The use of Architectural Frameworks in Enterprise Web-Development.

Kenneth Rohde Christiansen

Department of Mathematics and Computing Science
Rijksuniversiteit Groningen
Blauwborgje 3
NL-9747 AC Groningen

k.r.christiansen@student.rug.nl / Kenneth.christiansen@gmail.com

Abstract:

Some of the main objectives of software engineering are to improve software quality, reduce costs, as well as facilitate easy maintenance and evolution of the product in mind. Code re-use is a way to fulfil these objectives as it reduces costs as less new code has to be developed and it improves software quality as the code will be more widely tested. Additionally, it helps with maintainability and product evolution, as writing re-usable code requires the developer to code more modular.

In web development this is in particular apparent as enterprise applications have a great deal in common, such as the need for logging, automated testing and data persistence. For this reason many component-frameworks have been developed and are currently in use. In order to improve maintainability, reuse and analysis of these software systems the frameworks need to be modeled quite abstractly and provide mechanics for converting these models into actual implementations.

In this essay we will see to what extend this is possible and whether there exists practical architectural frameworks for developing enterprise web applications.

Keywords: Architectural Frameworks, Software Development, Enterprise Web-Development.

1. Introduction

Code re-use is an important aspect of software engineering as it improves software quality, reduces costs and facilitates easy maintenance and product evolution. When code is re-used it receives more testing which generally improves its quality and when you re-use code you often spend considerable less time than writing the code from scratch, which leads to a cost saving. Additionally, when writing code with re-use in mind, the developer has to write more modularly by encapsulating code belonging to a specific domain. This helps with maintainability and product evolution.

Enterprise Web Applications share many properties and are thus very good candidates for code re-use. This is also the reason why most enterprises base their products on so-called component-frameworks.

The contribution of this paper is a look at to what extend it is possible to use architectural frameworks for web development. Section 2 and 3 introduces Software and Component Frameworks and explain how they are advantageous for web development. Section 4 introduces Architectural Frameworks and gives a small example how the architectural descriptions can relate to implementation. It also states some of the major differences between Architectural and Component Frameworks. Section 5 described what kind of framework systems that are currently used for web development and section 6 introduces some of the most current research in the area. The paper ends with a conclusion summing up the main points.

2. Software Frameworks

In software development a framework is a support structure that can be used to organize and develop similar software projects. Such a framework often include class libraries and additional programs that can help you develop your project, by for instance gluing together different components. Instead of the code being in charge of calling upon different libraries to do the work you want, the framework itself is in charge. This idea is thus that you start our with something generic that works right away and then it is your job overriding and extending the framework to archive your desired goals.

In the age of web development many developers have realized that many enterprise applications have a great deal in common, such as the need for logging, automated testing, and data persistence, and that there thus is a great need for such frameworks.

Developing these services from scratch every time would be an enormous drain on resources and it will be error phone as well. Due to this fact, web development has turned into so-called component-based development. The benefits of using components are many: The components provide a reduced risk and cost of project deployment as the components consist of well-tested units of code. Using components also results in a faster and improved rapid development, as development more consists of gluing different components together. The use also has the advantage that the staff utilization can be spread across different projects, each working in their area of expertise. Prefabricated components can also be bought from other vendors who might have more experience developing for the given problem domain.

3. Component-frameworks

Components from different vendors do not necessarily integrate very well. In fact, our own experience is that they often do not. The problems of using components are that they seldom cover your needs by 100% and customizing the components is often not an easy ride. Another problem is that the components are often not interchangeable due to different APIs, and thus require some kind of encapsulation. The encapsulation is also required in order to only expose the needed parts of the component as well as not making the component seem alien to your software product. A consistent, concise API in a software product is important for maintainability reasons and work has been done of solving some of these problems at their heart [1] [2].

Many of the mentioned problems are solved to some extend by the development of component-frameworks, thus collections of non-overlapping, throughout-tested components with consistent API and comprehensive documentation.

