ARCHITECTURAL DESIGN OF MODERN WEB APPLICATIONS

Lech MADEYSKI[*], Michał STOCHMIAŁEK[†]

Abstract. Architectural design is about decisions which influence characteristics of arising system e.g. maintainability or scalability. Existing architectural frameworks, like MVC or PCMEF, allow building well-structured applications as a result of minimizing dependences between the system modules. Authors of this paper analysed these frameworks in the web application context. MVC and PCMEF appeared to be inspirations for the new XWA (*eXtensible Web Architecture*) architectural framework combining strengths of both frameworks and incorporating the idea of continuations into a separated controller. Additionally the detailed description of practical implementation of XWA on e-Informatyka portal example and guidelines for building web applications especially based on Apache Cocoon similar technologies are presented.

1. Introduction

The choice of a proper system architecture is always a serious challenge in the software development process. Large web applications are not an exception. This choice has a great influence on the system maintainability or scalability.

The purpose of this paper is to propose an architectural framework that tries to respond to the architectural challenges of the web applications. The proposed XWA (*eXtensible Web Architecture)* framework is based on two well-known architectural frameworks: MVC and PCMEF but takes into consideration web application specificity. XWA is not only a theoretical phenomenon, but has its practical implementation (in the form of the e-Informatyka portal). There are also presented some guidelines for software developers interested in web applications development generally and Cocoon web application framework specifically.

Section 2 discusses in detail MVC, the classic architectural framework and also PCMEF – an interesting layered architecture. In section 3, both of them are confronted with

[*] Wroclaw University of Technology, Poland. E-mail: lech.madeyski@pwr.wroc.pl
[†] Wroclaw University of Technology, Poland. E-mail: misto@e-informatyka.pl

web applications' specificity. Section 4 presents and describes XWA web application framework and its practical implementation in detail. Section 5 summaries work results.

2. Quick review of approaches to architecture

In this section the MVC and PCMEF, architectural frameworks are described. Both of them were inspirations for the new XWA architectural framework.

2.1. MVC triad

The beginnings of MVC (*Model-View-Controller*) date back to late seventies and SmallTalk-80 language. The MVC paradigm has quickly become the core idea behind user interfaces in most of object-oriented languages, including SmallTalk [2][5]. The MVC paradigm is also a superb example of the separation of concerns idea. As shown on Figure 1 system classes are separated into three groups: model, view and controller. Semantics of each MVC element and rules of communication inside the triad are discussed below.

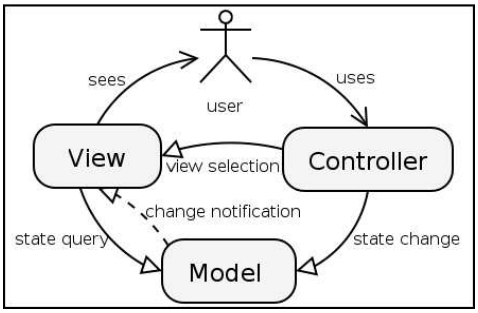

Figure 1. MVC classic architectural framework

The **Model** consists of static and dynamic parts of an application domain. The most important part of the model is the application logic with contained business logic. In other words, the model specifies the application data and behavior. The designer, while working on the model organization, should take into account that it has to be independent from a chosen presentation and user actions processing technology (*the model does not know anything about the view and controller*). *Change notification* is the only connection originating from the model. It is usually implemented using events or Observer design pattern.

The **View** is responsible for graphical or textual presentation of the model. The view implementation is strongly coupled with the model, because it should be aware of the specificity of presented data or operation results. Figure 1 illustrates this connection: the view collects the model state (*state query*) every time when it is notified about a change. On the other hand, the model is not coupled with the presentation technology, so the view can be reimplemented or even exchanged without any changes in the model implementation.

The main **Controller's** responsibility is to react to user actions (e.g. mouse button clicks) and map them to the model operations (*state change*) or view changes (*view selection*). The controller in conjunction with the view takes care of the *look and feel* of the application.

Unfortunately, the controller semantics is commonly misinterpreted [5]. It is sometimes mistakenly regarded as an element responsible for the behavior of application when at the same time the model is regarded as an element, which contains data only. This interpretation leads to a problematic tight coupling between the presentation layer and application logic. Changes in the first one require changes in the second one and vice versa. This is the main reason why the controller should not contain the application logic but only references to it.

