Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals & Proceedings of DAAAM International 2011

MULTI-PURPOSE MODULAR SOFTWARE PLATFORM FOR RAPID DEVELOPMENT OF WEB APPLICATIONS

KRISTALY, D[ominic] M[ircea]; MORARU, S[orin] - A[urel] & GRIGORESCU, C[ostin] - M[arius]

Abstract: *This paper presents the architecture, techniques and mechanisms of a software platform designed to facilitate the rapid development of complex web applications based on Java technologies. The paper details, from the MVC's architectural pattern perspective, every element of the platform – from their functionalities to the technologies used in their development. Also, the structure and basic use of the platform's API, the request handling and the response generation are described.*
Key words: *web, platform, java, MVC*

1. INTRODUCTION

In the modern economy, the software companies are required to develop and release complex application faster than ever before. These applications must be accessible from different locations of the world, from different software environments, having different security policies. The most plausible solution to these requirements is a web application – a piece of software that can be accessed through a web browser. Most of the web applications developed by a company share some facilities, but even if the source code it's reused, some effort it's required to customize them for each project. For these reasons, working with a software platform that can offer these facilities "out of the box" can be a real advantage, helping the companies deliver their products faster, and more importantly – cheaper.

In general, the user interface of a web application is developed using web technologies (Hypertext Markup Language – *HTML*, Cascading Style Sheets – *CSS*, client-side scripts and server-side scripts) and can be accessed by the end-users through a web browser (*Mozilla Firefox, Google Chrome, Microsoft Internet Explorer etc*).

Most of the web applications developed by a company for distinct clients will share some facilities, such as:

- user management;
- rights management;
- session management;
- monitoring and logging (Mustica et al., 2008).

Usually, the companies reuse parts of the source code for these features in new projects, but a considerable effort it's needed to customize them to the new look-and-feel.

In order to speed up the development (maximizing the reusability) and, subsequently, the release of the product, a modular platform that incorporates all these common features and offering an application programming interface (*API*) and a flexible templating system for rapid module development represents a serious advantage.

This paper proposes a multi-purpose, modular platform based on Java technologies.

2. ARCHITECTURE AND TECHNOLOGIES

The architecture of the platform is derived from the MVC (*Model-View-Controller*) paradigm, providing high levels of reusability and maintainability (Avedal et al., 2001).

The MVC architectural pattern is used in the development of applications to separate the application logic (business logic) from the presentation layer (user interface). This separation allows independent development, testing and maintenance of each layer. (Wikipedia, 2011)

Each major software technology provider (as *Microsoft* and *Oracle*) offer specialized mechanisms for implementing the MVC architectural pattern. This paper will focus only on the technologies supplied by the Java EE (*Enterprise Edition*) platform (offered by *Oracle*) for web application development.

Being free/open-source, Java technologies ensure a low cost for the initial investment when implementing a new software system.

The diagram in Fig. 1 illustrates the technologies used to implement each layer of the architecture.

The *Controller* is implemented using the Servlet technology. This implies that the web application must run inside a Servlet engine or Web container. For the platform presented in this paper, IBM's Websphere Application Server – Community Edition (*WASCE*) it's considered.

To reduce the servlet's workload, HTTP filters are used to check the incoming requests. In this way inappropriate requests can be rejected before reaching the servlet. There are 4 filters used and a request must pass successfully through all of them:

- session filter: checks the validity of the current session;
- uploader filter: prepares specialized objects for requests that use file uploads; the *Apache FileUpload* library it's used to implement this feature;
- authorization filter: performs an authorization check of the current user against the rights registry;
- monitoring filter: performs monitoring tasks; it collaborates with a context listener to feed monitoring data into the platform's database

The initialization of the platform is made by a specialized function of the *CacheFactory* component, that it's called at the initialization of servlet's context by a context listener class.

The *CacheFactory* loads into memory specific data that is often required by the platform's core, such as modules list, functions list, profiles list and rights registry, in order to achieve higher running speeds and to reduce the workload of the RDBMS (*Relational Database Management System*).

The platform's smallest execution units are functions – these are instances of classes that implement the *IManager* interface, which forces each *Manager* class to provide a standardized method that will perform the atomic task of the function. The platform maintains a list of all installed functions and provides management facilities through the user interface.

Depending on the action specified by a request, the servlet will create an instance of the appropriate function (a *Manager* class) using Java's reflection mechanism.

A Manager class receives as input the request data and returns a vector containing data about the result of the specific operation performed by the function and information related to how the data must be displayed – template information.

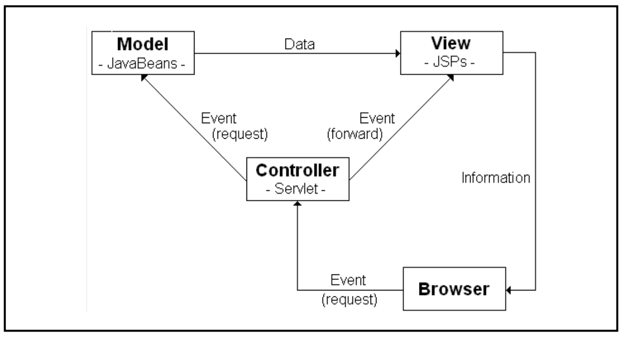

Fig. 1. MVC in Java EE context (Kristály, 2005)

All the access rights to functions are granted on user profiles. A user can be linked with one or more profiles and has the reunion of all the rights linked to the associated profiles.