A component in a component-framework is not only a reusable, independent encapsulated unit of functionality, but it often has a discoverable interface that can be used by other components and software tools. This makes it possible to easily combine and interchange components with each others. This is very important for you as a company as it is possible to abstract yourself from the use of specific tools. For instance using a different database might be as easy as replacing a component with another with a similar interface.

A component framework is a reusable design in that it defined the interfaces implemented by the components, their control flow as well as the contract between the system and the components. It is a mechanism that makes it possible for the developer to decompose the application into a set of interacting prefabricated building blocks that can be used, extended and customized to fit a given application.

4. Architectural frameworks

This idea of frameworks can be extended to architectures, by combining suitable architectural design patterns and techniques and language constructs with predefined elements.

An architectural framework is, according to TOGAF [3] „*a tool which can be used for developing a broad range of different architectures. It should describe a method for designing an information system in terms of building blocks, and for showing how the building blocks fit together. It should contain a set of tools and provide a common vocabulary. It should also include a list of recommended standards and compliant products that can be used to implement these building blocks*.”

A bit formally we can define such an architectural framework as collection of languages for architectural description. These languages needs to have a well-defined semantic and there also needs to be a collection of rules, properties and invariants to govern their use. The forming/constituting elements of an architectural description also have a description, as well as a definition on what forms an implementation of such a description.

Though not required, it is possible to also provide a collection of predefined elements, standards, tools etc.

It is clear that an architectural framework share quite some aspects with the component-frameworks. The main differences of the two are that architectural frameworks are less implementation oriented and more descriptive, which should allow for formal analysis. An architectural framework also has to define what the architecture of a combined system is and it may and may not provide an explicit component model.

4.2 An example Architectural Framework

To get a better idea how these architectural descriptions can look and how they can relate to the implementation we provide a little example of a non-existing architectural framework referred to as FW1. The FW1 framework is a quite simple architectural framework that allows software systems to be built by combining various components which are encapsulated and do not share objects or state information.

The framework uses a graphical notation to describe the components as well as the topology of the architecture. The interfaces are described by the use of Java, and the Java execution model is used as a way for describing the component behavior.

The architectural description of a system based on FW1 consists of component interface descriptions, a topology description as well as an initialization description. The interface description is based on the graphical notation and the java interface language and indirectly uses the java execution model a way to describe the component behavior. The topology describes the components in use as well as how the instances of these components are linked together. The initialization description simply describes the order in which the component instances are started. An implementation of such a system is then simply an implementation of each interface, the initialization of the component instances as well as the linking of them.

FW1 shows that the descriptions can be related to current programming languages and graphical description tools as UML [13], and we will probably expect something like this in a practical architectural framework.

It is obvious that our sample framework has various drawbacks though. The major problems are that FW1 is highly linked to a particular programming language and that the model is not very abstract which makes it harder analyzing and reasoning about the interfaces and system properties.

4.3 Architectural Description Languages

In recent years research has been done on developing so-called Architectural Description Language (ADLs) with the advantage of having the ability to rigorously specifying an architecture so that it can be analyzed. Though, these ADLs are of great help, they are in practice not sufficient to capture all the issues of software architectures, which results in the fact that there does not exist usable architectural frameworks that satisfy all the criteria we set out with.

5. The current situation in Web Application Frameworks

When we look at the current frameworks for developing web applications such as Struts [4], Tapestry [5], Cocoon [6] Expresso [7] and WebObjects [9] it becomes clear that most, if not all, are component frameworks providing common building blocks for developing web applications. The building blocks are glued together using architectural design patterns, such as the Model-View-Controller II.

The design approach of the MVC design pattern is to clearly separate the key-components of the architecture into three components: The Model, which contains data required by the application; the View, which manage the presentation of the data, and the Controller which acts as intermediate between client and data. Thus, behavior, presentation and control are separated which decreases code duplication and decentralizes control which makes the application more easily modifiable.

5.2 The development of Web Development

Web development has gone though some stages during the years. In the beginning web development were focused on the logic of dynamic page generation. HTML and SQL commands were thus embedded in standard programming languages such as C++, Perl, and Java.