The value of MVC is based on two essential rules. The first one is **separation of the model and presentation,** which allows to exchange user interface (e.g. provide two interfaces, graphical and batch mode). The second rule is **separation of the view and controller.** A typical example is providing two controllers (editable and not editable) for one view [5].

It must be noted that classic MVC form is often degenerated. This deformation is usually manifested in tightly coupled view and controller. For instance some SmallTalk versions and Swing library on the Java platform have this kind of design [4]. In Swing MVC framework is replaced by the *Model-Delegate* architecture.

The specificity of web applications causes that the classic form of MVC architectural framework should be taken into consideration. It will be discussed in section 3.1.

2.2. PCMEF architectural framework

Layered structures are often used in architectural design. They allow to group system classes into vertical hierarchy and to minimise package coupling. Those architectures can be called stable when changes in higher layers don't create a cascade of modifications in lower layers. This stability is possible when only downward dependencies are allowed (higher layers depend on lower layers). Sometimes for practical reasons, the architecture permits upward dependency realized by loose coupling implemented by, for instance, event processing.

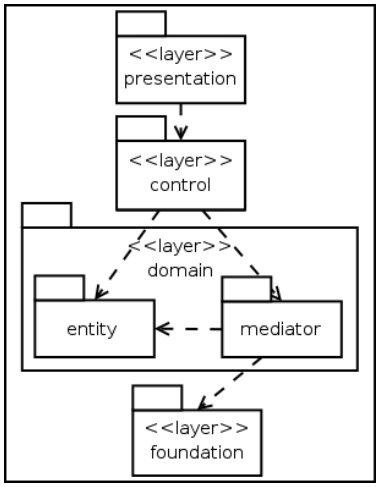

Figure 2. PCMEF framework

PCMEF is a layered architectural framework and is shown on Figure 2. It consists of four layers: `presentation`, `control`, `domain` and `foundation` [6]. The `domain` layer consists of two packages: `entity` and `mediator`.

The responsibility of `presentation` layer is the application presentation. It is usually composed of classes based on graphical user interface components. For example in the Java language, Swing or SWT components will be used. MVC equivalent to `presentation` layer will be the view strongly connected with the controller. A similar architecture (the *Model–Delegate* pattern) has been used in the Swing library.

The `control` layer is responsible for the processing of user requests from higher layer. It contains main application logic, computation algorithms or even user session maintaining.

The `entity` package, from the `domain` layer, contains business objects. Usually they are persistent and stored in some external data source (e.g. in database).

The `mediator` package, from the `domain` layer, mediate between `control` and `entity` packages and the `foundation` layer. Its introduction eliminates `entity` dependence on `foundation` package. In result it removes the necessity of business objects modification when the persistence technology is changed. It also gives possibility to separate the construction of queries to persistent data from the application logic included in the `control` layer.

The `foundation` layer is responsible for communication with data sources, such as databases, document repositories, web services or file systems.

3. Web application architecture

Every newborn idea has to face reality in which it has to come to live. In case of web applications the following has to be taken into consideration:

- HTTP protocol, the technological foundation of web applications, is stateless. Each command is executed independently, without any knowledge of the commands that came before it. Web application's responsibility is to keep the state between requests.
- Conversation between user and application can be initiated only by the user.
- Control flow of web applications is driven by user requests. It usually consists of complex sequence of interactions between the user and server.

3.1. MVC architectural framework and web applications

HTTP protocol specificity enforces view update to be driven by user requests. There is also no way to notify the view about the model changes[‡]. Required changes are delayed to the next user request processing. Therefore classic MVC model shown on Figure 1. has to be updated in such a way that *notification change* is removed.

3.2. PCMEF architectural framework and web applications

Web architecture design facets, discussed before, were concerning HTTP protocol. It is important to note, that behind HTML presentation there is often a complex business logic, a communication with external data sources or web services. Web application, to meet all requirements of maintainability and scalability, has to have a well-defined internal structure.

Unfortunately, MVC framework, discussed before, does not give any hints about this issue. This can lead to arise the network structure of objects dependences from different packages. Those structures can grow in exponential time and are difficult to control or maintain [6].

A verified solution to this problem is the introduction of a hierarchical structure. This is a recommended architecture in enterprise systems based on J2EE platform [1]. A good example of the hierarchical architecture is also, discussed before, PCMEF framework. It has well-defined semantics of each layer and at the same time, it is generic enough to be technology independent.