One or more functions make up a module. The platform maintains a list of modules.

The user interface uses the information from the modules and functions lists to generate the platform's main menu.

The *Model* it's built using Java Beans. The data it's stored inside a MySQL database.

All database related operations are performed by an abstraction layer composed of DAO (*Data Access Object*) classes that hide MySQL specific handling from the rest of the platform's components (Taylor, 1999).

A web platform it's accessed by many users at the same time. The facilities modify or use data taken from tables inside one or more databases. This means that the platform's modules must connect and disconnect to the RDBMS often. These operations take longer time than a simple query, so the response speed could be inadequate for the purpose of the platform. To address this issue, a database pooling system it's used.

A database pooling system keeps a number of connections to the database opened at all times; when a module needs a connection, it receives one from the pool and the pooling system marks the connection as *busy*. When the module finishes its work it releases the connection back to the pool, which marks it as *free* (Taylor, 1999).

WASCE offers a database pooling system that can be managed through its administrative console. The database pools are indexed inside the JNDI (*Java Naming and Directory Interface*) space, from where they are accessed and used by the DAO classes.

The *View* uses JSP (*Java Server Pages*) to implement the user interface. JSP files can contain HTML, CSS, JavaScript and Java source code. A good-practice rule it's to eliminate all Java source code from the JSPs. For this purpose, *Expression Language* (EL) and *Java Standard Tag Library* (JSTL) is used (Spielman, 2004).

The platform implements a templating system to preserve the unity of the look-and-feel across all modules and functions. A task of the controller it's to send the result data and the template information obtained from a to a JSP master file template that builds the response that, usually, is a HTML page. The programmer has the option to bypass the JSP master file and leave the rendering process to another JSP file. In this way, the platform can have different looks for each module or even each function.

3. REQUEST HANDLING AND RESPONSE GENERATION

Fig. 2 presents the platform's components and the interactions between them.

A request it's verified by the 4 filters before reaching the *Controller* servlet. If any test performed by the filters it's failed by the request, it is dropped and the user will be routed to the homepage or login (to repeat the authentication process).

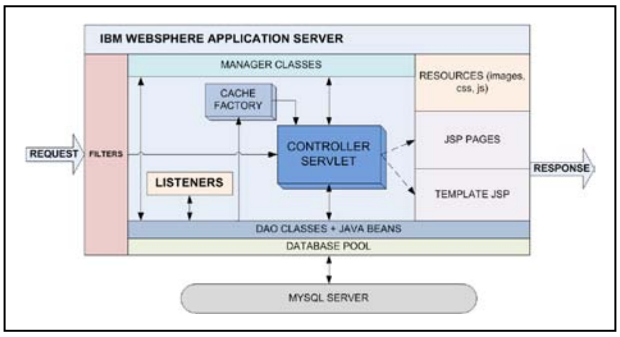

Fig. 2. Block diagram of the platform's core

The servlet examines the action specified in the request and identifies the corresponding function. Once the function is found, it prepares the input data, instantiates the *Manager* class and launches the function's code. The *Manager* object can access the data found in the database using the DAO classes.

After the function finishes its task, it returns to the controller the result data and the template information as a Java bean instance.

The servlet selects, according to the template information, the JSP page that it's in charge of rendering the response to be returned back to the client, and sends it the result data for processing.

The response it's forwarded to the master JSP file of the template (if this is not inhibited explicitly by the function) which generates the response that it's sent back to the user's browser.

Before every step, consistency data checks are made to ensure that all requirements are met for that step, eliminating all unrecoverable errors.

4. CONCLUSION

The developed platform can speed up the process of creating complex web application by offering built-in management facilities and a clear and simple API that requires only basic Java programming competencies.

The software systems based on this platform are less expensive and require less people, with less experience to develop them.

All the optimizations made for this platform recommend it for work environments with many users and frequent requests.

By using Java technologies and tools, the initial investment for deploying a new software platform is greatly reduced, and because of Java's dependability, the maintenance costs of this type of system are minimal.

5. REFERENCES

Avedal, K. et al. (2001), *Professional JSP - 2nd Edition*, Wrox Press Ltd., ISBN 978-1861004956, USA

Kristály, D.M.; Crăciun, A.V.; Pelcz & A.; Trican I. (2005). MVC Architecture in web applications developement, *Proceedings of the 14th ELECTRONICS*, Sozopol, Bulgaria, ISBN 954-438-520-7, Book 4, pp. 73-77, Tech. Univ. of Sofia & Tech. Univ. of Delft, Sofia

Mustica, M.; Moraru, G.; Grigorescu, C. & Kristály D.M. (2008). Portal-type applications for web campus, *Proceedings of "PETRA 2008"*, ACM (Association for Computer Machinery), Athens

Spielman, S. (2004). *JSTL: Practical Guide for JSP Programmers*, Morgan Kaufmann, ISBN 978-0126567557, USA

Taylor, A. (1999). *JDBC Developer's Resource*, Ed. Prentice Hall, ISBN 978-0139016615, USA

*** (2011) http://www.wikipedia.org/ - Wikipedia, Model-View- Controller, *Accessed on: 2011-09-21*