

A. Architectural properties induces by the derivation of the BAW-MVC style

The derivation tree for the BAW-MVC style is illustrated in the figure 14. The application of the first constraint to the null style derives the MTCS style, inducing basic Simplicity via partitioning; and Extensibility and Evolvability via multitier expansion capability. Applying modulating to the null style, the MVC has being derived, and applying it to the MTCS style the AW-MVC has being derived, inducing Modifiability, Reusability, and improved Simplicity. The AW-MVC is further improved into the BAW-MVC, inducing Performance and Scalability.

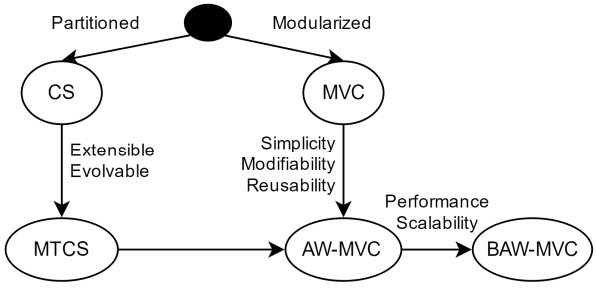

Figure 14. Derivation tree of the BAW-MVC style

B. Evaluate BAW-MVC against similar available styles

A tabular format of styles versus requirements [8], has been used to visualize the comparison between the styles. The symbols below are used to denote the comparative values of the effects of the evaluated facts.

(NA**)** – Not Applicable/Associated

(--) – Very less or no effect

(-) – Negative effect

(-+) – Moderate effect

(+) – Positive effect

(++) – Very high effect

Since the evaluated aspect are qualitative, the comparison is based on the knowledge gained through the literature and the empirical evidence. It should be noted that the values represented by the above symbols denotes the relative results among the artifacts compared, therefore not an indication of the perfection or imperfection. In the case of similar styles, it should be noted that they are evaluated in the perspective of this research; therefore even though they may show some positive/negative effects in their own scopes, they might not indicate similar effect in the perspective of this research.

Table 1 contains the evaluation of the BAW-MVC style as a contextualized comparison. The Mode column in the table indicates the application of the MVC is limited to either server-side (S) or client-side (C), or spread across server and client (S-C). The intended mode is the S-C, towards higher Simplicity, Performance, and Scalability.

Since the available style are reviewed in section III, they are not reviewed or discussed again here, compared to the BAW-MVC; instead, BAW-MVC is reviewed in detail in the direction of evaluation.

Additionally, the derived BAW-MVC is more suitable for browser-based applications, and the support for non-browser-based applications is not straight forward.

The mode of the BAW-MVC is (S-C), and helps in higher module separation across server and client, with lower coupling, which enriches the Simplicity of the system. The Simplicity delivered via lower coupling assists changes in components or adding new functionalities into the system in module level, without or with a minimal effect on the other components, increasing the Evolvability and the Extensibility. The Simplicity also supports customizing the components without impacting the non-targeted users, and also the post-deployment modifications to become easier, enriching the Customizability and Configurability. Furthermore, the higher separation of the modules increases the Testability in the module level, and also increases the Reusability, especially for the server-Model, which can be developed using OODD practices. Additionally, the client-Model lowers the communication with the server, which can increase the Performance and the Scalability of the system.

The BAW-MVC style provides an abstract implementation of MVC for the web applications, which is independent from platforms and TTs. The concept of the BAW-MVC delivers a firm abstract architectural formalism, hence helps one to realize the modularization of the web-based applications, with a minimal learning curve.

The main limitation of the BAW-MVC compared to the classic MVC can be mentioned as follows. The Model in the classic MVC is capable of broadcasting the state changes to all the related Views. In the web applications, there can be multiple users interacting with Views related to the same Model components; however, broadcasting the Model changes to all the running Views for all the connected users is not feasible in BAW-MVC. This can be achieved by incorporating data-push or push-simulation Rich Internet Application TTs.

TABLE I. EVALUATION OF BAW-MVC STYLE

Style	Mode	Simplicity	Evolvability	Extensibility	Customizability	Configurability	Testability	Reusability	Performance	Scalability	TR1. PI	TR2. TTsI	TR3. LLC
Model2	S	-	+	+	+	+	+	+	-	-	-	-	-
			+	+		+	+	+	-	-	-	-	+
Dual-MVC	S	-	-	-	-	-	-	-	+	-	-	-	-
	-		-	+				+		+			
	C												
BAW-MVC	**S**	+	+	+	+	+	+	+	+	+	+	+	+
	-												
	C	+	+	+		+	+	+	+	+	+	+	+

VII. Conclusion and Future Work

The introduced BAW-MVC provides an abstract conceptual implementation of MVC for the web applications, which does not depend on technologies. The concept of the BAW-MVC delivers a firm architectural formalism, and hence helps one to increase the realization of the web applications. We believe that it can be adopted in a wider range of development environments with a minimal learning curve.

BAW-MVC helps in higher module separation with lower coupling, which supports more for nonfunctional requirements related properties like Performance, Maintenance, Modifiability, etc.

The Model in the classic MVC is capable of broadcasting the state changes to all the Views related. In the web applications, there can be multiple View components – related to the same Model components – used by a user; or multiple users are interactive with the Views related to the same Model components; however, broadcasting the Model changes to all the running Views is not feasible in BAW-MVC, without incorporating the Rich Internet Application TTs.