A naturally **arising question** is whether PCMEF framework is taking advantage of basic MVC's principles: the separation of the model and presentation and separation of the view and controller. The introduction of the `presentation` layer is probably motivated by the first MVC rule. Unfortunately, the second MVC rule was not expressed in PCMEF. Like in Swing library, the view and controller are strongly coupled. In case of desktop applications, this rule is often omitted for practical reasons. But in web applications separation of the view and controller is always treated as a good practice, and should be expressed in the architectural design.

[‡] There is a technological solution for this limitation. View can enforce periodical controller querying about model's changes.

4. XWA architectural framework proposal

XWA architectural framework is a variation of discussed before solutions, adapted to web applications specificity. XWA combines advantages of both frameworks, introducing a separation of the view and controller (according to MVC) and organizing model classes into the hierarchical structure (according to PCMEF). The proposal is shown on Figure 3.

XWA is a layered architecture with clearly separated MVC triad. It consists of six packages arranged into four-level hierarchy where higher layers depend on lower ones. Semantics of each package are described below.

The **View** package is responsible for the presentation of application. It is an exact equivalent of MVC view element. In web applications the **View** package consists of files describing the appearance of web pages. Depending on the technology used, this could be HTML page templates or JSP page in Model 2 architecture. But the most interesting solution seems to be the use of technologies based on XML. This introduces a new type of contract between layers based on XML. For example, the contract between the view and application logic is specified by an XML document scheme (expressed in DTD, XML Schema or Relax NG scheme definition languages).

The **Controller** package is responsible for processing users' actions. It calls logic included in lower layers. The main responsibility of the **Controller** is to separate HTTP protocol specificity from the application logic. It is responsible for controlling application control flow within a single interaction and sequence of interactions in case of more complex applications. XWA suggests using the continuations controller for these purposes, which is discussed in the next section. Another crucial controller's responsibility is to control the application view. The **Controller** may be realized by *Front Controller* or *Application Controller* design patterns [1].

The **Service** package is responsible for providing application services. It centralizes application logic contained in many business objects which requires access to data sources or web services (e.g. e–mail sending). XWA suggests using *Application Service* [1] or *Service Layer* [5] design patterns in this package.

The **Business Objects** package contains business objects which form application domain model using *Business Object* [1] or *Domain Model* [5] design patterns.

The **Mediator** package isolates **Business Objects** and **Service** package classes from the implementation of access to data sources, persistence mechanisms and web services. Classes from this package realize *Data Access Object* pattern [1].

The **Resource** package is responsible for low-level implementation of access to external resources.

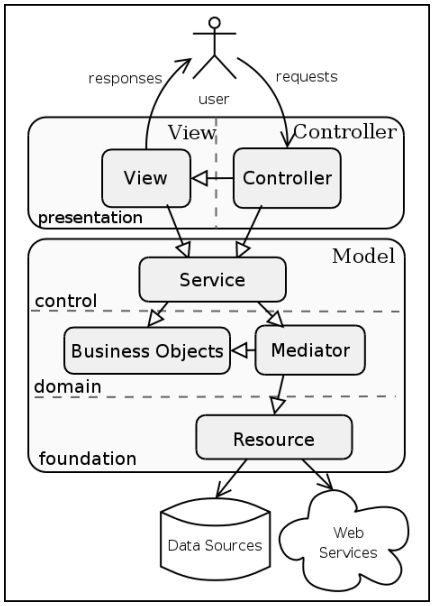

Figure 3. XWA architectural framework

4.1. XWA implementation on example of e-Informatyka architecture

The idea of e-Informatyka portal (www.e-informatyka.pl) appeared in June 2002. Its goal was to realize and promote a scientific journal with a strong support of electronic publication system. The portal was supposed to gather computer science community.

The academic character of the project, realization method and new approach to requirements specification based on XP (*eXtreme Programming*) methodology, which was worked out during the project, are described in [8]. At the moment, from the technical point of view, the project is advanced (e.g. trailer of the first issue of the journal is available).

Thanks to the academic character of the project and lack of constraints common to commercial projects, developers had a free hand in the choice of the latest technologies and in experiments with the system architecture. As a result XWA architecture framework was born.