The next step was embedding data access in logic which then again is embedded in the interface. Instead of embedding HTML into code, special programming languages were used which instead consisted of HTML with embedded code. This made it possible for non-coders to alter the interface provided that they did not touch the embedded code globs.

Unfortunately, the above methods encourage code duplications and not code-reuse. Using architectural design patterns as the Model-View-Controller web development frameworks aroused that made it possible to clearly separate the data access and the logic from the interface. Detailed information on how this is accomplished can be found in [8].

Persistence frameworks have also aroused that made it possible to separate the data access from the business logic. Most of these frameworks makes it possible to access database tables as were they standard programming language objects without worrying about updating the tables in when the objects changes. These persistence frameworks most often also support a query language that is independent of the database used, due to the fact that most databases divagate in their implementation of SQL and that some of the persistence frameworks support non-SQL data sources such as LDAP, and flat files [9] [10]. These frameworks are often used in combination with the above MVC-frameworks or even included.

5.3 The Web Application Frameworks

Though, the mentioned web application frameworks are based on architectural design patterns and some providing pre-factored building blocks which allows for good, abstract architectural software design, they do not live up to our specification of an architectural framework.

The frameworks are not based on architectural descriptions but are instead normal component-frameworks following good architectural design patterns/practice. This can be a bit confusing as at least one of these [7] is marketed as an Architectural Frameworks.

5.4 Current architectural practice

Though the above frameworks are not directly based on architectural descriptions it does not mean that the developers using them are not producing architectural descriptions in form of design documents and UML diagrams. The problem is that mapping these design documents into implementation is often quite time consuming and implementers often tend to forget the advantages of such good modeling practices. What often happens is that the relationship between the design documents and the implementation is lost and it becomes hard to trace back design decisions, which seriously weakens the maintainability and support for product evolution.

Having architectural descriptions and reflecting this in the implementation has many advantages as the company can improve communication with the stakeholders, due to the fact that the architecture represents a high-level abstraction to the actual system; something that can form a base for mutual understanding and consensus. It is also possible to represent early design decisions which most often represent the most important design decisions with respect to the entire system. There are many other advantages and the most important ones can be found in [11].

Even though, there are several good reasons for using architectural descriptions, many web developers do so in a not very well-defined way due to the fact that the companies lack a good Architectural Framework for developing enterprise web applications. The most defined architectural description used is the use of UML for defining the use-cases and component/class interfaces and relations. Good tools exist for creating classes and modifying existing ones with the use of a graphical UML editor such as Borland Together [12] and IBM Rational XDE [21] which builds on the popular open source project, Eclipse [22].

6. Current research

As stated before architecture is very important as it promotes communication among stakeholders, captures early design aspects and is reusable in nature due to the abstraction level.

The inventor of UML, the Object Management Group, have set out to provide a common architectural framework for object oriented applications based on widely available interface specifications, and are currently working on a Model Driven Architecture (MDA) [14] specification that advocates shifting to model-oriented paradigms.

This approach deals with model description (architectural description), model exchange and model evolution and is based on standards such as the UML that acts as modeling language, Object Constrain Language, OCL [17] for describing expressions on UML models, XML Metadata Interchange (XMI) [15] for securing interoperability between Computer-Aided Software Engineering (CASE) tools, as well as Meta Object Facility (MOF) Query / Views / Transformations [16] for managing model evolution.

The central issue in MDE approaches is so-called model transformations. [18] [19] There are two different approaches for model transformations; textual and visual ones. The textual ones are based on standard programming languages as well as OCL extensions for UML. OCL makes it possible to make a class diagram refined enough to provide all the relevant aspects of a specification, by describing additional constraints about the objects in the model. OCL is especially developed to do this in an unambiguous way which is not possible in natural language. The visual approaches are relying on graph transformation techniques in order to accomplish the same.

Even though there are being done a lot of research in this area there are, according to our knowledge, still no mature tools available that automatically link the implementation of architectural descriptions found in the MDA “architectural framework” with the actual descriptions. This means that MDA has not really been adopted in practice. Work is being done in order to remedy this situation such as the FIDJI CASE tool [20].