The usage of the BAW-MVC – as discussed in this paper – has been introduced for the classical web applications, and not for the RIAs. Therefore, the rich user experience as in RIAs could not be gained using the BAW-MVC, but if the system is designed carefully, with the help of client-side Controller and Model, better user experience than the traditional web applications could be gained.

The adoption of TTs into this style should be experimented, presented, and discussed in a separate forum towards utilizing this style practically. In future, we expect to extend the BAW-MVC architecture, to offer an abstract architectural formalism for RIAs, to reduce the RIA engineering complexities and difficulties by increasing the realization.

References

[1] A. Ginige and S. Murugesan, "Web engineering: an introduction," *IEEE MultiMedia,* pp. 14-18, March 2001.

[2] A. Mesbah, A. v. Deursen and D. Roest, "Invariant-Based Automatic Testing of Modern Web Applications," *IEEE Transactions on Software Engineering,* vol. 38, no. 1, pp. 35-53, 2011.

[3] L. Shklar and R. Rosen, Web Application Architecture Principles, protocols and practices, England: John Wiley & Sons Ltd, 2003.

[4] N. R. Dissanayake and G. K. A. Dias, "Abstract concepts: A contemporary requirement for Rich Internet Applications engineering [Accepted]," in *9th International Research Conference of KDU (KDU-IRC 9)*, Colombo, Sri Lanka, 2016.

[5] J. Conallen, "Modeling Web Application Architectures with UML," Rational Software, 1999.

[6] A. Leff and J. Rayfield, "Web-Application Development Using the Model/View/Controller Design Pattern," 2001.

[7] M. D. Jacyntho, D. Schwabe and G. Rossi, "A SOFTWARE ARCHITECTURE FOR STRUCTURING COMPLEX WEB APPLICATIONS," *Journal of Web Engineering,* vol. 1, no. 1, 2002.

[8] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, 2000.

[9] A. Solutions, *The Importance of Software Architecture,* Architech Solutions, 2014.

[10] W3C, "Architecture of the World Wide Web, Volume One," 15 December 2004. [Online]. Available: http://www.w3.org/TR/webarch/. [Accessed 28 10 2015].

[11] D. Kegel, "The C10K problem," 05 02 2014. [Online]. Available: http://www.kegel.com/c10k.html. [Accessed 20 04 2015].

[12] R. Kazman, G. Abowd and M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures," in *Proceedings on 16th International Conference on Software Engineering*, 1994.

[13] R. Kazman, M. Klein and P. Clements, "ATAM: Method for Architecture Evaluation," Carnegie Mellon Software Engineering Institute, Pittsburgh, 2000.

[14] D. M. Selfa, M. Carrillo and M. d. R. Boone, "A Database and Web Application Based on MVC Architecture," in *Electronics, Communications and Computers, 2006. CONIELECOMP 2006. 16th International Conference*, 2006.

[15] M. Model, "Model View Controller History," [Online]. Available: http://c2.com/cgi/wiki?ModelViewControllerHistory. [Accessed 10 08 2015].

[16] S. Burbeck, Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC), Softsmarts, Incorporated, 1987.

[17] J. Deacon, *Model-View-Controller (MVC) Architecture,* 2009.

[18] S. Prakash, A. Kumar and R. B. Mishra, "MVC ARCHITECTURE DRIVEN DESIGN AND AGILE IMPLEMENTATION OFA WEB-BASED SOFTWARE SYSTEM," *International Journal of Software Engineering & Applications (IJSEA),* vol. 4, no. 6, pp. 13-28, 2013.

[19] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object Oriented Programming,* pp. 26-49, 1988.

[20] S. F. Morse and C. L. Anderson, "Introducing Application Design and SE Principles in Introductory CS Courses MVC Java Application Framework," *Journal of Computing Sciences in Colleges,* vol. 20, no. 2, pp. 190-201, 2004.

[21] W. Cui, L. Hung, L. J. Lang and J. Li, "The Research of PHP Development Framework Based on MVC Pattern," in *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, 2009.

[22] K. Brown, "Whatsa Controller Anyway," [Online]. Available: http://c2.com/cgi/wiki?WhatsaControllerAnyway. [Accessed 10 09 2015].

[23] Oracle, "About the Model 2 Versus Model 1 Architecture," [Online]. Available: http://download.oracle.com/otn_hosted_doc/jdeveloper/1012/developing_mvc_applications/adf_aboutmvc2.html. [Accessed 15 06 2015].

[24] K. Betz, A. Leff and J. Rayfield, "Developing Highly-Responsive User Interfaces with DHTML and Servlets," IBM, 1999.

[25] R. Morales-Chaparo, M. Linaje, J. C. Preciado and F. Sánchez-Figueroa, "MVC Web design patterns and Rich Internet Applications," in *Proceedings of the Jornadas de Ingeniería del Software y Bases de Datos*, 2007.

[26] W3C, "Cross-Origin Resource Sharing," 16 Jan 2014. [Online]. Available: https://www.w3.org/TR/cors/#resource-requests. [Accessed 26 10 2015].

[27] W. Cui, L. Hung, L. J. Lang and J. Li, "The Research of PHP Development Framework Based on MVC Pattern," in *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, Seoul, 2009.