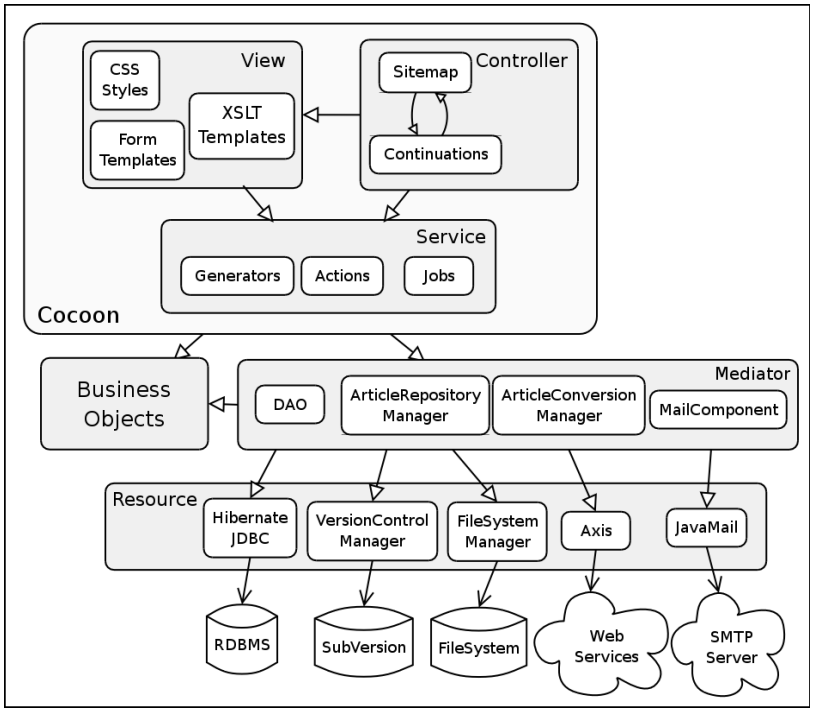

Figure 4. e-Informatyka portal architecture

4.1.1. Architecture of e-Informatyka portal

Technologies based on the Java language, XML and Apache Cocoon framework [10] are technological core of the e-Informatyka portal. A combination of these technologies offers a wide range of possibilities, but there is a lack of documents describing guidelines and good practices for designing Apache Cocoon based applications. Authors make an attempt to fill this gap.

Since the beginning the project has been based on the Apache Cocoon publication framework. But at the end of the year of 2003 the system architecture was significantly redesigned. XWA architectural framework consists of crucial elements of contemporary e-Informatyka architecture.

The core idea behind the Cocoon framework is a pipe architecture concept and therefore mapping it to the layered architecture is a challenge. The system architecture and organization of Cocoon components proposed by the authors is a result of their experiences. It also takes into consideration some suggestions of experts and consultants from the enterprise community.

The architecture of e-Informatyka portal is shown on Figure 4. Implementation details concerning each package are described in the next section.

4.1.2. The View and the Controller

XML-based technologies are intensively used in the presentation layer of e-Informatyka portal. The **View** package includes XML documents describing logical content of the portal and XSL (*Extensible Stylesheet Language*) stylesheets which describe transformations of XML documents (e.g. into HTML pages or PDF documents).

Main responsibility of the **Controller** package is to control the flow within interactions between the user and the system. The Controller calls actions from the application logic and activates a generation of the view.

The Controller maps the user requests to the application logic calls or to the view change. This mapping is usually based on requested URL address. It can also take into account the user profile or user agent type, which may be useful to personalize the web application [7].

The Cocoon **sitemap** provides the controller's functionality. The sitemap is an XML document which contains mapping of user requests to proper pipelines. Pipelines specify the view generation process by describing sequence of XML document transformations. During this process system actions (from the **Service** package) can be called.

The reuse of all pipeline elements is very simple. For instance, a transformer which inserts an article content into the portal layout, can be also used for inserting another document. More details about Cocoon's mechanisms and components can be found in [11][9].

In case of more complex web applications, which require an implementation of long interaction sequences between a user and the system, there is a need to store the state of interaction (current position in interaction sequence). The **continuations** idea and introduction of the **continuations controller** can extremely simplify implementation of that task and accelerate the development. Programmers do not have to deal with low level HTTP protocol mechanisms.

The Apache Cocoon provides a control flow mechanism which in fact is an implementation of the continuations controller [10]. A simple example illustrating the flow idea is shown on Figure 5. It is a user registration form with a few steps.

```
1. function register() {
2.   sendPageAndWait("user-form");
3.   var user = cocoon.request.get("user");
4.   sendPageAndWait("password-form");
5.   var password = cocoon.request.get("password");
6.   sendPageAndWait("register-confirm");
7.   AccountService.register(user, address, password);
8.   sendPage("register-successful");
9. }
```

Figure 5. Flowscript example

The course of conversation with the user consists of a few steps. The user fills in every form. He enters his personal data, then a password, and finally confirms all entered data. At the end the system informs about the registration success. It is important to note that every step is presented as a separate WWW page.