7. Conclusion

We started out with the question to what extend it is possible to use architectural frameworks for developing enterprise web solutions. Evaluating the frameworks currently available for enterprise web developers it is clear that none of these are Architectural Frameworks per se, in the sense that they for instance provide no architectural description language. What they provide instead is great component-frameworks build upon proven and non-proven architectural design patterns.

Though these frameworks have seriously helped web developers design better maintainable systems, developers are still faced with the problem that there is no direct relationship between the design documents and the implementation, which seriously weakens the maintainability and support for product evolution.

This does not mean that web developers cannot use good architectural practice when developing enterprise solutions and many companies does so. But due to the fact that there is no direct link between design documents and implementation, developers, including ourselves, unfortunately tend to forget the importance of using architectural descriptions throughout the design process.

The Object Management Group are working on many specifications that with time will allow companies to more easily use architectural frameworks for designing their products and for guiding their implementation. Unfortunately, making this a practical solution there need to be good tool support and making good tools will require much feedback from the people using these. Another problem that is slowing down the adoption of these specifications is that the learning process is hard. Reading though some of the specifications shows that there are a lot to learn and you really only benefit when you know the practices to a great extend. This will be improved in the future as tools arise together with good teaching material.

8. Acknowledgement

We would like to thanks our teacher Dieter Hammer and the teacher assistants Mugur Jonita and Frank Kramer for an interesting and worthwhile course.

9. References

[1] Jan Bosch (1998), *Superimposition: A component adaptation technique*

[2] Kenneth Rohde Christiansen, Niek Oost (2005), *A look at Programming Methods for solving problems of current Software Development*. In the proceedings for StudColl. 2005, University of Groningen

[3] The Open Group, *TOGAF; the Open Group Architecture Framework*, http://www.opengroup.org/architecture/togaf/

[4] Apache Software Foundation, *Jakarta Struts*, http://struts.apache.org/

[5] Apache Software Foundation, *Jakarta Tapestry*, http://jakarta.apache.org/tapestry/

[6] Apache Software Foundation, *Apache Cocoon*, http://cocoon.apache.org/

[7] JCorporate, *Expresso*, http://www.jcorporate.com/html/products/expresso.html

[8] Sun Microsystems, *Designing Enterprise Applications with the J2EE Platform, Second Edition*, http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/index.html

[9] Red Shed Software (2002) *Introduction to WebObjects 5*, http://rentzsch.com/webobjects/introTo5

[10] JBoss, *Hibernate: Relational Persistence for Idiomatic Java*, http://www.hibernate.org/

[11] L. Bass, P. Clemens, R. Kazman (1998), *Software Architecture in Practise*, Addison Wesley

[12] Borland *Together*, http://www.borland.com/us/products/together/index.html

[13] Object Management Group (2003) *Unified Modelling Language (UML)*, version 1.5 http://www.omg.org/technology/documents/formal/uml.htm

[14] Object Management Group (2003), *MDA Guide Version 1.0.1*

[15] Object Management Group, *XML Metadata Interchange Specification*

[16] Object Management Group (2002), *Meta-Object Facility (MOF™)*, version 1.4 http://www.omg.org/technology/documents/formal/mof.htm

[17] Object Management Group (2003), *UML 2.0 OCL Final Adopted specification* http://www.omg.org/cgi-bin/doc?ptc/2003-10-14

[18] A. Gerber, M. Lawley, K. Raymond, J. Steel et al. (2002) *Transformation: The Missing Link of MDA*, 1st International Conference on Graph Transformation (ICGT), Barcelona, Spain, pp. 90-105, 2002

[19] D. Pollet, D. Vojtisek, J.M. Jézéquel (2002), *OCL as a Core UML Transformation Language* WITUML 2002, Malaga, Spain

[20] Nicolas Guelfi, Gilles Perrouin, *Using Model Transformation and Architectural Frameworks to Support the Software.* Development Process: The FIDJI Approach.University of Luxembourg

[21] IBM *Rational Rose XDE*, http://www-130.ibm.com/developerworks/rational/products/xde/

[22] The Eclipse Project, *Eclipse IDE*, http://www.eclipse.org/