The key points of the example are `sendPageAndWait` function calls. They generate the page specified by the URL address, send it to the user and wait for his next request. The state of interaction is saved. When the user sends the next request, the script is continued.

4.1.3. The Model

The Avalon component container is the core technology in the Apache Cocoon. The architectural design of the e-Informatyka portal also uses it. All application logic from the **Service** package is based on the component interfaces. Classes from this package can be divided into three groups: generators, actions and jobs.

Generators are components, which produce XML document in the form of SAX events. Their main responsibility is providing data from business objects to the presentation layer (e.g. generating a list of recently uploaded articles). Generators hide details of business objects implementation (typical *Facade* [3]) and express business data in XML.

Actions are components, which implement application logic. They centralize calls to business logic, data source access or data persistence mechanisms concerning a single use case. Article upload action is a good example. It would consist of calls which add some information about the new article to database and stores article contents in an external repository. It must be emphasized that the action does not contain implementation of those operations. Their details are located in lower layers. Actions just group operations into transactions and could be seen as the implementation of *Application Service* pattern [1] but with fine-grained interfaces, which is a cause of Apache Cocoon's specificity.

Jobs are similar to actions. The difference lies in the activation method. Jobs are invoked not by a user request but by a built-in schedule (e.g. once every week). A good example is a job responsible for removing user accounts, which were not activated within a specific time frame.

The **Business Objects** package consists of classes, which implement *Business Object* design pattern [1]. In case of e-Informatyka portal, these classes represent mostly static aspects of the system, but in larger systems behavioral aspect may appear.

The **Mediator** package isolates access to a wide range of external data sources, which are implemented in **Resource** packages shown on Figure 4. Classes from this package usually implement the *Data Access Object* design pattern. A typical example is the persistence mechanism, which in case of the e-Informatyka portal is implemented by the Hibernate object/relational mapping system. Another example is the access to the content repository, which is implemented by the SubVersion versioning system.

5. Summary

The authors proposed the XWA architectural framework based on MVC and PCMEF. XWA takes into consideration the specificity of modern web applications. In particular:

- Semantics of each package and layer was precisely specified in web applications context, especially on the Java technology platform and Apache Cocoon framework.

- Clear separation of the view, controller and model was ensured by defining their responsibilities in the web applications context.
- A new element – the continuation controller was introduced to the architecture. Now web applications can be written like classic programs. This will simplify the implementation of complex interaction sequences between the user and the system.
- A new kind of contracts between layers was discussed. Those contracts are based on XML interfaces specified using XML documents schemas not classic interfaces known from e.g. the Java language. In case of XWA architecture a good example of such a contract is the interface between the application logic and view.

It is important to note, that XWA is designed as an extensible architecture, thanks to the organization into layers (taken from PCMEF), use of component technology (to realize the application logic) and XML technologies (in the presentation and integration tasks).

XWA and its practical implementation also fill a gap caused by the lack of documents describing good practices for the Apache Cocoon framework.

References

[1] Alur D., Crupi J., Malks D., *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall Ptr, 2003.

[2] Burbeck S., Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC), 1987, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[3] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns, *Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] Fowler A., A Swing Architecture Overview – The Inside Story on JFC Component Design, http://java.sun.com/products/jfc/tsc/articles/architecture/

[5] Fowler M., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

[6] Maciaszek L.A., Liong B.L., Bills S., *Practical Software Engineering*, A Case-Study Approach, Addison-Wesley, 2004.

[7] Madeyski L., New ideas of web applications development on example of e-informatyka.pl portal [Polish], in: E. Niedzielska, H. Dudycz, M. Dyczkowski (eds.), *Advanced Information Technologies for Management*, Research Papers No 955, Wroclaw University of Economics, Wroclaw, 2002, 425-437.

[8] Madeyski L., Kubasiak M., Agile Requirements Specification [Polish], in: Z. Huzar, Z. Mazur (eds.), *Problems and Methods of Software Engineering*, Wydawnictwa Naukowo-Techniczne, Warsaw, 2003, 53-68.

[9] Madeyski L., Mazur P., Modern internet applications [Polish], *Telenet Forum*, 5, 2003, 14-17.

[10] The Apache Cocoon Project, http://cocoon.apache.org

[11] Ziegler C., Langham M., *Cocoon: Building XML Applications*, New Riders, 2002.